

NOVEL USAGE OF DEEP LEARNING AND
HIGH-PERFORMANCE COMPUTING IN
LONG-BASELINE NEUTRINO OSCILLATION
EXPERIMENTS

by

SAÚL ALONSO-MONSALVE

A dissertation submitted in partial fulfilment of the requirements for
the degree of Doctor of Philosophy in

Computer Science and Technology

Universidad Carlos III de Madrid

Supervisors:

Dr. Leigh Howard Whitehead
Prof. Dr. Félix García Carballeira

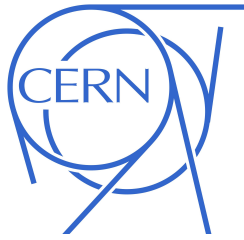
February 2021



This thesis is distributed under license “Creative Commons **Attribution – Non Commercial – Non Derivatives**”.



*“The saddest aspect of life right now is that science gathers
knowledge faster than society gathers wisdom”*
- Isaac Asimov (1920–1992).



This PhD dissertation has been mostly developed at CERN, supported by the CERN Doctoral Student Programme (from February 1, 2018 to January 31, 2021).



This PhD dissertation has been partially supported by the Spanish Ministry of Education, Culture and Sports under an FPU fellowship with identifier FPU16/01957 (from October 1, 2017 to January 31, 2018).



Part of this PhD dissertation has been developed during a research stay at the Cavendish Laboratory, University of Cambridge, UK (March 2019).

Please feel free to contact the author at: saul.alonso.monsalve@cern.ch.

I would like to dedicate this thesis to both my sister and my mother,
for all their support despite the distance.

I love you.

Acknowledgements

Many people come to my mind when I sit down and reflect upon the last three and a half years. First of all, I would like to acknowledge the only person I have been in touch with every single day since I arrived at CERN in February 2018, Leigh Whitehead, for becoming not only an outstanding thesis supervisor but also a friend. You are the person from whom I have learnt the most. Also, thank you for inviting me to do a research stay at Cambridge, and for being always available to offer help and discuss things with me. It is always a pleasure to work with you, and I hope we keep working together much longer.

I would also like to thank Félix García, my University supervisor, for having shared all his knowledge with me, and for being the first person who placed his trust in me in the academic world. I am sure this is not the end of our working relationship.

From the CERN side, I would like to thank Albert de Roeck, my supervisor at CERN, for always supporting me no matter what. I am glad to acknowledge the rest of the CERN neutrino group too. In particular, thanks to Davide Sgalaberna - my ex-colleague (...and future colleague!) - for all our discussions about solving the world problems by combining physics and deep learning, and for inviting me to give a seminar at ETH Zurich; to Georgios Christodoulou, for being the best office mate one could have; and to Stephen Dolan, one of the most outstanding young scientists, for always being able to share all his knowledge with me. Thanks also to Francesca Stocker, Manuel Rodríguez, and Stefania Bordoni; you have also contributed to making this thesis successful (as well as making my daily life a bit happier). Last but not least, I would like to thank Marta Babicz, a bright scientist as well as a phenomenal athlete and musician, not only for being an extraordinary colleague but also for having accompanied me during the last months of writing. Thank you for all the good moments.

I must also acknowledge all the members of the DUNE and T2K collaborations, for allowing me to contribute to such exceptional experiments, and for providing me with their support every time I needed it. Among all these great scientists, I would like to thank Chang Kee Jung, probably one of the most fascinating persons I have ever met, for all our endless discussions, and for inviting me to give a seminar at Stony Brook University and hosting me in my trip to New York. I would also like to acknowledge Viktor Pec, for inviting me to give a seminar at the University of Sheffield, and for his

hospitality when I arrived in the UK. Special thanks to Lorena Escudero, a fantastic physicist and writer, for making me feel at home when I arrived at Cambridge, and for all her valuable comments about my research. I appreciate all the effort you have put into getting this work ready on time. I have met loads of great people during the development of this thesis. However, I feel with the need to mention Sebastian Pina, a coworker that became a real friend, and with whom I still have many more adventures to share.

I should not forget to acknowledge all the people from the ARCOS group at UC3M. Mainly, I want to thank Alejandro Calderón for all his help. I cannot count the number of calls we have had at ungodly hours. If someone has to have his name written here with big letters, this is Javier Prieto, one of my best friends, and the person that is continuously taking care of how I am doing. Of course, I am happy to be able to acknowledge the small ex-ARCOS community that we formed around Saint-Genis-Pouilly, better known as the *Team Kernel*: Guillermo Izquierdo, my *brother* and one of the persons I appreciate the most, thanks for being such a good friend (and excellent hairdresser); Irene García, amazing journalist and queen of surprises, I hope you never change; Nerea Luna, my *Open Days* mate, I am sure there will be many more football games to enjoy; José Cabrero, one of the smartest guys that I know, thanks for being always ready to help; and the captain, Javier López (*Kernel*), thank you for all your advice and for being always available to help or chat about anything.

Finally, I cannot conclude without referring to my lifelong friends. Érik Monjas, my friend since we were four years old: thank you for always being there. Sergio Penela and David González: I hope we can organise more trips soon (pandemic permitting). The last words are for Fran Rodríguez, a real friend that is (even) further away, but that will remain forever.

Published and submitted content

The following publications from the author have been included as part of this thesis:

- B. Abi *et al.* (DUNE Collaboration) [**main author**]. «Neutrino interaction classification with a convolutional neural network in the DUNE far detector.» Published in: *Physical Review D (PRD)* (2020). DOI:10.1103/PhysRevD.102.092003. Ref. [Abi et al., 2020b].
 - This publication is fully included in this thesis in Chapter 3 and Appendix B.
- B. Abi *et al.* (DUNE Collaboration) [**significant contributor**]. «Long-baseline neutrino oscillation physics potential of the DUNE experiment.» Published in: *European Physical Journal C (EPJ C)* (2020). DOI:10.1140/EPJC/S10052-020-08456-Z. Ref. [Abi et al., 2020d].
 - This publication is partially included in this thesis in Chapter 3.
- B. Abi *et al.* (DUNE Collaboration) [**significant contributor**]. «Far Detector Technical Design Report, Volume II: DUNE Physics.» Posted in: *arXiv* (2020). arXiv:2002.03005. Ref. [Abi et al., 2020a].
 - This publication is partially included in this thesis in Chapter 3.
- S. Alonso-Monsalve, L. H. Whitehead. «Image-based model parameter optimization using Model-Assisted Generative Adversarial Networks.» Published in: *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)* (2020). DOI: 10.1109/TNNLS.2020.2969327. Ref. [Alonso-Monsalve and Whitehead, 2020].
 - This publication is fully included in this thesis in Chapter 5.
- M. J. Rodriguez for the the DUNE Collaboration¹. «Fast inference using FPGAs for DUNE data reconstruction.» Published in: *Proceedings of the 24th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2019)*. DOI: 10.1051/epjconf/202024501030. Ref. [Rodriguez, 2020].

¹The author of this thesis is the second author of the publication. However, do to the authorship policy of the DUNE Collaboration, only the first author must sign conference proceedings.

- This publication is fully included in this thesis in Chapters 2 and 6.
- S. Alonso-Monsalve, F. García-Carballeira, A. Calderón. «A heterogeneous mobile cloud computing model for hybrid clouds.» Published in: *Future Generation Computer Systems (FGCS) (2018)*. DOI: 10.1016/J.FUTURE.2018.04.005. Ref. [Alonso-Monsalve et al., 2018].
 - This publication is partially included in this thesis in Chapter 6.
- S. Alonso-Monsalve *et al.*. «Graph neural network for 3D node classification in scintillator-based neutrino detectors.» Accepted for publication in: *Third Workshop on Machine Learning and the Physical Sciences at the 34th Conference on Neural Information Processing Systems (NeurIPS 2020)*. Ref. [Alonso-Monsalve et al., 2020b].
 - This publication is partially included in this thesis in Chapter 4.
- S. Alonso-Monsalve *et al.*. «Graph neural network for 3D classification of ambiguities and optical crosstalk in scintillator-based neutrino detectors.» Submitted for publication in: *Physical Review D (PRD) (2020)*. arXiv:2009.00688. Ref. [Alonso-Monsalve et al., 2020a].
 - This publication is fully included in this thesis in Chapters 2, 4, and Appendices A and B.
- S. Alonso-Monsalve, F. García-Carballeira, A. Calderón, J. Rivadeneira. «Analyzing the distributed training of deep-learning models via data locality.» Submitted for publication in: *29th International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2021)*.
 - This publication is fully included in this thesis in Chapters 1, 2, 6, and Appendix A.

The materials from the sources above included in this thesis are not singled out with typographic means and references.

Abstract

Deep-learning methods are playing a crucial role in numerous scientific and industrial applications. Over the past two decades, these techniques have helped in the collection, reconstruction, and analysis of large data samples in particle physics experiments. The main topic of this PhD research is the study of deep-learning techniques in long-baseline neutrino oscillation experiments. Neutrinos are mysterious light elementary particles, and their investigation is essential to shed light on some of the remaining open questions in physics. The work presented here describes an algorithm based on a convolutional neural network developed to provide highly accurate and efficient selections of electron neutrino and muon neutrino interactions in the Deep Underground Neutrino Experiment (DUNE). With this algorithm, the electron neutrino (antineutrino) selection efficiency peaks at 90% (94%) and exceeds 85% (90%) for reconstructed neutrino energies between 2-5 GeV. The selection efficiency for muon neutrino (antineutrino) interactions is found to have a maximum of 96% (97%) and exceeds 90% (95%) efficiency for reconstructed neutrino energies above 2 GeV. When considering all electron neutrino and antineutrino interactions as signal (both those appearing from oscillations and those intrinsic to the beam), a selection purity of 90% is achieved. These event selections are critical to maximise the sensitivity of the experiment to CP -violating effects, key to further understand the matter-antimatter asymmetry of the Universe.

In high-energy physics experiments, deep learning has also been explored for producing fast simulations and physically-motivated manipulations of simulated images. Some of those simulations, such as the light production and detection, are very computationally expensive and require novel methods to produce the necessary samples while controlling the varied underlying physics model parameters. To do so, we invented the model-assisted generative adversarial network (MAGAN), first validated on simple generic case studies and then successfully applied to the DUNE photon-detector simulation.

Moreover, we also developed graph neural networks for 3D-voxel classification of ambiguities and optical crosstalk for a different particle physics experiment, most precisely for the proposed SuperFGD. This novel 3D-granular plastic-scintillator neutrino detector will be used to upgrade the near detector of the T2K neutrino oscillation experiment,

and our method reports efficiencies and purities of 94-96% per event in the classification of particle track voxels.

Due to the growth and complexity of deep neural networks, researchers have been investigating techniques to train those networks in a more computationally-efficient way. Many efforts have been made by the community to optimise deep-learning models by parallelising or distributing their training computation across multiple devices. In this thesis, we study an approach based on data locality for those neural networks that cannot benefit from scaling their computation due to a significant bottleneck in the data I/O. The research also includes a detailed study on the performance of deep neural networks on hardware accelerator boards.

Resumen

Los métodos de aprendizaje profundo son cada vez más utilizados en numerosas aplicaciones científicas e industriales hoy en día. Durante las dos últimas décadas, estas técnicas se han empleado en la recolección, reconstrucción y análisis de la gran cantidad de datos generados por experimentos de física de partículas. El tema principal de esta tesis doctoral es el uso de estos modelos de aprendizaje profundo en experimentos de física de neutrinos, en concreto en los experimentos de larga distancia DUNE y T2K. Los neutrinos, partículas fundamentales neutras, de las más ligeras del Universo, pueden ser clave para explicar algunas de las cuestiones todavía sin resolver en física fundamental. Entre las diferentes contribuciones que esta tesis ha hecho a su estudio, cabe destacar el desarrollo de un algoritmo basado en una red de neuronas convolucional para seleccionar con gran eficiencia y precisión las interacciones de neutrinos electrónicos y muónicos en el Deep Underground Neutrino Experiment (DUNE). La eficiencia de selección obtenida para neutrinos (antineutrinos) electrónicos alcanza un máximo del 90% (94%) y supera el 85% (90%) para neutrinos con energías reconstruidas en el rango 2-5 GeV. La selección de neutrinos (antineutrinos) muónicos tiene una eficiencia máxima del 96% (97%) y excede el 90% (95%) para neutrinos con energías reconstruidas de más de 2 GeV. Considerando como señal todas las interacciones de neutrinos y antineutrinos electrónicos (procedentes tanto de oscilaciones como intrínsecos en el haz inicial), se logra una pureza en la selección del 90%. Dichas selecciones de eventos son fundamentales para maximizar la sensibilidad del experimento a los efectos de violación de CP, necesarios para entender la asimetría entre materia y antimateria en el universo.

Por otro lado, también se ha explorado el uso de métodos de aprendizaje profundo en otras tareas en experimentos de altas energías, como por ejemplo el desarrollo de redes generativas adversarias para producir simulaciones rápidas y manipulaciones de las imágenes simuladas. Tales simulaciones, como en el caso de la producción y detección de luz procedente de las interacciones de neutrinos, son computacionalmente muy costosas y requieren métodos nuevos para producir los datos necesarios controlando la variación de los parámetros del modelo físico que los describe. Para ello, hemos propuesto un nuevo tipo de red generativa adversaria asistida por modelos (MAGAN, del inglés model-assisted generative adversarial networks), validando su funcionamiento en casos simples

y genéricos primero, para después aplicarla con éxito a la simulación de detección de fotones en DUNE.

Además, en esta tesis se ha estudiado también el uso de redes neuronales aplicadas a grafos para la clasificación tridimensional de vóxeles evitando ambigüedades y diafonía óptica en el SuperFGD, un nuevo detector de neutrinos de centelleo en plástico, granular en tres dimensiones, que se utilizará en la futura mejora del detector cercano del experimento de oscilación de neutrinos T2K, donde nuestro método alcanza eficiencias y purezas del 94-96% por evento en la clasificación de vóxeles producidos por trazas de partículas.

Debido a la expansión y complejidad de las redes neuronales profundas, se están investigando diferentes técnicas para entrenar estas redes de manera más eficiente. La comunidad ha llevado a cabo intentos para optimizar los modelos de aprendizaje profundo mediante paralelización y distribución del cómputo de entrenamiento en múltiples dispositivos. En esta tesis hemos estudiado esta optimización mediante un enfoque basado en la localidad de datos para aquellas redes de neuronas que no pueden beneficiarse de escalar su cómputo debido a un embotellamiento en la E/S de los datos. Nuestra investigación también incluye un estudio detallado sobre el rendimiento de redes neuronales profundas en placas de aceleración de hardware.

Table of contents

List of figures	xix
List of tables	xxiii
Acronyms and abbreviations	xxv
1 Introduction	1
1.1 Motivation	2
1.2 Objectives	4
1.3 Structure of the document	5
2 State of the art	7
2.1 Deep learning in high-energy physics	7
2.1.1 Big data and deep learning at the Large Hadron Collider	7
2.1.2 Deep learning at neutrino experiments	11
2.2 High-performance computing for deep learning	17
2.2.1 Deep learning on hardware accelerators	19
3 Event identification using convolutional neural networks	21
3.1 The DUNE Convolutional Visual Network	21
3.1.1 Inputs to the CVN	22
3.1.2 Network architecture	24
3.1.3 Outputs from the CVN	25
3.1.4 Training details	26
3.1.5 Feature maps	28
3.2 Neutrino flavour identification performance	30
3.3 Exclusive final state results	34
3.4 Robustness	36
3.4.1 Charged lepton studies	36
3.4.2 Occlusion tests	39

3.5	Summary	43
4	Hit-level identification using graph neural networks	45
4.1	Motivation	45
4.2	Methodology	47
4.2.1	Data sample generation	47
4.2.2	Network architecture	50
4.2.3	Training	51
4.3	Results	52
4.4	Systematic uncertainty considerations	58
4.5	Summary	61
5	Fast simulation and model parameter optimisation using generative adversarial networks	63
5.1	Model-assisted generative adversarial networks	63
5.1.1	Algorithm description	64
5.1.2	Training details	67
5.2	Case studies	69
5.2.1	Case study 1: first order polynomial images	69
5.2.2	Case study 2: circular signal with noise and amplitude variation .	71
5.2.3	Fast simulation with the emulator	73
5.3	Application to the DUNE photon-detector simulation	74
5.3.1	Problem description	75
5.3.2	Evaluation	76
5.4	Summary	76
6	Performance study of deep-learning workloads	79
6.1	Analysing the distributed training of deep-learning models via data locality	79
6.1.1	I/O overload study	79
6.1.2	DaloFlow: a deployment environment for distributed deep-learning computation	81
6.1.3	Evaluation	82
6.2	Fast inference using FPGAs for DUNE data reconstruction	85
6.2.1	The Micron Deep Learning Accelerator technology	85
6.2.2	Benchmark using the DUNE CVN	86
6.3	Summary	89

7 Conclusion and future work	91
7.1 Contributions	92
7.2 Dissemination	93
7.2.1 Publications	93
7.2.2 Presentations	94
7.3 Future work	96
References	97
Appendix A Introduction to deep learning	115
A.1 Notation and definitions	115
A.2 Machine-learning concepts	118
A.2.1 Supervised learning	118
A.2.2 Unsupervised learning	119
A.2.3 Reinforcement learning	119
A.3 Neural networks	119
A.3.1 Forward and backward propagation	121
A.3.2 Considerations	123
A.3.3 Regularisation	126
A.3.4 Convolutional neural networks	128
A.3.5 Graph neural networks	130
A.3.6 Generative adversarial networks	132
A.4 Parallel and distributed training	133
A.4.1 Model parallelism	133
A.4.2 Data parallelism	134
A.4.3 Horovod	135
A.4.4 Deployment of deep-learning models on HPC environments	135
Appendix B Neutrino physics	137
B.1 Neutrino oscillations	138
B.2 Neutrino oscillation experiments	140
B.3 The DUNE experiment	141
B.3.1 DUNE far detector	141
B.3.2 CP -violation measurement	142
B.4 The T2K experiment	144
B.4.1 SuperFGD	144

Appendix C Complementary information to Chapter 4	147
C.1 Input variables	147
C.2 Event gallery	150
Appendix D Complementary information to Chapter 6	157
D.1 DaloFlow user guide	157
D.1.1 Several containers on a single node	157
D.1.2 Several containers on several nodes	158
D.2 Horovod deployment comparison	159
Index	161

List of figures

2.1	Background rejection vs signal efficiency results for different configurations of shallow networks and deep networks trained on high-level and low-level features.	8
2.2	Example of <i>jet</i> images from hadronically decaying W bosons used as input for <i>jet</i> tagging.	9
2.3	Average photon shower images, produced by the Geant4 simulation toolkit and the generator module of a GAN.	10
2.4	Simulated CC ν_μ event from NOvA.	12
2.5	Semantic segmentation in MicroBooNE. Example of a neutrino event display from a CC π^0 candidate detector data.	13
2.6	Data and computation size comparison between a 3D image and a graph.	15
2.7	Simulated LArTPC event.	16
2.8	Full end-to-end deep-learning reconstruction chain for 3D LArTPC neutrino events.	17
2.9	TensorFlow I/O pattern on ImageNet data using Horovod for distributing the computation across six workers.	18
2.10	Flexibility vs computational efficiency for different hardware processing units.	19
3.1	A 2.2 GeV CC ν_e interaction shown in the three readout views of the DUNE LArTPCs showing the characteristic electromagnetic shower topology.	22
3.2	Three interactions shown in the collection view.	23
3.3	Simplified diagram of the DUNE CVN architecture.	24
3.4	Loss and accuracy results for training and validation, given for the four main CVN outputs.	28
3.5	Visualisations of the feature extraction in the CVN for a 12.2 GeV CC $\bar{\nu}_e$ interaction.	29
3.6	The number of events as a function of the CVN CC ν_e classification score shown for FHC and RHC beam modes.	30

3.7	The number of events as a function of the CVN CC ν_μ classification score shown for FHC and RHC beam modes.	31
3.8	The CC ν_e selection efficiency for FHC-mode and RHC-mode simulation with the criterion $S(\nu_e) > 0.85$	32
3.9	The CC ν_μ selection efficiency for FHC-mode and RHC-mode simulation with the criterion $S(\nu_\mu) > 0.5$	33
3.10	Significance of the DUNE determination of CP -violation as a function of the true value of δ_{CP} , for seven and ten years of exposure, in both normal and inverted ordering.	34
3.11	The CC ν_μ 1 proton and NC $1\pi^0$ combined score distributions from the CVN.	35
3.12	The variation of the signal selection efficiency as a function of the charged lepton energy shown for three ranges of hadronic energy.	37
3.13	The background acceptance in the CC ν_e and ν_μ selections for different background interactions.	37
3.14	The signal selection efficiency for CC ν_μ and CC ν_e interactions as a function of the angle between the outgoing charged lepton and the parent neutrino.	38
3.15	Occlusion test generation for a CC ν_μ event.	39
3.16	Example of an occlusion test for a CC ν_e event.	40
3.17	Example of an occlusion test for a CC ν_e event.	40
3.18	Example of an occlusion test for a CC ν_μ event.	41
3.19	Example of an occlusion test for a CC ν_μ event.	41
3.20	Example of an occlusion test for a CC ν_μ event.	41
3.21	Distribution of largest change in the true classification score of the event after running the occlusion tests.	42
4.1	Visualisation of a neutrino interaction in a finely segmented 3D scintillator detector.	46
4.2	Visual illustration of the GraphSAGE sample and aggregate approach with a depth of three.	51
4.3	Validation F1 results on GENIE and P-Bomb samples.	52
4.4	Example GNN prediction results for the interaction shown in Fig. 4.1. . .	54
4.5	Efficiency and purity as a function of the number of voxels in the event for a sample trained and tested on GENIE simulated data.	55
4.6	Efficiency and purity as a function of the number of tracks in the event for a sample trained and tested on GENIE simulated data.	55

4.8	Efficiency and purity as a function of the distance to the neutrino interaction vertex for a sample trained and tested on GENIE data.	56
4.9	The distribution of the minimum charge among the three voxel charges for the GENIE sample.	56
4.10	The fraction of the true total deposited energy obtained when using the GNN (trained on GENIE) or the charge cut as a classification method. . .	56
4.11	Distribution of the total true deposited energy after rejecting the ghost voxels.	59
4.12	Difference between the total true deposited energy computed after rejecting the ghost voxels.	59
4.13	Difference between the total true deposited energy; fractional standard deviation of the difference of the total true deposited energy.	59
5.1	An overview of the model-assisted GAN.	64
5.2	The architecture of the Siamese network.	66
5.3	The two training phases of the MAGAN.	68
5.4	Emulator output vs simulator output.	70
5.5	Generator-Simulator output from five random noise vectors vs some random true images.	71
5.6	Emulator output vs simulator output.	72
5.7	Generator-Simulator output from some random noise vectors vs some random true images.	73
5.8	Anode plane assembly (APA) example.	74
5.9	Image format of the light pattern seen across the photon detectors. . . .	75
5.10	Arbitrary training example.	76
6.1	Simulated scenarios using SimGrid.	80
6.2	DaloFlow usage example: deployment of 4 containers on a single node or on multiple nodes.	81
6.3	Percentage of the total training time used for I/O in some scenarios from Tab. 6.1.	84
6.4	Confusion matrices for the classification task of the different neutrino interaction.	88
6.5	Histogram for the absolute error between the GPU and the SB852 board. . .	89
A.1	Artificial neuron example.	120
A.2	Example of a multi-layer perceptron.	120
A.3	Simple neural network used for logistic regression.	121

A.4	A comparison of three different ways a model can fit the training data. . .	126
A.5	Training and test errors depending on the complexity of the model. . . .	127
A.6	State-of-the-art CNN performance on ImageNet dataset comparison. . . .	129
A.7	Semantic-segmentation example.	130
A.8	GAN training.	132
A.9	Examples of images generated with a GAN from some random noise. . .	133
A.10	Data parallelism graphical example.	134
B.1	Overview of the Standard Model particles.	137
B.2	DUNE will consist of two neutrino detectors.	141
B.3	Reconstructed energy distribution of ν_e and $\bar{\nu}_e$ CC-like events selected by the CVN assuming 3.5 years running in the neutrino-beam mode and antineutrino-beam mode, for a total of seven years exposure.	143
B.4	The T2K experiment has a intense muon neutrino beam at the J-PARC facility, close to the near detectors.	144
B.5	Geometry of a single SuperFGD element. Each cube is intersected by 3 WLS fibres.	145
B.6	Sketch of the signal generation, fibre transport and signal detection processes highlighting the production of optical crosstalk signals.	145
B.7	Example of a ghost voxel appearance.	145
C.1	Correlation matrices for the input variables of the GENIE and P-Bomb datasets used.	149
C.2	3D visualisation of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views (I).	151
C.3	3D visualisation of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views (II).	152
C.4	3D visualisation of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views (III).	153
C.5	3D visualisation of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views (IV).	154
C.6	3D visualisation of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views (V).	155
C.7	3D visualisation of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views (VI).	156
D.1	Several workers on a single computer.	158
D.2	Several workers on several computers.	158

List of tables

2.1	SSCN vs dense MicroBooNE U-ResNet scalability with the 3D image spatial size on an NVIDIA V100 GPU.	14
3.1	Actual flavour and CVN scores for input events and output maps from occlusion tests shown in Figs. 3.16 to 3.20.	42
4.1	Descriptions of both GENIE and P-Bomb datasets.	48
4.2	Architectural parameters.	51
4.3	Mean efficiencies and purities of voxel classification.	53
4.4	Mean efficiencies and purities of voxel classification for the GNN and a simple charge cut.	57
4.5	Mean efficiencies and purities of voxel classification, per voxel, for different crosstalk values.	60
5.1	The mean and standard deviation of the model parameters used to make the true data images and those from the best match parameters learned by the generator in the first case study.	70
5.2	The mean and standard deviation of the model parameters used to produce the true data images and the corresponding values from the best match parameters from the generator in the second case study.	72
5.3	Image production time comparison of the simulator and the emulator for each case study.	73
5.4	Geometric extent of the model parameters.	75
5.5	Image production time using the emulator.	76
6.1	Comparison between a distributed filesystem and the solution leveraging data locality.	83
6.2	Classification report for each of the 13 different categories.	87
6.3	Classification report for each neutrino flavour.	88

6.4	Inference time comparison between the Micron SB852 and an Intel CPU processor.	89
A.1	Training results for the Fashion-MNIST dataset using different gradient-descent optimisers.	125

Acronyms and abbreviations

ANN	Artificial neural network
API	Application programming interface
ArgoNeuT	Argon Neutrino Test
ASIC	Application-specific integrated circuit
ATLAS	A Toroidal LHC Apparatus
CC	Charged current
CDR	Conceptual Design Report
CKM	Cabibbo–Kobayashi–Maskawa
CMS	Compact Muon Solenoid
CNN	Convolutional neural network
CPU	Central processing unit
CVN	Convolutional Visual Network
DAQ	Data acquisition
DCGAN	Deep convolutional generative adversarial network
DIS	Deep inelastic
DLA	Micron Deep Learning Accelerator
DNN	Deep neural network
DONuT	Direct Observation of Nu Tau
DUNE	Deep Underground Neutrino Experiment

EM	Electromagnetic
eV	Electronvolt
EXO	Enriched Xenon Observatory
FCN	Fully convolutional network
FD	Far detector
FHC	Forward horn current
FLOPS	Floating-point operations per second
FPGA	Field-programmable gate array
GAN	Generative adversarial network
GAP	Global Average Pooling
GlusterFS	Gluster File System
GNN	Graph neural network
GPU	Graphics processing unit
GraphSAGE	Graph SAmple and aggreGatE
HDFS	Hadoop Distributed File System
HEP	High-energy physics
HLS	High-level synthesis
HMC	Hybrid Memory Cube
HPC	High-performance computing
Hyper-K	Hyper-Kamiokande
ID	Identification
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
INGRID	Interactive Neutrino GRID
I/O	Input/output

IoT Internet of things

LArTPC Liquid-argon time projection chamber

LBL Long Baseline

LBNF Long-Baseline Neutrino Facility

LHC Large Hadron Collider

LMDB Lightning Memory-Mapped Database

LSTM Long short-term memory

LTS Long-term support

MAC Multiply-accumulate

MAGAN Model-assisted generative adversarial network

MC Monte Carlo

MicroBooNE Micro Booster Neutrino Experiment

ML Machine learning

MLP Multilayer perceptron

MNIST Mixed National Institute of Standards and Technology

MPI Message Passing Interface

NC Neutral current

ND Near detector

NEXT Neutrino Experiment with a Xenon TPC

NOvA NuMI Off-Axis Electron Neutrino Appearance

ONNX Open Neural Network Exchange

P-Bomb Particle bomb

PD Photon-detection

PMNS Pontecorvo–Maki–Nakagawa–Sakata

QE	Quasi-elastic
QSFP	Quad Small form-factor pluggable
RAM	Random-access memory
ReLU	Rectified linear unit
ResNet	Residual neural network
Res	Resonant
RHC	Reverse horn current
RNN	Recurrent neural network
SDK	Software development kit
SE	Squeeze-and-Excitation
SGD	Stochastic gradient descent
SIMD	Single Instruction, Multiple Data
SiPM	Silicon photomultiplier
SNO	Sudbury Neutrino Observatory
SSCN	Submanifold sparse convolutional network
SS	Semantic segmentation
Super-K	Super-Kamiokande
SuperFGD	Super Fine-Grained Detector
SURF	Sanford Underground Research Facility
T2K	Tokai to Kamioka
TDR	Technical Design Report
TPU	Tensor processing unit
WLS	Wavelength shifting

Chapter 1

Introduction

The existence of the neutrino, ν , was first postulated by Wolfgang Pauli in 1930, who called this new particle the *neutron*. Enrico Fermi renamed it the *neutrino* shortly after [Fermi, 1934]. To be precise, what they hypothesised was the existence of the *electron antineutrino*, $\bar{\nu}_e$, in order to explain the beta-decay processes observed. Neutrinos were not experimentally discovered until 1956 by Reines and Cowan [Cowan et al., 1956; Reines and Cowan, 1953]. In 1962, an experiment at Brookhaven National Laboratory published the discovery of a second kind of neutrino: the *muon neutrino*, ν_μ [Danby et al., 1962]. The third type of neutrinos, the *tau neutrino*, ν_τ , postulated to exist since 1975, was finally detected in 2000 by the DONuT experiment at Fermilab [Kodama et al., 2001]. These three neutrino types (or *flavours*), ν_e , ν_μ , and ν_τ , are associated with the corresponding charged leptons: electron, muon, and tau, respectively.

Neutrinos are prevalent throughout the Universe, being the second most abundant particles, after photons. There are many different neutrino sources, such as nuclear reactors, the Sun (solar neutrinos), cosmic rays (atmospheric neutrinos), or even human-made accelerators. Neutrinos have no electric charge, a very tiny mass, and can interact only via the weak force. Neutrinos oscillate, meaning that they can change their flavour: a neutrino generated with a specific flavour can later be measured to have a different flavour. Neutrino oscillations were first proposed in 1957 [Pontecorvo, 1957], but the phenomenon remained unproven until four decades later, where the Super-K and SNO experiments showed that some of the electron neutrinos changed their flavour into the other two neutrino flavours [Ahmad et al., 2002; Fukuda et al., 1998b].

The three neutrinos, ν_e , ν_μ , and ν_τ (and their corresponding antiparticles $\bar{\nu}_e$, $\bar{\nu}_\mu$, and $\bar{\nu}_\tau$) are elementary particles belonging to the Standard Model of particle physics. The Standard Model is a theory in physics that classifies all the elementary particles and describes the behaviour of three of the four known fundamental forces (electromagnetic, weak, and strong interactions, excluding gravity). In fact, it is one of the most successful

theories in physics, and it can be used to explain most of the experimental observations, including interactions at neutrino experiments. However, despite its success, it cannot explain the phenomenon of neutrino oscillations, as neutrinos are predicted to be massless in this model.

Therefore, neutrinos can be the key to discover physics beyond the Standard Model. One of the most fundamental observations of the Universe is that it is matter-dominated, and our current measurements do not explain why there is not an equal amount of antimatter. It implies that there is an intrinsic difference in how matter and antimatter particles interact, and this difference is known as *CP-violation*. It is possible that neutrinos and antineutrinos can oscillate differently, and a discovery of *CP-violation* in neutrino oscillations could be the catalyst to understanding the matter-antimatter asymmetry of the Universe.

The nature of the neutrino remains mysterious, hence the study of neutrino oscillations is crucial to learn more about neutrinos and particle physics in general. Appendix B gives a broader overview of neutrino theory and experimentation.

1.1 Motivation

Since 1999, a series of experiments have measured a number of the parameters that describe neutrino oscillations [Abe et al., 2020; Acero et al., 2019; Adamson et al., 2014; Aharmim et al., 2005; Ahn et al., 2006; An et al., 2012; Araki et al., 2005; Fukuda et al., 1998a]. Some of these experiments are long-baseline neutrino oscillation experiments that use two detectors to characterise a beam of (anti)neutrinos: a near detector, located a few hundred meters away from the target, that determines the original beam composition, and a far detector, located several hundred kilometres away, that measures neutrinos flavour oscillations.

A new generation of neutrino detectors are under development, aiming to resolve the remaining open questions in neutrino physics. To achieve this, two main detector technologies stand out. The first one is based on liquid-argon time projection chambers (LArTPCs) [Rubbia, 1977]. An example of an experiment based on that technology is DUNE [Abi et al., 2020a], a next-generation neutrino oscillation experiment that aims to measure *CP-violation* in the neutrino sector as part of a wider physics programme. In order to maximise the *CP-violation* sensitivity, it is key to be able to efficiently identify the signal (i.e., CC ν_μ , CC $\bar{\nu}_\mu$, CC ν_e and CC $\bar{\nu}_e$) interactions and have a powerful

rejection of background events, hence a precise algorithm is needed to achieve very high signal efficiency¹ and background rejection for event classification.

The other next-generation neutrino oscillation experiment proposed is Hyper-K [Abe et al., 2018], which will use an upgraded version of both near and far detectors of T2K [Abe et al., 2019]. The SuperFGD detector will be one of the systems in the upgraded near detector complex for T2K, and the response to a charged particle is read out into three orthogonal 2D projections [Blondel et al., 2018]. The 2D views of the events are combined into 3D as a series of hits, which is a process that can lead to some ambiguities. To fully understand the neutrino interactions, it is important to clean the events of these ambiguities, especially to be able to reconstruct short particle tracks. In order to achieve this, it is necessary to develop algorithms to categorise the different 3D hits, so that most of the ambiguities can be identified and rejected.

Deep-learning techniques have emerged as a powerful solution to problems in multiple domains: from object detection [Liu et al., 2020; Minaee et al., 2020] to text recognition [Yousef et al., 2020]; and they are applied to numerous fields, such as autonomous driving [Kuutti et al., 2020], natural language processing [Otter et al., 2020], health-care [Esteva et al., 2019; Haskins et al., 2020], or computer games [Justesen et al., 2020]. These techniques mainly consist of deep artificial neural networks (DNNs), which are mathematical algorithms that learn to approach problems by tuning their parameters throughout an iterative gradient-descent-based optimisation process (see Appendix A for more information). Once a neural network is correctly trained, it may be used to give accurate predictions to real problems, outperforming standard computing methods and human experts in many fields [Buetti-Dinh et al., 2019; Sturman et al., 2020; Voulodimos et al., 2018]. High-energy physics (HEP) experiments, and neutrino experiments, in particular, are not an exception, and deep-learning algorithms could prove critical in addressing most of the challenges related to this type of detectors [Radovic et al., 2018].

In HEP experiments, the detector design and optimisation are always guided by accurate and computationally-expensive simulations of the detector behaviour [Agostinelli et al., 2003; Sjöstrand, 1994]. This is also the case of neutrino experiments [Andreopoulos et al., 2010], where innumerable analyses are performed using simulated data. DNNs are typically trained on simulated data and used for making accurate predictions on eventual experimental data. If the experiment has sufficient computing resources, simulated data might effectively be unlimited, minimising the risk of DNNs to overtraining. However, the difference in performance between data and simulation could be an issue, and ensuring the robustness of deep-learning models against systematic uncertainties becomes

¹The efficiency of the signal is calculated by dividing the number of correctly classified events (i.e., events with a predicted label that matches the actual label) by all true events with that label.

a fundamental requirement [Louppe et al., 2016]. Besides, training such DNNs (neural networks with tens or hundreds of layers) is a task that generally requires a considerable amount of time [Toronto, 2018], which leads to scaling the workloads to be able to handle computationally-efficient² trainings.

1.2 Objectives

Event classification, reconstruction, and simulation are all essential steps to achieve the goals of the next-generation neutrino experiments (e.g., CP -violation measurements). The aim of this PhD thesis is to address these matters using different deep-learning techniques and high-performance computing methods, and it can be divided into the following sub-objectives:

- O1 **Proposing algorithms for maximising the accuracy of neutrino event classification.** Specifically, the purpose is to use a convolutional neural network for high signal efficiency and background rejection in the DUNE experiment.
- O2 **Boosting the reconstruction of three-dimensional events by developing rigorous mechanisms for cleaning charged particle tracks.** The idea is to use graph neural networks for assisting a 3D particle-set reconstruction of neutrino interactions in the SuperFGD detector of the T2K experiment.
- O3 **Providing fast alternatives to standard simulations and trying to minimise the gap between simulated and experimental data.** The approach is to propose a variation of a standard generative adversarial network to produce fast simulations as well as physically-motivated manipulations of simulated images through variation of underlying model parameters.
- O4 **Analysing the computational performance of the techniques studied to end up with more efficient algorithms.** The plan is to analyse the bottlenecks and solutions of distributed deep-learning trainings and to study the performance of neural networks on hardware accelerators.

²Computational efficiency refers to the amount of time or memory required by an application or piece of software (i.e., a deep-learning model in the context of this thesis).

1.3 Structure of the document

The rest of the document is divided into the following chapters:

- Chapter 2, *State of the art*, reviews the state-of-the-art deep-learning methods in high-energy physics and the current status of high-performance computing for deep learning.
- Chapter 3, *Event identification using convolutional neural networks*, presents the convolutional neural network developed to provide accurate and efficient selections of neutrino interactions in DUNE.
- Chapter 4, *Hit-level identification using graph neural networks*, presents the approach based around a graph neural network used for 3D-voxel classification on the SuperFGD detector.
- Chapter 5, *Fast simulation and model parameter optimisation using generative adversarial networks*, proposes and demonstrates the use of model-assisted generative adversarial networks for fast simulations, and to re-tune the default simulation to minimise any bias.
- Chapter 6, *Performance study of deep-learning workloads*, analyses an approach based on data locality for distributed trainings of deep-learning models, and studies the performance of deep neural networks on hardware accelerators.
- Chapter 7, *Conclusion and future work*, summarises the main contents of this PhD thesis, including a list of contributions and future work.

The chapters above are complemented with two appendices:

- Appendix A, *Introduction to deep learning*, gives a technical overview of all related deep-learning topics discussed in this document.
- Appendix B, *Neutrino physics*, includes basic concepts about neutrinos that are discussed throughout the chapters above, as well as giving a brief description of two international neutrino experiments: DUNE and T2K.
- Appendix C, *Complementary information to Chapter 4*, includes some event displays and information about the variables used for the work presented in Chapter 4.
- Appendix D, *Complementary information to Chapter 6*, gives instructions for using the deployment environment presented in Chapter 6.

Chapter 2

State of the art

This chapter presents a review of the state-of-the-art deep-learning methods used in the field of high-energy physics, as well the latest stage about the convergence of deep learning and high-performance computing.

2.1 Deep learning in high-energy physics

Machine-learning algorithms, and artificial neural networks, in particular, are techniques widely used in the high-energy physics (HEP) community [Teodorescu, 2008]. Over the past two decades, these techniques have been applied to help in the collection, reconstruction, and analysis of large data samples of particle physics experiments [Baldi et al., 2014; Racah et al., 2016; Sadowski et al., 2017].

2.1.1 Big data and deep learning at the Large Hadron Collider

Several machine-learning methods have been developed for the reconstruction and analysis of data from the Large Hadron Collider (LHC) detectors [Guest et al., 2018; Józefowicz et al., 2016; Wielgosz et al., 2017]. The LHC experiments produce data at rate of about one petabyte per second; this rate is dramatically reduced by performing some hardware filtering and compression techniques into a rate of 50 terabytes per second [Evans and Bryant, 2008], which is still higher than the amount of data Facebook collects in a year [Pamela Vagata, 2014]. Machine-learning techniques are needed here to efficiently select which data to keep in real time, and to accurately process these data offline for further analysis [Adam-Bourdarios et al., 2015].

Event selection

Event selection consists of separating signal events (events producing particles of interest) from background events (events producing other particles), and it is a well-studied application of machine learning in HEP. The first approaches in the field were focused on using shallow neural networks along with feature engineering [Whiteson and Whiteson, 2009], where modern methods use deeper and more complex models, such as deep convolutional neural networks [Bhimji et al., 2018].

Recent studies suggest that feature engineering, where scientists apply their knowledge to modify the datasets by using physics-inspired high-level features to characterise each event (instead of the original low-level features), is often sub-optimal [Guest et al., 2018]. Typically, these high-level features are a strict function of the original low-level features; thus, they do not provide new information to the deep-learning algorithm. As an example, the work in Ref. [Baldi et al., 2014] shows that shallow networks that are trained on high-level feature datasets (with extra physics-inspired features) are significantly outperformed by deep networks that only use the original low-level data (see Fig. 2.1). We use a similar idea for the work presented in Chapter 4, where our feature engineering is minimal, and our datasets mainly use low-level features to characterise the input events.

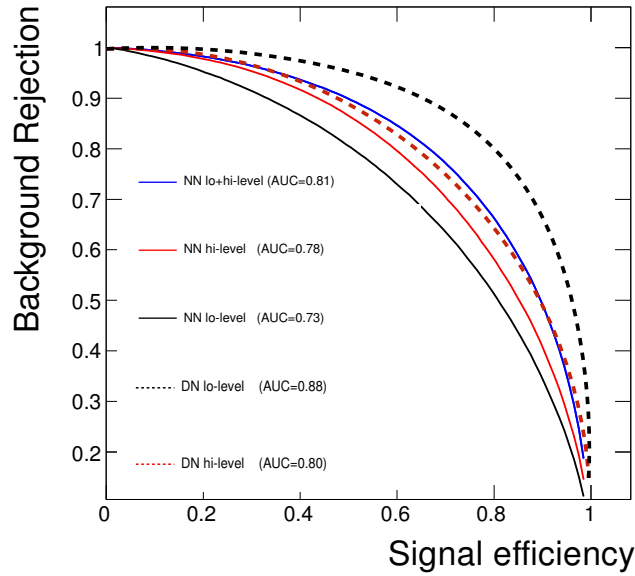


Fig. 2.1 Background rejection vs signal efficiency results for different configurations of shallow networks (NN) and deep networks (DN) trained on high-level (hi-level) and low-level (lo-level) features. The deep network trained on events with only low-level features outperforms all other configurations. Figure taken from Ref. [Guest et al., 2018], based on the results from [Baldi et al., 2014].

Computer vision

Computer-vision techniques have a direct application in HEP since the outputs of particle-physics detectors can usually be displayed as 2-D or 3-D images. The CMS and ATLAS experiments at CERN have already explored the usage of convolutional neural networks (CNNs) for classification of *jets*¹ and the analysis of signals in particle physics detectors [Andrews et al., 2020; Baldi et al., 2016; Barnard et al., 2017; Cogan et al., 2015; de Oliveira et al., 2016; Komiske et al., 2017; The ATLAS Collaboration, 2017]. Figure 2.2 illustrates an example of *jet* images that can be made from either ATLAS or CMS.

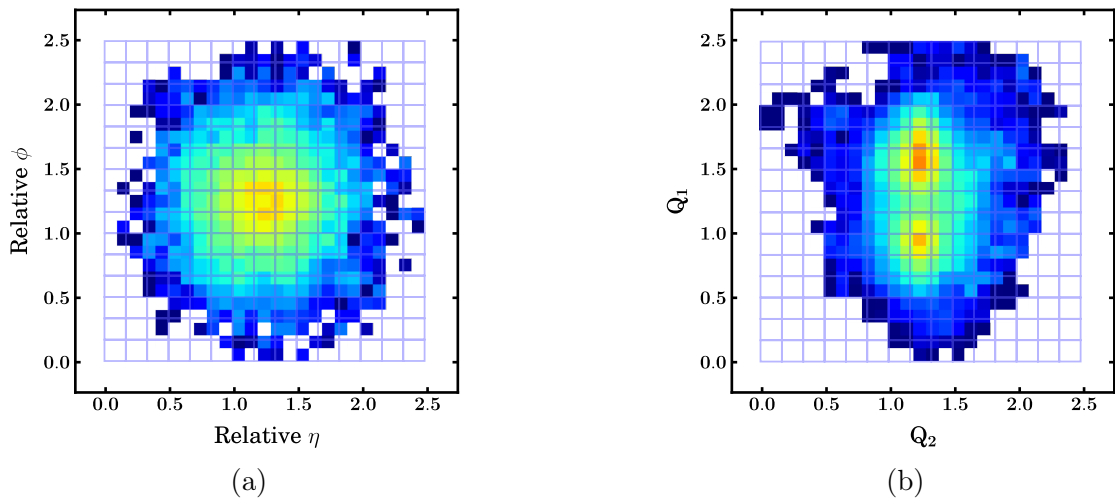


Fig. 2.2 Example of *jet* images from hadronically decaying W bosons used as input for *jet* tagging. (a) Average image from a large sample of $W^\pm$ *jet* images, before rotation; (b) average image from the same sample, after pre-processing. Figure taken from Ref. [Cogan et al., 2015].

Even though computer vision has been successfully used in simplified scenarios such as *jet* classification, CNNs are translation invariant, making them challenging for LHC data where the collision region is critical. Conversely, CNNs have a more direct application in neutrino physics, as shown in Sec. 2.1.2 and the work presented in Chapter 3.

Fast simulation

Simulations are critical in the HEP world. They are able to describe and model the behaviour of detectors accurately and are fundamental to understanding phenomena in particle physics, with the disadvantage of being computationally intensive [Beringer et al., 2012]. The computation of heavy simulations is a task that consumes significant

¹*Jet*: collimated streams of high-energy particles flying in roughly the same direction.

computing resources. As a consequence, fast simulations play a relevant role by reducing the computing resources needed and providing physicists with valuable results in a shorter period.

Traditionally, fast simulation techniques were performed by parameterising distributions by hand [Guest et al., 2018; Lukas, 2012], while modern solutions use generative adversarial networks (GANs) [Goodfellow et al., 2014] to learn the outputs of the simulation from available simulated samples. The latter typically occasions a small reduction in the quality of the simulation, but the speed makes up for it. During training, the generator module of a GAN learns the underlying simulated distribution and, once it is trained, it can be used to produce fast emulated images by mapping some random input noise to images that look like the simulated ones. Although GANs have not been broadly used in real-world scientific experiments, some promising work has been done on the production of *jet* images [de Oliveira et al., 2017], GAN-based calorimeter simulations [Paganini et al., 2018a,b; Vallecorsa, Sofia et al., 2019], and in the production of galaxy images [Schawinski et al., 2017]. Figure 2.3 shows an example of average simulated calorimeter images against the corresponding average images produced by the generator module of a GAN.

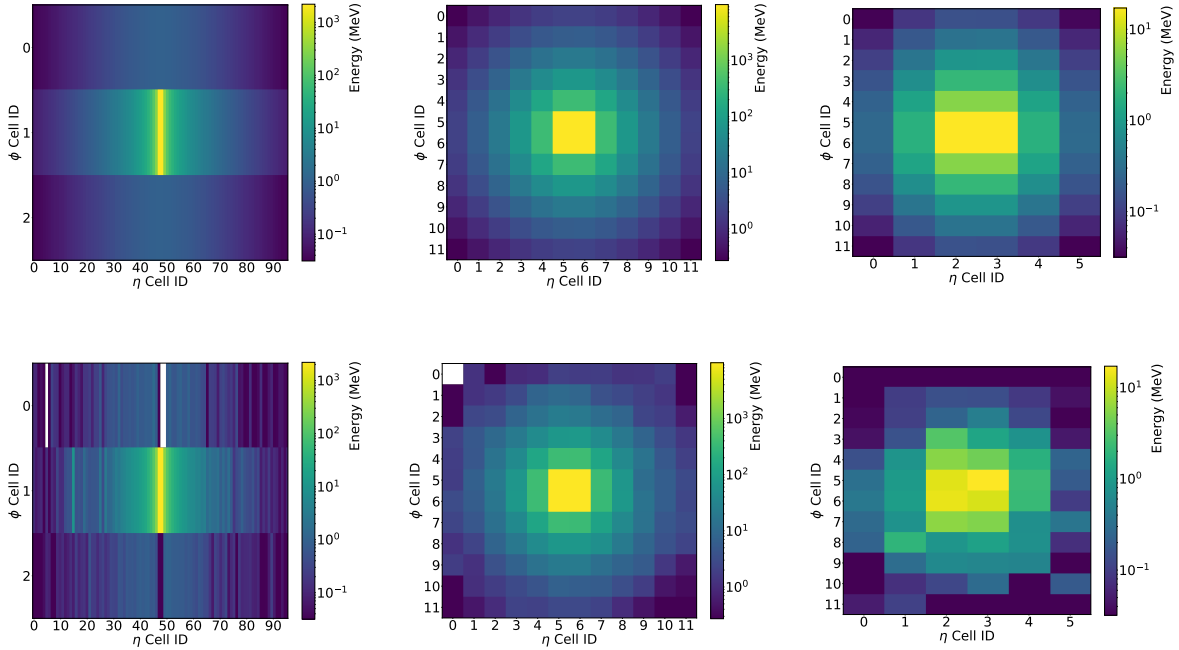


Fig. 2.3 Average photon shower images, produced by the Geant4 simulation toolkit (top) and the generator module of a GAN (bottom). Figure taken from Ref. [Paganini et al., 2018b].

To the best of our knowledge, there is no GAN variant in the literature that aims to generate a vector of physics parameters that are used to produce fake images through a defined mapping of the physics parameters to an image as opposed to generating the fake images directly (as presented in Chapter 5). However, there are some related studies to consider. One example is the conditional GAN [Mirza and Osindero, 2014] that was used to generate MNIST digits conditioned on class labels. More recent studies used conditional GANs for more complex tasks, such as generating aged versions of people’s faces that preserve their identities [Antipov et al., 2017]. There is also some novel work that proposes a conditional GAN framework which is robust against forgetting² in generative models [Wu et al., 2018].

During the last few years, some studies successfully learnt knowledge constraints from image and text generation [Hu et al., 2018] that were used to improve the results over base generative models or to learn disentangled representations in a completely unsupervised manner [Chen et al., 2016]. In addition, the authors of Ref. [Creswell and Bharath, 2018] introduced a new inversion technique to identify attributes of a dataset that a trained GAN is able to model, and quantitatively compare the performance of different generative networks.

Several domains could benefit from the approach we present in Chapter 5, but its best application is probably in physical experiments [Radovic et al., 2018]. Contrary to the above studies, the Model-Assisted GAN that we present in Chapter 5 could be, for instance, used to learn the optimal parameters needed by a Monte Carlo simulation for mimicking detector images in physics experiments.

2.1.2 Deep learning at neutrino experiments

Deep-learning techniques are now commonly applied within the field of neutrino physics. In particular, convolutional neural network (CNN) [LeCun et al., 1989] algorithms that operate on two-dimensional images of the neutrino interactions have been very successful in a number of tasks, such as event classification [Abi et al., 2020a,b,d; Acciarri et al., 2017a; Aurisano et al., 2016] and pixel-level identification of track-like (linear) and shower-like (locally dense) energy deposits [Abi et al., 2017; Adams et al., 2019]. CNNs have found particular success in neutrino experiments where interactions can occur at any location in large uniform detector volumes [Adams et al., 2020b; Radovic et al., 2018], and the characteristic translational invariance of CNN methods represents an advantage rather than a challenge.

²Degradation suffered by neural networks trained sequentially without revisiting data from previous tasks.

Application of convolutional neural networks in neutrino physics

The NOvA experiment [Patterson, 2013] pioneered the use of CNNs for neutrino event classification [Aurisano et al., 2016]. They use a customised version of the GoogLeNet architecture (also known as Inception-v1) [Szegedy et al., 2014] to identify neutrino interactions based on their topology in the detector (see Fig. 2.4). The results showed that the proposed CNN clearly outperforms other traditional algorithms used by the NOvA collaboration, especially in terms of signal efficiency with gains of 35% for CC ν_e interactions. Apart from the neutrino identification CNN, NOvA is also using another CNN for classifying reconstructed particles as either leptons or hadrons [Yu, 2019], and a regression CNN to reconstruct ν_e and electron energies [Baldi et al., 2019], with improvements up to 16% in terms of root-mean-square error. DUNE is currently using a slightly modified version of the NOvA regression CNN for reconstructing energies of CC ν_e events in the DUNE far detectors [Seong et al., 2019].

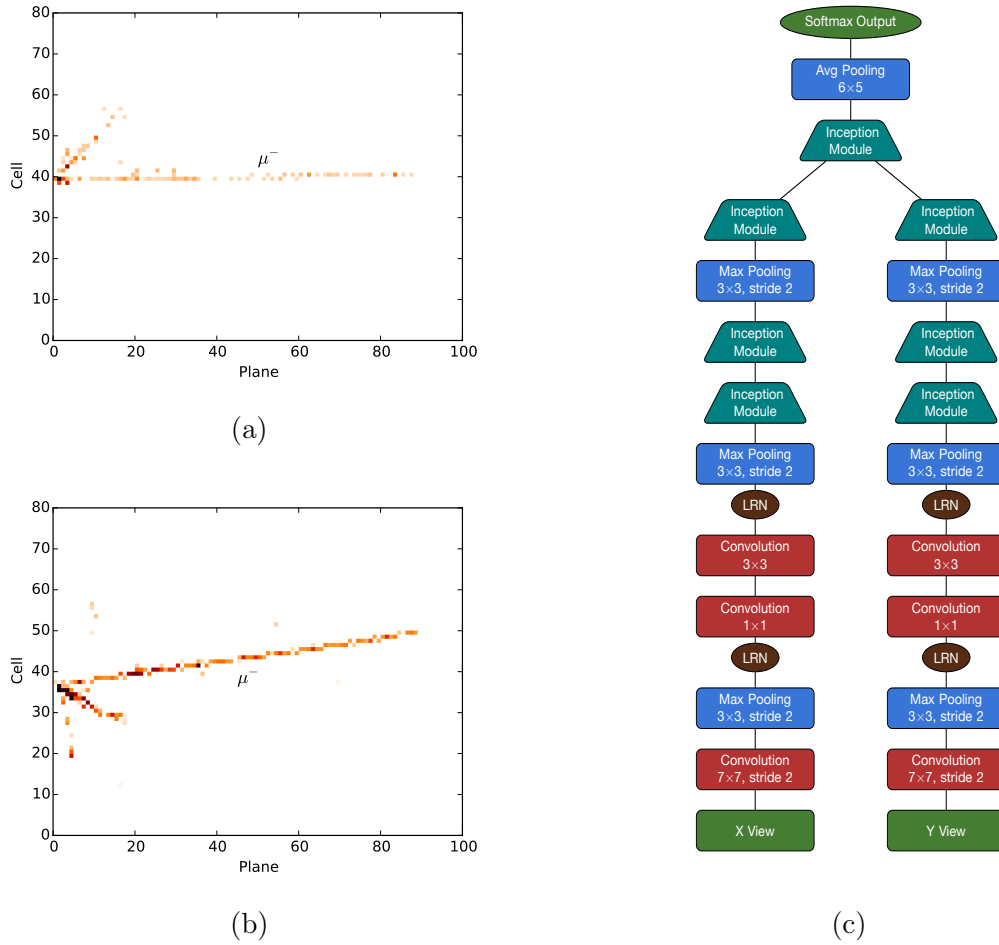


Fig. 2.4 Simulated CC ν_μ event from NOvA: (a) X-view; (b) Y-view; (c) architecture of the CNN used. Figure taken from Ref. [Aurisano et al., 2016].

NOvA is not the only neutrino experiment that used a GoogLeNet-inspired network in order to classify their events. The NEXT collaboration [Adams et al., 2020a] started to use GoogLeNet as their first deep-learning-based algorithm used in the analyses, in this case, to perform signal/background separation of neutrino events. The proposed method outperformed conventional methods by a factor ranging from 1.2 to 1.6 [Renner et al., 2017]. For its part, the MicroBooNE collaboration [Acciarri et al., 2017b] performed in Ref. [Acciarri et al., 2017a] a detailed study on how state-of-the-art CNNs (AlexNet [Krizhevsky et al., 2012], GoogLeNet [Szegedy et al., 2014], Faster-RCNN [Ren et al., 2017], Inception-ResNet-v2 [Szegedy et al., 2016], and ResNet [He et al., 2015a]) can be successfully used for particle classification, particle and neutrino detection, and neutrino event identification of LArTPC images. The MicroBooNE collaboration have been active in the last few years looking for deep-learning applications to neutrino physics. They have developed several approaches, including a semantic segmentation CNN for pixel-level electromagnetic particle identification [Adams et al., 2019], where a U-ResNet model (a combination of U-Net and ResNet for semantic segmentation) is used to classify each pixel of an image into a track, a shower, or a background pixel (see Fig. 2.5); and a CNN for multi-particle identification [Sword-Fehlberg, 2017], which looks for the following particles in the input image: electrons, photons, muons, and charged pions.

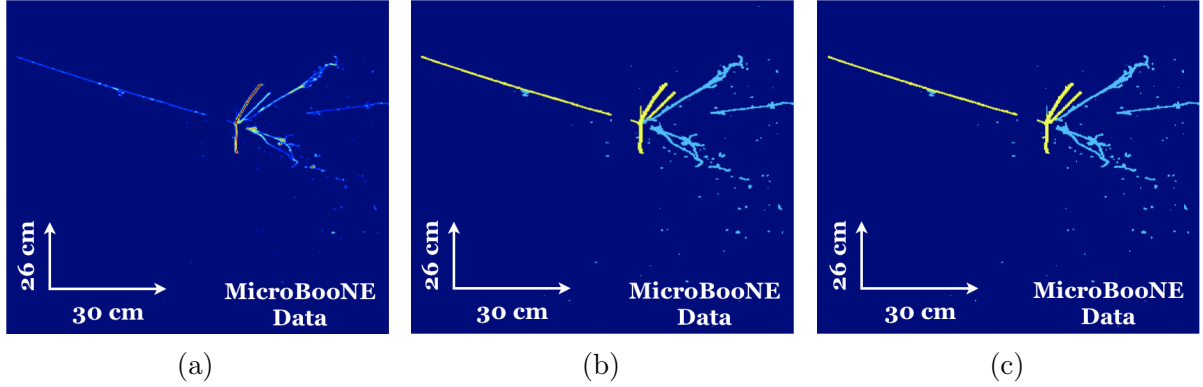


Fig. 2.5 Semantic segmentation in MicroBooNE. Example of a neutrino event display from a CC π^0 candidate detector data. (a) input image; (b) physicist labels, with tracks in yellow and showers in cyan; (c) labels predicted by the network, with tracks in yellow and showers in cyan. Figure taken from Ref. [Adams et al., 2019].

The IceCube experiment [Aartsen et al., 2017] uses deep-learning approaches to guide their analyses too. Among other works, in Refs. [Huennefeld, 2018, 2019] the IceCube collaboration use CNNs with 2-D and 3-D convolutions to reconstruct the energy of muon-neutrino events, while in Ref. [Kronmueller and Glauch, 2020] they use a sophisticated CNN inspired by the Google's Inception-ResNet architecture [Szegedy et al., 2016] to categorise events into different types. EXO-200 [Auger et al., 2012] was an experiment that

used CNNs for different purposes, such as energy and position reconstruction [Delaquis et al., 2018]. The nEXO experiment, the successor of EXO-200, showed that CNNs outperform traditional machine-learning algorithms like boosted-decision trees (BDTs) for effectively discriminating signal and background events [Li et al., 2019].

The sparsity of neutrino interactions and solutions

Despite the success of CNNs in the neutrino world, images of neutrino interactions are typically very sparse as, due to the large size of the detectors, only a small number of readout channels will have detected signals for each event, resulting in relatively few pixels with non-zero values. For instance, in the case of the detector presented in Chapter 4, the average occupancy of the detector for a neutrino interaction is less than 0.02%. Thus, much of the computation time is spent unnecessarily applying convolutions to a large number of patches of pixels with zero values, an operation that always returns zero.

A popular approach to deal with the sparsity of neutrino interactions is the submanifold sparse convolutional network (SSCN) [Graham and van der Maaten, 2017]. Standard “dense” CNNs are very inefficient when applied on images of neutrino interactions, whereas SSCNs require considerably less computation and report almost identical (or even better) results in terms of accuracy [Graham et al., 2017]. Some neutrino experiments have improved their reconstruction deep-learning algorithms by moving to SSCNs. For example, MicroBooNE recently updated the implementation of their semantic segmentation CNN [Adams et al., 2019] to an SCNN-based model [MicroBooNE Collaboration, 2020], reporting improvements for inference by a factor of 354 and 33 in memory and wall-time, respectively, as shown in Table 2.1 [Dominé and Terao, 2020]. The NEXT collaboration are also exploring the idea of using an SSCN for track classification [Kekic et al., 2020].

	Dense	SSCN			
Batch size	4	4	64		
Image size [pixels]	192	192	192	512	768
Accuracy [%]	92	94	98	99	99
Memory (test) [GB]	16	0.04	0.2	0.7	1.3
Memory (train) [GB]	26x4	0.21	1.3	5.1	9.3
Wall-time (test) [s]	3.3	0.10	0.7	2.4	4.4
Wall-time (train) [s]	25	0.21	1.2	5.0	8.8

Table 2.1 SSCN vs dense MicroBooNE U-ResNet scalability with the 3D image spatial size on an NVIDIA V100 GPU. The dense version could not fit 3D images of size 512px nor 768px on a single GPU. Table adapted from Ref. [Dominé and Terao, 2020].

An alternative approach for handling sparse data is to represent hits (or voxels) as nodes in a graph. Figure 2.6 shows a comparison of the 3D-CNN and graph data

structures, as well as the radial search method used for defining edges between nodes. One class of deep-learning algorithms that operate on graphs are called graph neural networks (GNNs) [Sperduti and Starita, 1997; Zhou et al., 2018]. The application of GNNs to data from neutrino experiments has been recently demonstrated by the IceCube experiment in order to identify entire events as atmospheric neutrino interactions, outperforming a 3D CNN [Choma et al., 2018].

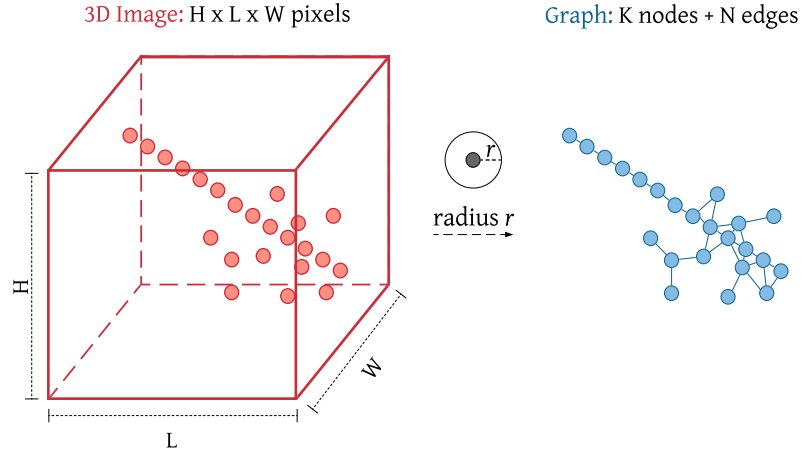


Fig. 2.6 Data and computation size comparison between a 3D image and a graph. The size of the 3D image on the left is fixed ($H \times L \times W$) regardless of the number of hits as CNNs require fixed image sizes. The connected graph shown on the right is a much more efficient representation of the data. Each hit is represented as a graph node and connections, called edges, are made between neighbouring hits within a sphere of radius r .

The work in Ref. [Drielsma et al., 2020a] also shows an application of GNNs for both node and edge classification for a neutrino detector. Other GNN-based studies have been performed for particle reconstruction in high energy physics detectors [Farrell et al., 2018; Ju et al., 2020; Qasim et al., 2019]. To the best of our knowledge, the approach we present in Chapter 4 is the second attempt to use GNNs for node classification in neutrino experiments.

End-to-end deep-learning chain for neutrino reconstruction

DeepLearnPhysics [DeepLearnPhysics, 2020] is an international collaboration focused on the application of deep-learning techniques to the analysis of data from particle-physics experiments. Their current goal is to investigate solutions for the study and reconstruction of data collected by LArTPCs, and they have already developed an end-to-end deep-learning chain for neutrino reconstruction using 3D particle interaction images as an input:

- *Pixel feature*: the first step in the reconstruction chain is to extract voxel-level information in 3D images: categorise each voxel as track-like or shower-like [Dominé and Terao, 2020], and locate points of interest (i.e., predict particle endpoints) [Dominé et al., 2020]. An example is shown in Fig. 2.7.

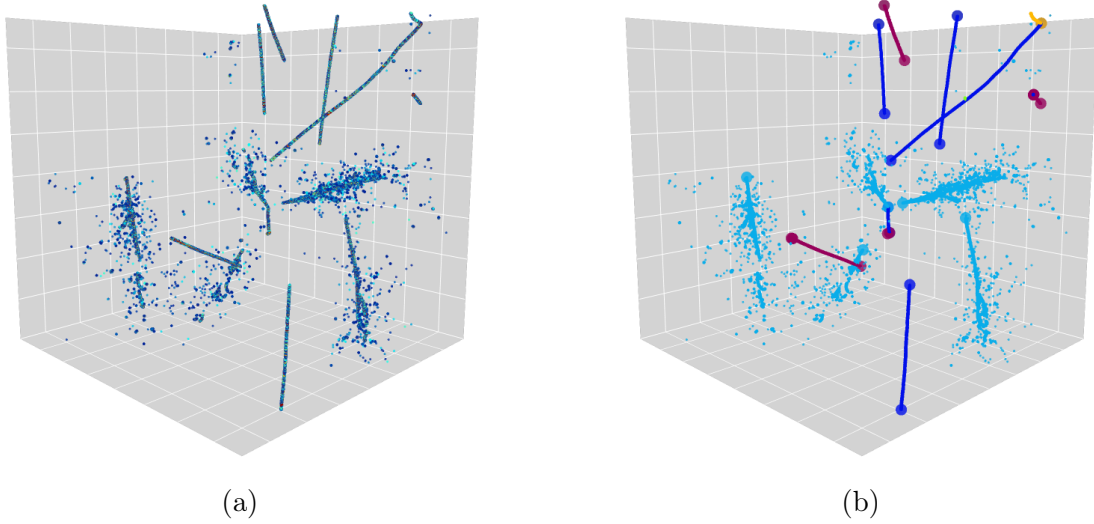


Fig. 2.7 Simulated LArTPC event. (a) Network input: shows energy deposits from charged particle trajectories, whose colour corresponds to an energy scale; (b) labels: the goal of the pixel-feature step of the reconstruction chain. The dots represent the points of interest, where the voxel colours correspond to their class. Figure taken from Ref. [Dominé et al., 2020].

- *Pixel clustering*: the second step is to group pixels by finding a path between particle endpoints using CNNs [Koh et al., 2020], and identify showers using GNNs [Drielsma et al., 2020a].
- *Particle clustering*: the last step in the reconstruction chain is to identify each interaction using edge classification, and use node classification for identifying each particle type [Drielsma et al., 2020b].

The full end-to-end reconstruction chain, gathering the three steps above into the same workflow, is depicted in Fig. 2.8.

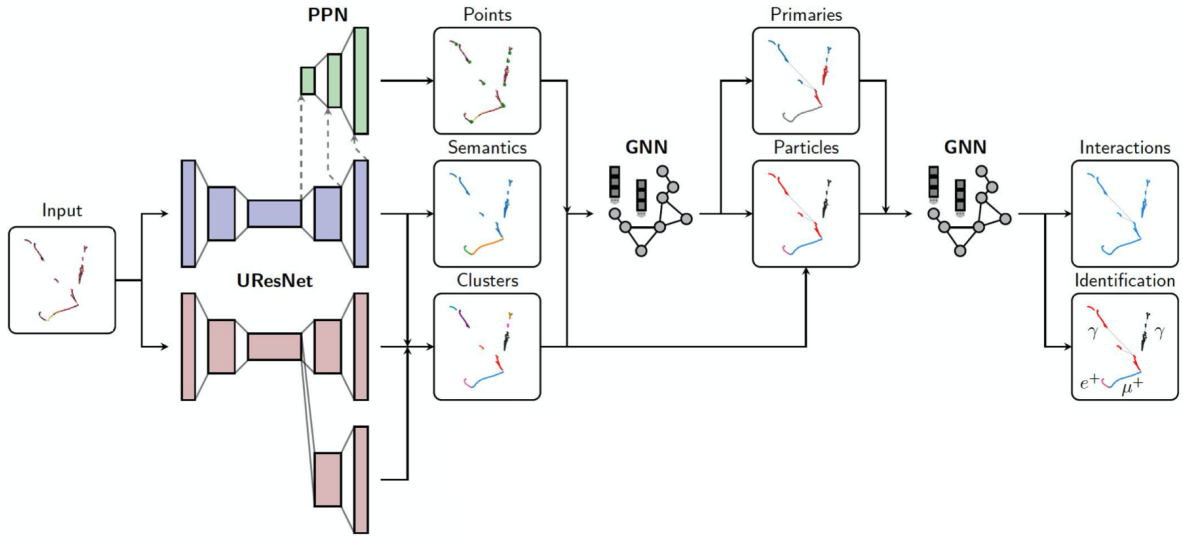


Fig. 2.8 Full end-to-end deep-learning reconstruction chain for 3D LArTPC neutrino events (from left to right). In the first step, the U-ResNet and PPN networks perform the voxel categorisation and particle-endpoint location, respectively; then, the first GNN clusters pixels and looks for showers; finally, the second GNN identifies interactions and particle types. Figure taken from Ref. [Terao, 2020].

2.2 High-performance computing for deep learning

High-performance computing (HPC) allows scientific and industrial deep-learning applications to handle vast volumes of data by scaling their computation. Scaling the computation of deep-learning models has been gaining attention over the last few years, while being able to distribute the training of deep neural networks efficiently is still a major challenge [Mayer and Jacobsen, 2020]. Training models asynchronously is usually the best way to efficiently train neural networks in a distributed manner (see Sec. A.4 from Appendix A for more details). The work in Ref. [Pumma et al., 2020] studies the scalability limitations of TensorFlow and Horovod in terms of load imbalance (i.e., some components have to contend with others for hardware resources during gradient synchronisation) and proposes a number of solutions to address these limitations. Nevertheless, the load imbalance limitations discussed usually are not focused on the I/O bottleneck caused by the input data (Fig. 2.9 shows an example of I/O overhead in a distributed deep-learning scenario), but on the worker synchronisation and how to take advantage of the processing units available.

Current distributed filesystems are not designed for deep-learning jobs scaling requirements [Aizman et al., 2019], mainly due to long queueing delays caused by big-data

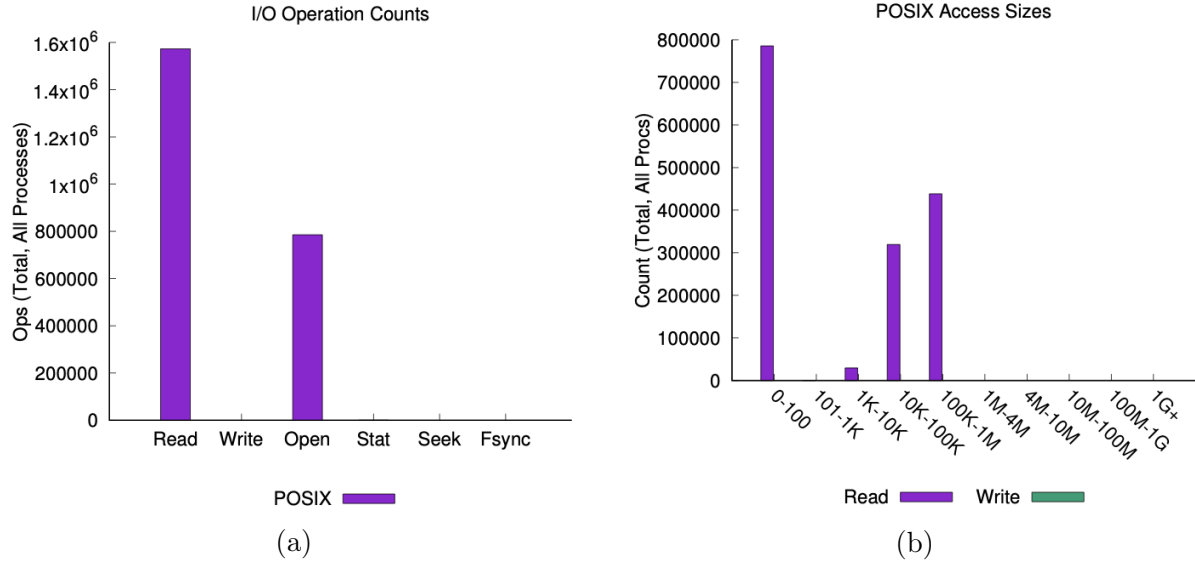


Fig. 2.9 TensorFlow I/O pattern on ImageNet data [Deng et al., 2009] using Horovod for distributing the computation across six workers. The underlying parallel filesystem used is BeeGFS. (a) Ops vs I/O operations; (b) operation count vs file size. Most I/O overhead is caused by file open and read operations. Figure taken from Ref. [Chowdhury et al., 2019].

schedulers on large GPU clusters [Gu et al., 2019]. Authors in Ref. [Pumma et al., 2017; Pumma et al., 2019] show data I/O can take up to 90% of the total training time, and present a modification to Lightning Memory-Mapped Database (LMDB), the underlying embedded database, in order to reduce the I/O overload. Other solutions are introduced in Ref. [Aizman et al., 2019], where authors propose a new distributed filesystem to efficiently access large datasets; and in Ref. [Zhang et al., 2019], where authors present FanStore, a transient runtime object store leveraging local storage and enabling efficient and scalable deep-learning trainings.

There are two main differences between the approach studied in the first half of Chapter 6 and the solutions above. First, the method analysed does not require the deployment of any underlying distributed storage system. For example, authors of [Zhang et al., 2019] claim in their GitHub page that the current version of FanStore only supports a limited number of platforms, so the work in progress consists of porting the software to new platforms. Second, the approach studied is inspired by the idea of ensuring the computation task to be close to the data source and not vice-versa, leveraging from data locality, taking into account the global impact of random file accesses in deep learning. In this way, we want to avoid transferring large amounts of data over the network (the architecture introduced in Ref. [Aizman et al., 2019] does not reduce the network bottleneck).

2.2.1 Deep learning on hardware accelerators

There is a growing demand for computing resources needed by modern deep-learning methods; consequently, the use of hardware accelerators has begun. Regarding the underlying hardware where deep-learning models are typically deployed, we can find many types of accelerators, from general-purpose computing units, such as standard GPUs [Strigl et al., 2010], to specialized devices designed to speed up deep-learning workloads [Google, 2020] (see Fig. 2.10 for a comparison between flexibility and computational efficiency of neural networks for different hardware implementations).

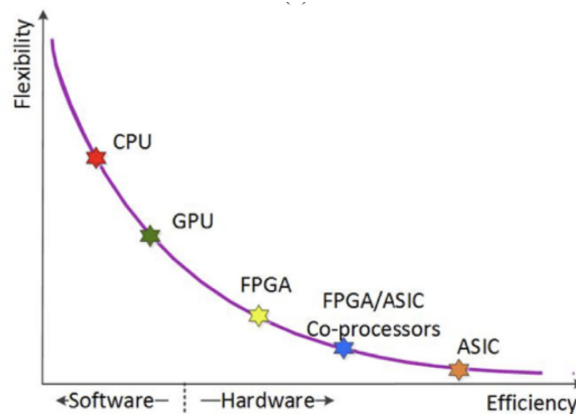


Fig. 2.10 Flexibility vs computational efficiency for different hardware processing units. As can be seen, there is a trade-off between the two metrics. Figure taken from Ref. [Feng et al., 2019].

Some work has already been done to try to distribute the computation on non-traditional devices, such as Google TPUs [You et al., 2019] or Intel’s Xeon processors [Serpa et al., 2018]. [Wang et al., 2020] and [Chen and Ran, 2019] even study the possibility of deploying deep-learning applications at the edge of the network, while [Chen et al., 2020] gives an overview on how to address the convergence between deep learning and Internet of things (IoT). The use of field-programmable gate arrays (FPGA) plays a crucial role in hardware accelerators. Programming custom logic directly on the chip allows us to obtain maximum performance from the hardware without needing to manufacture an application-specific integrated circuit (ASIC). As a disadvantage, FPGAs are generally challenging to program, and their capacity remains very limited, but this is changing in the last years [Trimberger, 2015].

The high-level synthesis (HLS) language introduces a more intuitive way for even non-experts to program FPGAs in a C/C++ like code [Martin and Smith, 2009]. Some techniques allow converting neural networks to HLS in a quasi-automated way [Duarte et al., 2018]. The work that we present in the second half of Chapter 6 is an efficient way

to implement deep-neural networks, especially CNNs, into FPGAs, avoiding the complex part of hardware programming.

Chapter 3

Event identification using convolutional neural networks

The Deep Underground Neutrino Experiment (DUNE) is a next-generation neutrino oscillation experiment that aims to measure CP -violation in the neutrino sector as part of a wider physics programme (see Sec. B.3 from Appendix B for more details). This chapter presents a deep-learning approach based on a convolutional neural network developed to provide highly efficient and pure selections of electron neutrino and muon neutrino charged-current interactions in the DUNE experiment.

3.1 The DUNE Convolutional Visual Network

The DUNE Convolutional Visual Network (CVN) classifies neutrino interactions in the DUNE far detector (FD) through image recognition techniques. Prior to the work presented in this chapter, the DUNE experiment used a convolutional neural network (CNN) to select electron neutrino and muon neutrino interactions in the DUNE FD. The CNN algorithm was ported from the NOvA experiment, and the model was inspired by the GoogLeNet architecture, reporting a peak signal efficiency of $\sim 73\%$ and $\sim 82\%$ for CC ν_e and CC ν_μ , respectively. The reported results were not sufficient to reach the predicted sensitivity of the experiment to CP -violating effects, which led the experiment to explore alternative approaches. The investigation culminated with the development of the current DUNE CVN, a more sophisticated CNN-based algorithm, which is described in detail in the following sections.

3.1.1 Inputs to the CVN

The DUNE far detectors are LArTPCs that have three readout views, which generate three different images of each neutrino interaction. The CVN was therefore designed to take three input images, as shown for an example CC ν_e interaction in Fig. 3.1. The three inputs are 500×500 pixel images of simulated neutrino interactions with one image produced for each of the three readout views of the LArTPC. The images are produced at the hit-level stage of the reconstruction algorithms and are hence independent of any potential errors in high-level reconstruction such as clustering, track-finding and shower reconstruction. The images are produced in (wire number, time) coordinates, where the wire number is simply the detector readout wire on which the reconstructed hit was detected, and the time is the interval from when the interaction happened to when the hit was detected on that wire (given by the peak time of the hit). Each of the input images have depth one (they are greyscale images), where the pixel value corresponds to the hit charge. The images are presented here using a colour scale, ranging from pale yellow for low charge to bright red for high charges. Pixels with zero value are shown in white. Each pixel represents approximately 5 mm in the wire coordinate due to the spatial separation of the wires in the readout plane, and the time coordinate is down-sampled to approximately correspond to the same 5 mm size after consideration of the electron drift velocity within the LArTPC.

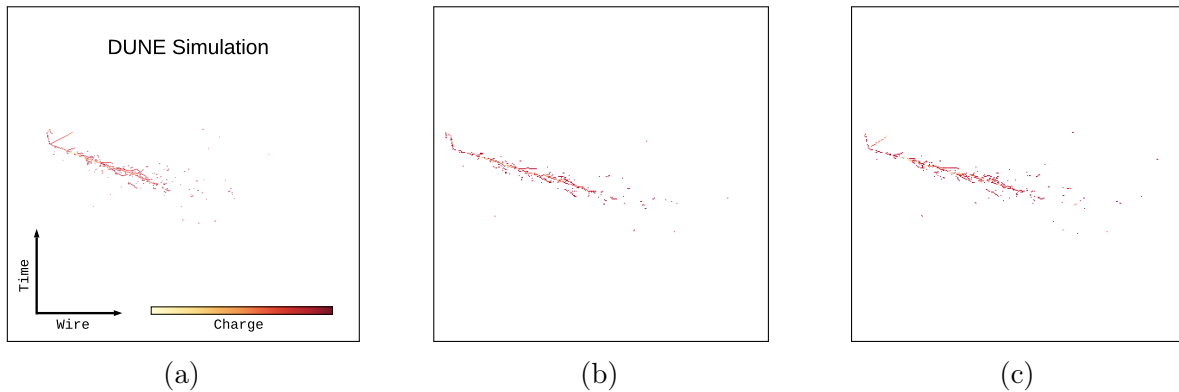


Fig. 3.1 A 2.2 GeV CC ν_e interaction shown in the three readout views of the DUNE LArTPCs showing the characteristic electromagnetic shower topology. a) View 0: induction plane; b) view 1: induction plane; c) view 2: collection plane. The horizontal axis shows the wire number of the readout plane and the vertical axis shows time. The colour scale shows the charge of the energy deposits on the wires.

Convolutional neural networks normally operate on fixed-size images, hence the neutrino interaction images must all be of a fixed size. The detectors are very large so it is unfeasible to include the entire detector in each image without downsampling in the

wire coordinate (this corresponds to a significant loss of information). To facilitate this, interactions that span more than 500 wires in a given view are cropped to fit in 500×500 pixel images. The steps below are used to find the optimal choice of 500 wires:

1. For each readout wire, integrate the charge from all detected hits.
2. Scan from low wire number, where low wire number corresponds to the upstream end of the detector, to high wire number and check the following 20 wires for recorded signals. If fewer than five of the 20 subsequent wires have no signals then this wire is chosen as the first column of the image.
3. If no wire satisfies the requirement in Step 2, choose the continuous 500 wire range that contains the most deposited charge.

For the time axis, a window of $3200 \mu\text{s}$ centred on the mean time of the hits is formed and divided into 500 bins that form the 500 pixels. All interactions are contained within this time window, hence no cropping is required.

In order to provide high quality images for the training and testing samples, images were only produced for events that have their true neutrino interaction vertex within the detector fiducial volume¹ described in Ref. [Abi et al., 2020d]. Additionally, all events with one or more images containing fewer than 10 non-zero pixels were not considered. Figure 3.2a shows a signal CC ν_μ interaction, and example NC background images containing a long $\pi^\pm$ track and a π^0 are given in Figs. 3.2b and 3.2c, respectively.

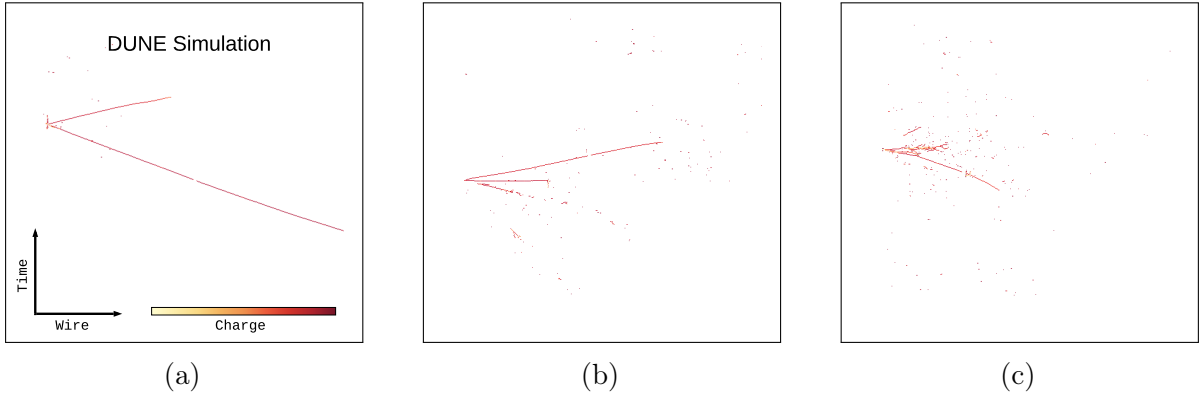


Fig. 3.2 Three interactions shown in the collection view: a) a 1.6 GeV signal CC ν_μ interaction; b) a 2.2 GeV NC interaction with a long π^+ track; c) a 2.4 GeV NC interaction with one π^0 . The NC interactions shown in b) and c) form the primary backgrounds to CC ν_μ and CC ν_e event identification, respectively.

The number of pixels in the images was chosen to maximise the size of the image whilst ensuring that the memory usage during training and inference of the network was

¹Fiducial volume: the volume in which the interactions are fully contained within the instrumented region of the detector.

manageable. The spatial dimension of the images covers 2.5 m, meaning any tracks with projected lengths in the readout planes above 2.5 m will not be fully contained within the image, as is the case for the majority of muon tracks, including the one shown in Fig. 3.2a. However, much of the key information for neutrino event classification comes from the region close to the interaction vertex, so this choice of image size does not significantly impact the classification performance.

3.1.2 Network architecture

A simple overview of the architecture is shown in Fig. 3.3. The detailed architecture of the CVN is based on the 34-layer version of the SE-ResNet architecture, which consists of a standard ResNet (Residual neural network) architecture [He et al., 2015a, 2016] along with Squeeze-and-Excitation blocks [Hu et al., 2017]. Residual neural networks allow the n^{th} layer access to the output of both the $(n - 1)^{\text{th}}$ layer and the $(n - k)^{\text{th}}$ layer via a residual connection, where k is a positive integer ≥ 2 . This is an important feature as it allows the fine-grained detail in the input images to be propagated into the deeper layers of the network.

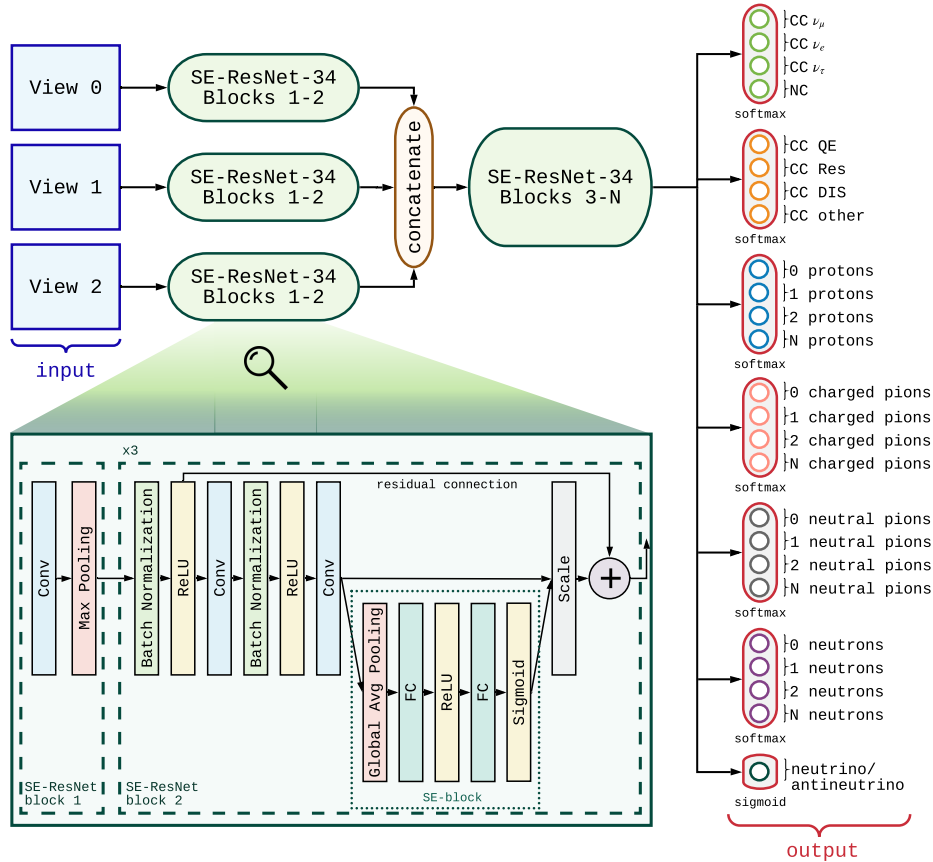


Fig. 3.3 Simplified diagram of the DUNE CVN architecture.

The DUNE CVN differs from the architectures of other residual networks discussed in the literature [He et al., 2015a, 2016] in the following ways:

- The input and the shallower layers of the CVN are forked into three branches - one for each readout view - to let the model learn features from each individual view (see section 3.1.1 for more details). The outputs of the three branches are merged together by using a concatenation layer² that forms the input to the deeper layers of the model, as shown in Fig. 3.3.
- The CVN returns scores for each event through seven individual outputs (see section 3.1.3 and Fig. 3.3 for more details). Since the deeper layers of the CVN contain the network parameters³ that are simultaneously in charge of the classification for the different outputs of the network, some outputs might take advantage of the learning process of other outputs to improve their performance. Also, a multi-output network lets us weight the outputs in order to make the network pay more attention to some specific outputs (see section 3.1.4 for more details).
- Each of the three branches (blocks 1-2, the shallower layers of the architecture shown in Fig. 3.3) consists of 7 convolutional layers, while the deeper layers (blocks 3-N in Fig. 3.3) consist of 29 convolutional layers, making a total of 50 convolutional layers for the entire network. The total number of trainable parameters of the network is 22,055,744.

3.1.3 Outputs from the CVN

As shown on the right of Fig. 3.3, there are seven outputs from the CVN, each consisting of a number of neurons with values v_i for $i = 1 \rightarrow n$ where n is the number of neurons. The sum of neuron values for each output (except for the last output since it consists of a single neuron) is given by $\sum_{i=1}^n v_i = 1$ such that each value of a neuron within a single output gives a fractional score that can be used to classify images.

The first output, which has four neurons to classify the flavour of the neutrino interaction, is the primary output and it is the only one used in the oscillation sensitivity analysis⁴ presented in Refs [Abi et al., 2020a] and [Abi et al., 2020d]. The other outputs are included in the architecture for potential use in future analyses.

- The 1st output (4 neurons) returns scores for each event to be one of the following flavours: CC ν_μ , CC ν_e , CC ν_τ and NC. This is the primary output of the network used for the main goal of neutrino interaction flavour classification.

²Concatenation layer: network layer that concatenates a list of inputs along the depth axis.

³Network parameters: coefficients of the model learnt during the training stage, also known as weights.

⁴Official oscillation sensitivity analysis published by the DUNE collaboration.

- The 2nd output⁵ (4 neurons) returns scores for each event to be one of the following interaction types: CC quasi-elastic (CC QE), CC resonant (CC Res), CC deep inelastic (CC DIS) and CC other.
- The 3rd output (4 neurons), returns scores for each event to contain the following number of protons: 0, 1, 2, >2.
- The 4th output (4 neurons), returns scores for each event to contain the following number of charged pions: 0, 1, 2, >2.
- The 5th output (4 neurons), returns scores for each event to contain the following number of neutral pions: 0, 1, 2, >2.
- The 6th output (4 neurons), returns scores for each event to contain the following number of neutrons: 0, 1, 2, >2.
- The 7th output⁵ (1 neuron) returns the score for each event to be a neutrino as opposed to an antineutrino.

Outputs 2, 6 and 7 are not considered in the analyses presented here and are hence not further discussed, but they are included in the training and the overall loss calculations. The prediction of an event as a given underlying (anti)neutrino interaction is highly model-dependent and not as important as the number of final-state particles that can be observed in the detector, hence output 2 is not used. The neutron counting is very difficult since it is hard to define whether a neutron interaction would be visible and identifiable in the detector, so this output will not be used until it has been shown to work reliably. Finally, the antineutrino vs neutrino output is not likely to provide highly efficient or pure event selections since there is only a weak dependence on the event observables to try to differentiate neutrinos and antineutrinos.

3.1.4 Training details

The CVN⁶ was trained using Python 3.5.2 and Keras 2.2.4 [Chollet et al., 2015] on top of Tensorflow 1.12.0 [Abadi et al., 2016], on eight NVIDIA Tesla V100 GPUs. Stochastic gradient descent (SGD) is used as the optimiser, with a mini-batch size of 64 events (192 views), a learning rate of 0.1 (divided by 10 when the error plateaus, as suggested in Ref. [He et al., 2015a]), a weight decay of 0.0001, and a momentum of 0.9⁷. The network was trained/validated/tested on 3,212,351 events (9,637,053 images/views), consisting of 27% CC ν_μ , 27% CC ν_e , 6% CC ν_τ and 40% NC, from a single Monte Carlo sample as

⁵Output to ‘sub-classify’ CC events. NC events are not considered for the training of this output.

⁶A small data release of the code is available at <https://github.com/DUNE/dune-cvn>.

⁷See Appendix A for description of optimisers and associated terminology.

follows: training ($\sim 98\%$), validation ($\sim 1\%$) and test ($\sim 1\%$). The sample of events is an MC prediction for the DUNE unoscillated FD neutrino event rate (flux times cross section) distribution in FHC beam mode as described in Ref. [Abi et al., 2020a]. Samples where the input fluxes to the MC are “fully oscillated” (i.e. all ν_μ are replaced with ν_e , or all ν_μ are replaced with ν_τ) are also used (these samples are usually weighted by oscillation probabilities and combined to produce oscillated FD event rate predictions). Analogous versions of each input sample are used for the RHC beam mode. For training purposes all CC ν_e events were considered signal since the intrinsic beam ν_e are indistinguishable from signal (appearance) ν_e at any given energy. The results presented in the following sections use a statistically independent Monte Carlo sample.

The individual cost functions for the different outputs that were used for training the model, as well as the overall cost function, are given below (see Secs. A.1 and A.3 from Appendix A for more details):

- Neutrino flavour ID, interaction type⁸, proton count, charged pion count, neutral pion count, neutron count cost functions (J_1, J_2, J_3, J_4, J_5 , and J_6 , respectively): categorical cross-entropy, the cost function needed for multi-class classification.

$$J_1 = J_2 = J_3 = J_4 = J_5 = J_6 = -\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^c y_j^{(i)} \log \hat{y}_j^{(i)} \quad (3.1)$$

- Neutrino/antineutrino ID cost function⁸ (J_7): binary cross-entropy, the cost function needed for binary classification.

$$J_7 = -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \quad (3.2)$$

- Overall cost function:

$$J = \sum_{i=1}^o w_i J_i = w_1 J_1 + w_2 J_2 + w_3 J_3 + w_4 J_4 + w_5 J_5 + w_6 J_6 + w_7 J_7 \quad (3.3)$$

where:

- $\mathbf{y}^{(k)}$: true values of a specific output for the k -th training example.
- $\hat{\mathbf{y}}^{(k)}$: predicted values of a specific output for the k -th training example.
- m : number of training examples $\{\mathbf{X}^{(1)}, \mathbf{y}^{(1)}\}, \{\mathbf{X}^{(2)}, \mathbf{y}^{(2)}\}, \dots, \{\mathbf{X}^{(m)}, \mathbf{y}^{(m)}\}$, where $\mathbf{X}^{(k)}$ means the input readout views for the k^{th} training example.
- c : number of classes corresponding to a specific output $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_c$.

⁸A mask is applied to only consider true CC events during the loss computation.

- o : number of outputs of the network; the CVN has seven different outputs.
- $\mathbf{w}$: output weights; vector of length o (not to be confused with model weights).

The CVN was trained over a period of 15 epochs for ~ 4.5 days (7 hours per epoch), and similar classification performance was obtained for the training and test samples. Figure 3.4 shows the loss and accuracy training and validation results for the four main CVN outputs, where accuracy is defined as the fraction of events correctly classified for a given output. The red vertical lines show the epoch at which the CVN weights were taken for the model used in the presented analysis. After that epoch, the validation accuracy remains constant and there are small signs of overtraining. The relatively small difference between training and validation seen at epoch 10 has a negligible effect.

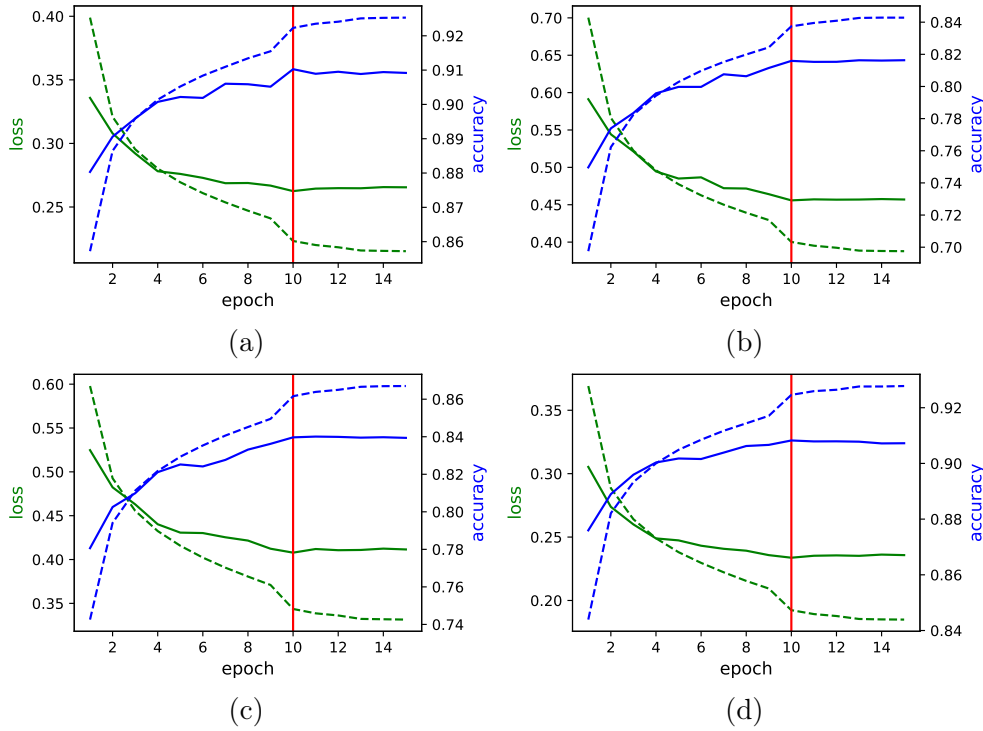


Fig. 3.4 Loss and accuracy results for training (dashed lines) and validation (solid lines), given for the four main CVN outputs: a) flavour, b) protons, c) charged pions, d) neutral pions. The red vertical lines guide the eye to the network results at epoch 10, after which flavour classification performance of the validation sample does not improve.

3.1.5 Feature maps

To study how the CVN is classifying the interactions it is advisable to look at feature maps⁹ at different points in the network architecture. An example is shown in Fig. 3.5

⁹A feature map is the output activations for a given filter of a convolutional neural network.

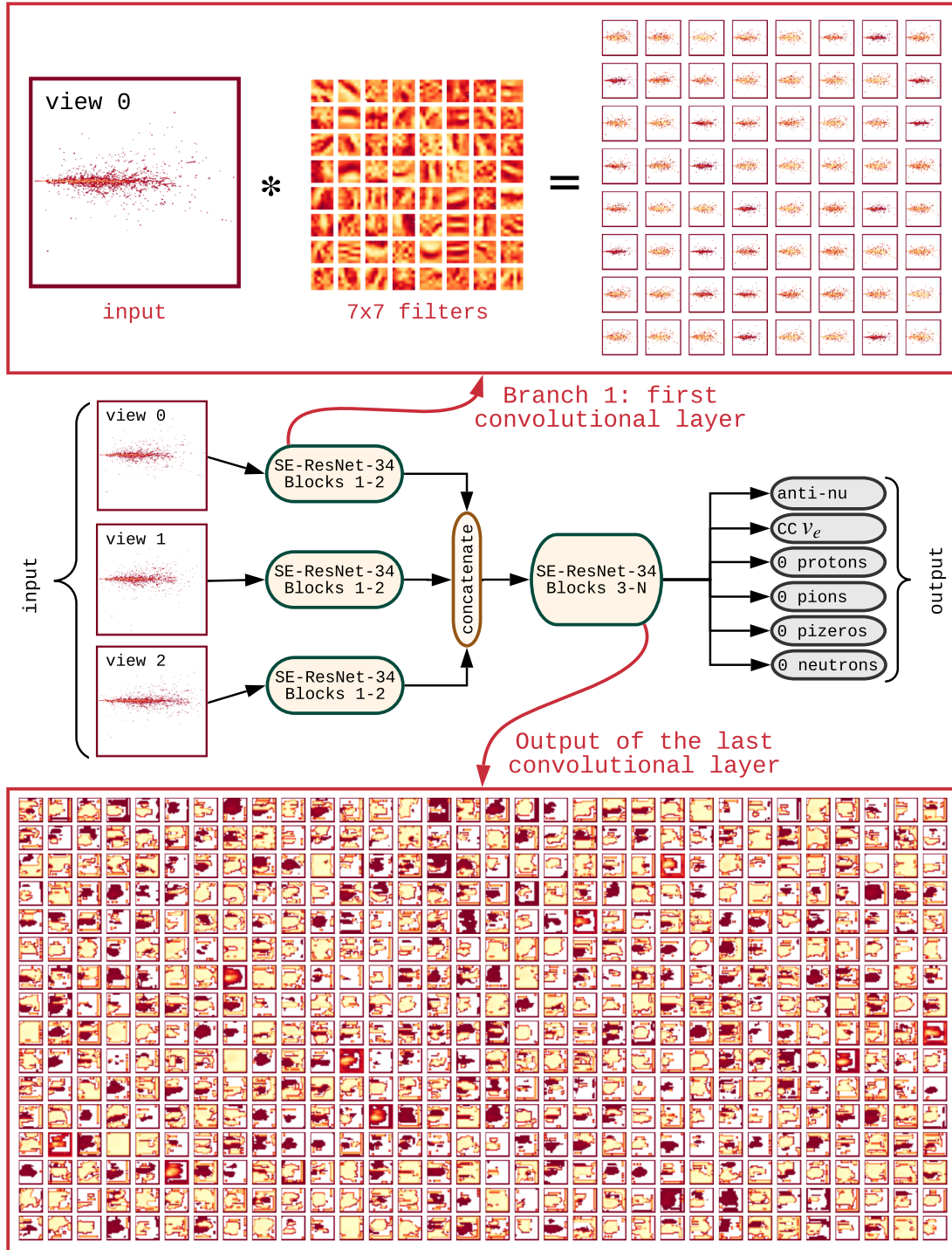


Fig. 3.5 Visualisations of the feature extraction in the CVN for a 12.2 GeV CC $\bar{\nu}_e$ interaction. The top box shows the output from the first convolutional layer of the first branch of the network: 64 convolution kernels of size 7x7 each are applied to the image, resulting in 64 different feature maps. The bottom box shows the 512 feature maps produced by the final convolutional layer.

for a CC $\bar{\nu}_e$ interaction, demonstrating the position from which two sets of feature maps are viewed within the network. The set of images in the top right shows the response of the filters in the first convolutional layer to the input electromagnetic shower image, where red shows a high response to a given filter, and yellow shows a low response.

Across the different particle types and event topologies, the filters respond to different components in the images. The 512 feature maps from the final convolutional layer, shown at the bottom of Fig. 3.5 for the aforementioned CC $\bar{\nu}_e$ interaction, are much more abstract in appearance since the input images have passed through many convolutions and have hence effectively been down-sampled to a size of 16×16 pixels from their original 500×500 pixel size.

3.2 Neutrino flavour identification performance

The primary goal of the CVN is to accurately identify CC ν_e , CC $\bar{\nu}_e$, CC ν_μ and CC $\bar{\nu}_\mu$ interactions for the selection of the samples required for the neutrino oscillation analysis. The values of the neurons in the flavour output give the score for each neutrino interaction to be one of the neutrino flavours. The CVN CC ν_e score distribution, $S(\nu_e)$, is shown for the FHC beam mode (left) and RHC (right) in Fig. 3.6 for all interactions with a reconstructed event vertex within the FD fiducial volume, as described in Ref. [Abi et al., 2020d].

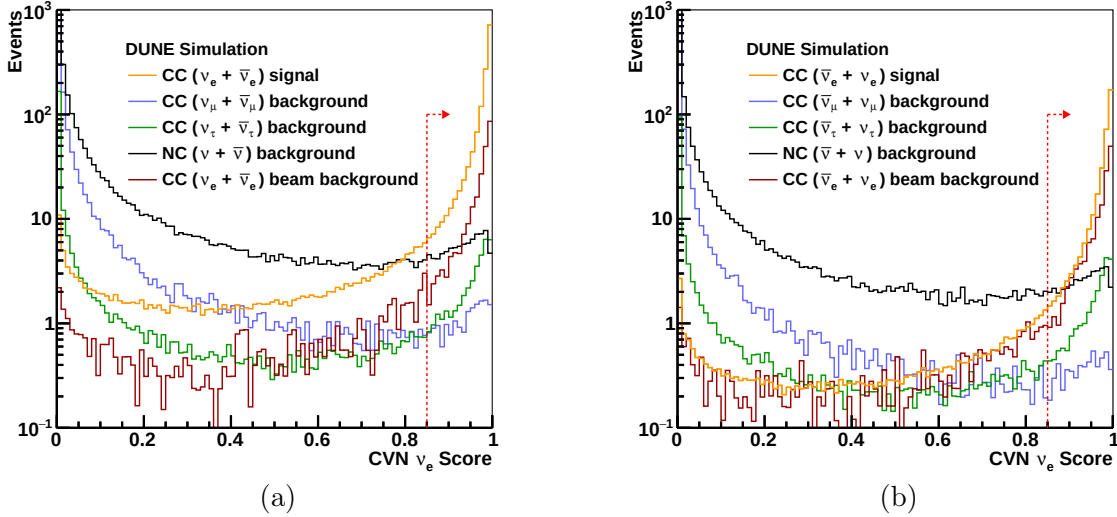


Fig. 3.6 The number of events as a function of the CVN CC ν_e classification score shown for FHC (a) and RHC (b) beam modes. For simplicity, neutrino and antineutrino interactions have been combined within each histogram category. A log scale is used on the y-axis, normalized to 3.5 years of staged running, and the arrows denote the cut values applied for the DUNE TDR and LBL analyses [Abi et al., 2020a,d].

The contributions from neutrino and antineutrino components for each flavour are combined since the detector can not easily distinguish between them. Very clear separation is seen between the signal (CC ν_e and CC $\bar{\nu}_e$) interactions and the background interactions including those from NC ν and NC $\bar{\nu}$ events. The beam CC ν_e background is seen to peak in the same way as the CC ν_e signal, which is expected since both arise from the same type of neutrino interaction. Figure 3.7 shows the corresponding plots for $S(\nu_\mu)$ for FHC and RHC beam modes for the same set of interactions. In all four histograms the signal interactions are peaked closely to score values of unity and the backgrounds lie close to zero score, as expected.

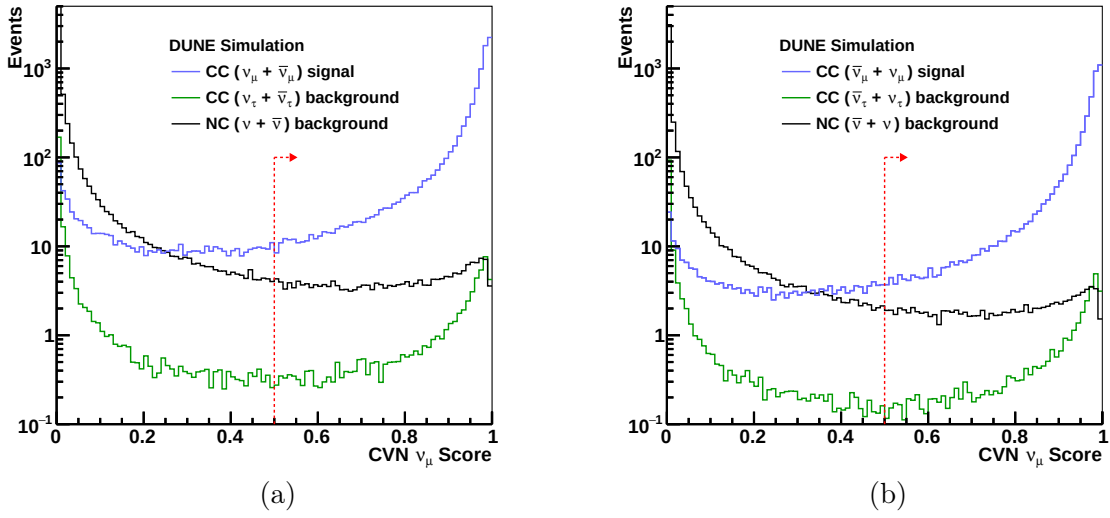


Fig. 3.7 The number of events as a function of the CVN CC ν_μ classification score shown for FHC (a) and RHC (b) beam modes. For simplicity, neutrino and antineutrino interactions have been combined within each histogram category. Backgrounds from CC ν_e interactions are negligible and not shown. A log scale is used on the y-axis, normalised to 3.5 years of staged running, and the arrows denote the cut values applied for the DUNE TDR and LBL analyses [Abi et al., 2020a,d].

The CC ν_e event selection criteria are chosen to maximise the oscillation analysis sensitivity to CP -violation [Abi et al., 2020d]. The optimisation was performed using a simple scan of cuts on $S(\nu_e)$ for a single true value of δ_{CP} . CP -violation sensitivity does not strongly depend on the selection criterion for $S(\nu_\mu)$ so this cut was chosen by visual inspection of Fig. 3.7. The resulting requirements are $S(\nu_e) > 0.85$ for an interaction to be selected as a CC ν_e candidate and $S(\nu_\mu) > 0.5$ for an interaction to be selected as a CC ν_μ candidate. These cut values are represented by the red arrows in Figs. 3.6 and 3.7. Since all of the flavour classification scores must sum to one, these two samples are mutually exclusive. The same trained CVN model and selection criteria are used for both FHC and RHC event selections.

Figure 3.8 shows the signal efficiency¹⁰ as a function of reconstructed energy (under the electron neutrino hypothesis, as discussed in Ref. [Abi et al., 2020b]) for the CC ν_e and CC $\bar{\nu}_e$ event selections. The efficiency for the CVN is shown compared to the predicted efficiency used in the DUNE Conceptual Design Report (CDR) [Acciarri et al., 2015], demonstrating that, across the most important part of the flux distribution (less than 5 GeV), the performance can exceed the CDR assumption. The efficiency in FHC (RHC) mode peaks at 90% (94%) and exceeds 85% (90%) for reconstructed neutrino energies between 2-5 GeV. Antineutrino interactions, on average, produce more energetic leptons and fewer hadrons than neutrino events, leading to greater lepton tagging efficiency with respect to neutrino-induced events. The training was optimized over the oscillation peak between 1 GeV and 5 GeV, and hence the CVN performs best in this region where the sensitivity to neutrino oscillations is greatest. The CDR analysis was based on a fast simulation that employed a parameterised detector response based on Geant4 single particle simulations, and a classification scheme that classified events based on the longest muon/charged pion track, or the largest EM shower if no qualifying track was present.

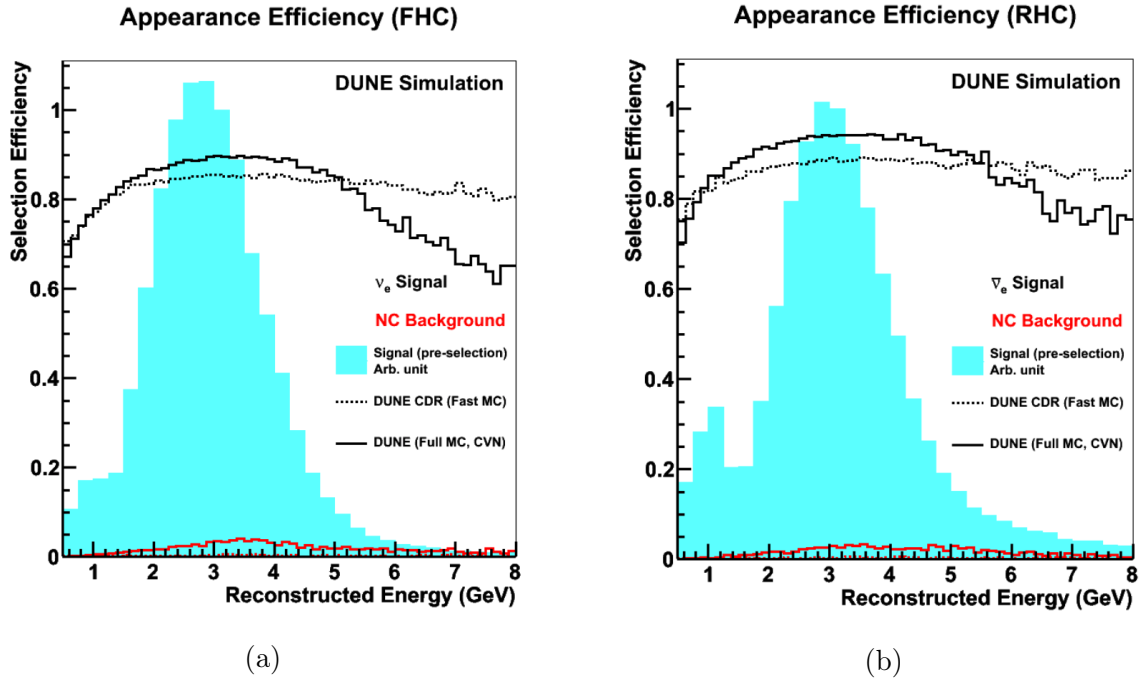


Fig. 3.8 The CC ν_e selection efficiency for FHC-mode (a) and RHC-mode (b) simulation with the criterion $S(\nu_e) > 0.85$. The solid (dashed) lines show results from the CVN (CDR) for signal CC ν_e and CC $\bar{\nu}_e$ events in black and NC background interactions in red. The cyan shaded region shows the oscillated flux to illustrate the most important regions of the energy distribution.

¹⁰Signal efficiency is here defined as the number of correctly identified CC ν_e events divided by the true number of CC ν_e events.

The optimised cut values permit a larger background component than the CDR analysis but the overall performance of the selection is increased due to the significantly improved signal efficiency. Considering all electron neutrino interactions (both appeared and beam background CC ν_e and CC $\bar{\nu}_e$ events) as signal interactions, the CVN has a selection purity¹¹ of 91% (89%) for the FHC (RHC) beam mode, assuming the normal neutrino mass ordering and $\delta_{CP} = 0$ [Abi et al., 2020d]. Figure 3.9 shows the corresponding selection efficiency for the CC ν_μ event selection. The selection has a maximum efficiency of 96% (97%) and exceeds 90% (95%) efficiency for reconstructed neutrino energies above 2 GeV for the FHC (RHC) beam mode.

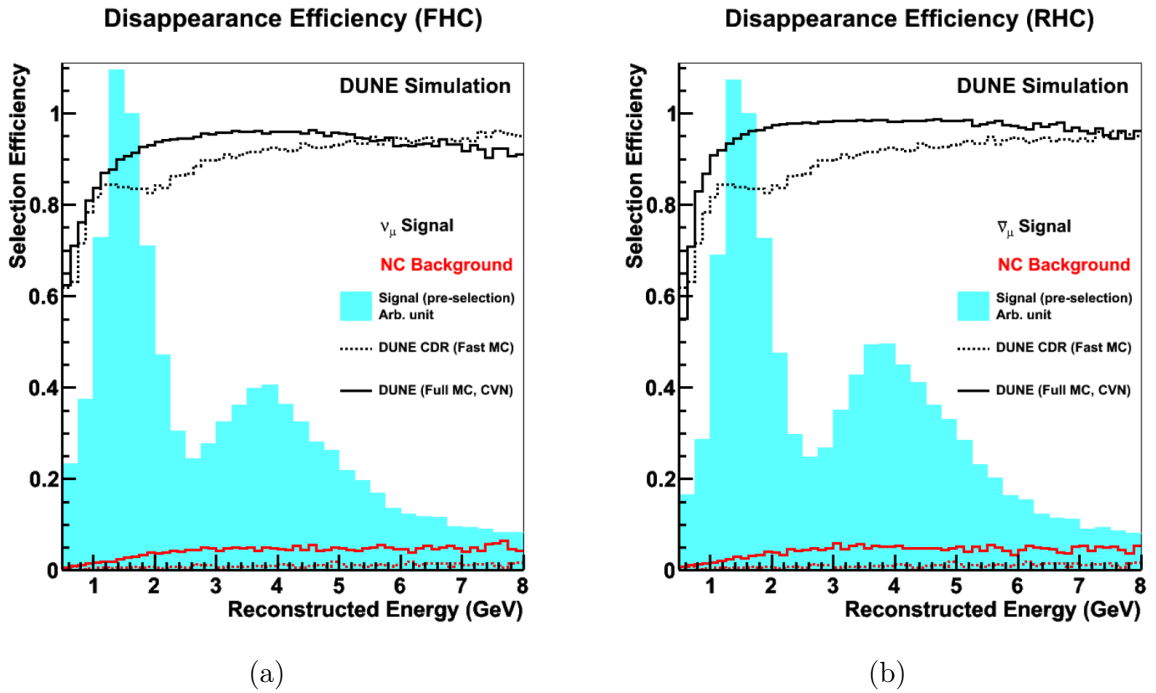


Fig. 3.9 The CC ν_μ selection efficiency for FHC-mode (a) and RHC-mode (b) simulation with the criterion $S(\nu_\mu) > 0.5$. The solid (dashed) lines show results from the CVN (CDR) for signal CC ν_μ and CC $\bar{\nu}_\mu$ events in black and NC background interactions in red. The cyan shaded region shows the oscillated flux to illustrate the most important regions of the energy distribution.

The CVN results above are used to calculate the CP -violation sensitivity in DUNE. Figure 3.10 shows the significance with which CP -violation can be discovered as a function of the true value of δ_{CP} for seven and ten years of data collection, assuming the normal mass hierarchy (3.10a) and inverted hierarchy (3.10b). The upper dashed line is drawn at the 5σ confidence level, the level of significance required for big discoveries to be accepted

¹¹The purity of the signal is here defined as the number of correctly identified CC ν_e events divided by the total number of events classified as CC ν_e .

in particle physics. DUNE will be able to measure CP -violation at over 5σ for a narrow (broad) range of δ_{CP} values for the normal (inverted) hierarchy after seven years, and this will widen to broad ranges in both hierarchies after ten years.

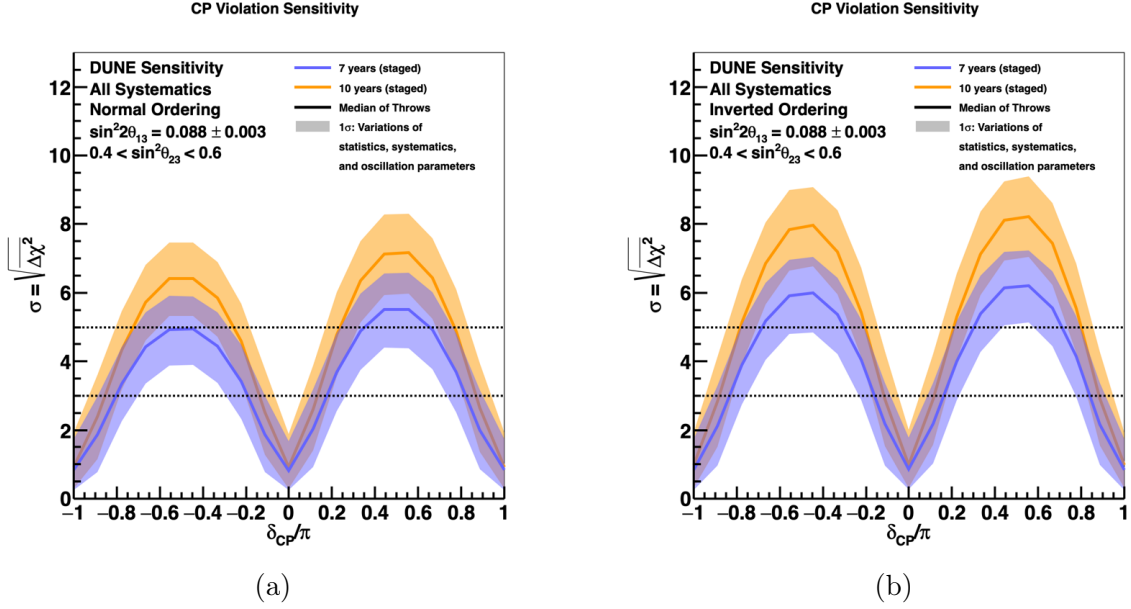


Fig. 3.10 Significance of the DUNE determination of CP -violation ($\delta_{CP} \neq [0, \pm\pi]$) as a function of the true value of δ_{CP} , for seven (blue) and ten (orange) years of exposure, in both normal (a) and inverted (b) ordering. The width of the transparent bands cover 68% of fits in which random throws are used to simulate statistical variations and select true values of the oscillation and systematic uncertainty parameters, constrained by pre-fit uncertainties. The solid lines show the median sensitivity.

3.3 Exclusive final state results

The CVN has three outputs that count the number of final-state particles for the following species: protons, charged pions, and neutral pions. Neutrino interactions with different final-state particles can have different energy resolutions and systematic uncertainties depending on the complexity and particle multiplicity of the interaction. It may be possible to improve the oscillation sensitivity of the analysis by identifying subsamples of events with specific interaction topologies and very good energy resolution.

The individual output scores from the CVN can be multiplied together to give compound scores for exclusive selections. For example, the left plot in Fig. 3.11 shows the combined score for an event to be CC ν_μ with only a single proton in the final-state

hadronic system, formed by the product

$$P(\text{CC}\nu_\mu 1\text{p}) = P(\nu_\mu) P(1\text{p}) P(0\pi^\pm) P(0\pi^0). \quad (3.4)$$

Similarly, the right plot of Fig. 3.11 shows NC1 π^0 score, which contains only a single visible π^0 meson in the final state, defined as:

$$P(\text{NC } 1\pi^0) = P(\text{NC}) P(0\text{p}) P(0\pi^\pm) P(1\pi^0). \quad (3.5)$$

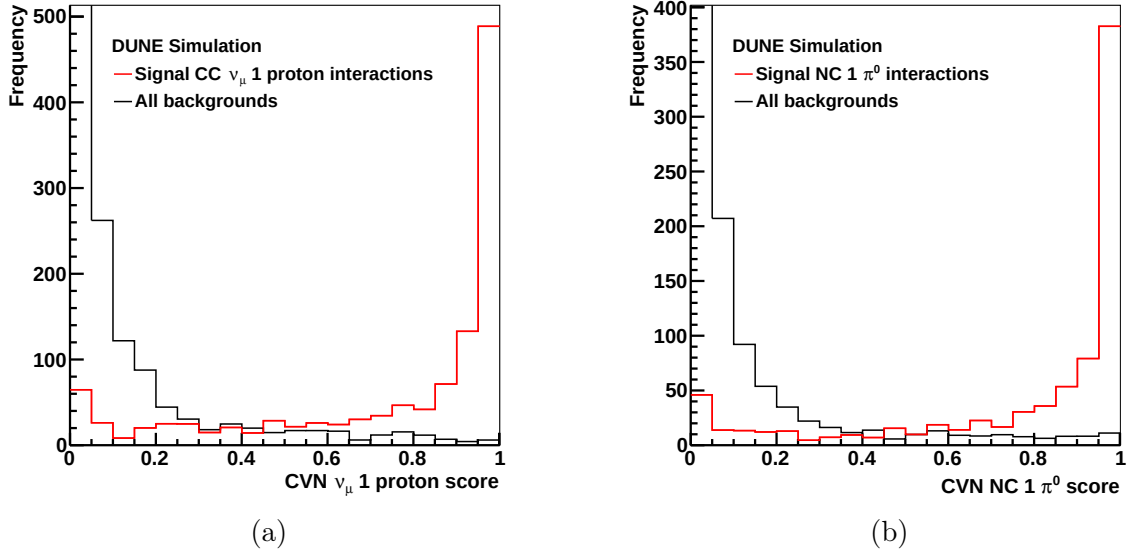


Fig. 3.11 The CC ν_μ 1 proton (a) and NC $1\pi^0$ (b) combined score distributions from the CVN. In both cases the number of other particles is required to be zero. All events that do not fit the signal description comprise the all backgrounds histograms. The histograms are shown in the expected relative fractions but the overall scale is arbitrary.

The background and signal distributions, closely peaked toward 0 and 1 respectively, demonstrate that the efficient selection of exclusive final states will be possible with the DUNE CVN technique. However, it is possible that the CVN is keying in on features of the simulation that are not well-supported by data (e.g., kinematic distributions of particles in the hadronic shower) rather than well-supported features, like the individual particle energy deposition patterns. Studies of potential bias from selections based on these classifiers are required before they can be used to generate analysis samples. Provided that the particle counting outputs can be shown to work in a robust manner for simulations and experimental data, these detailed selections have the potential to significantly improve the scientific output of DUNE FD data.

3.4 Robustness

A common concern about the applications of deep learning in HEP is the difference in performance between data and simulation. A straightforward check of the CVN robustness is to inspect plots of the CVN efficiency as a function of various kinematic quantities. More advanced studies could be imagined where the underlying input physics model is changed to produce alternate input samples for training and testing purposes. Studies of this nature are beyond the scope of this research, but should be part of the validation scheme for any deep-learning discriminant used in eventual analyses of DUNE data.

3.4.1 Charged lepton studies

- *The results presented in this section were produced by Prof. D. Cherdack to check the robustness of the CVN.*

To be considered well-behaved, the CVN flavour identification should be sensitive to the presence of a visible charged lepton and not highly dependent on the details of the hadronic system, which could be poorly modelled. A visible charged lepton requires that the track or shower that it produces has clearly distinguishable features that are not masked by the presence of many overlapping energy depositions from particles from the hadronic shower. Furthermore, background interactions selected by the CVN should be those containing charged pions (for CC ν_μ) or neutral pions (for CC ν_e) that mimic the charged leptons in the signal interactions.

Figure 3.12 shows the variation of the signal selection efficiency as a function of the charged lepton energy for three ranges of hadronic energy for the CC ν_e (left) and CC ν_μ (right) selections. There is a threshold around 0.1 GeV below which no events are correctly identified, and a region at higher lepton energy where the efficiency reaches a maximum and remains relatively flat. As the hadronic energy increases the maximum efficiency decreases, and this effect is more pronounced for the CC ν_e selection since electron-induced EM showers are more easily masked by hadronic shower energy depositions, as compared with long, straight muon tracks. The CC ν_μ efficiency as a function of the true muon energy also demonstrates that the performance is not affected by the lack of confinement of higher energy muons ($\gtrsim 1$ GeV) within the 500×500 pixel images, as was discussed in Sec. 3.1.1.

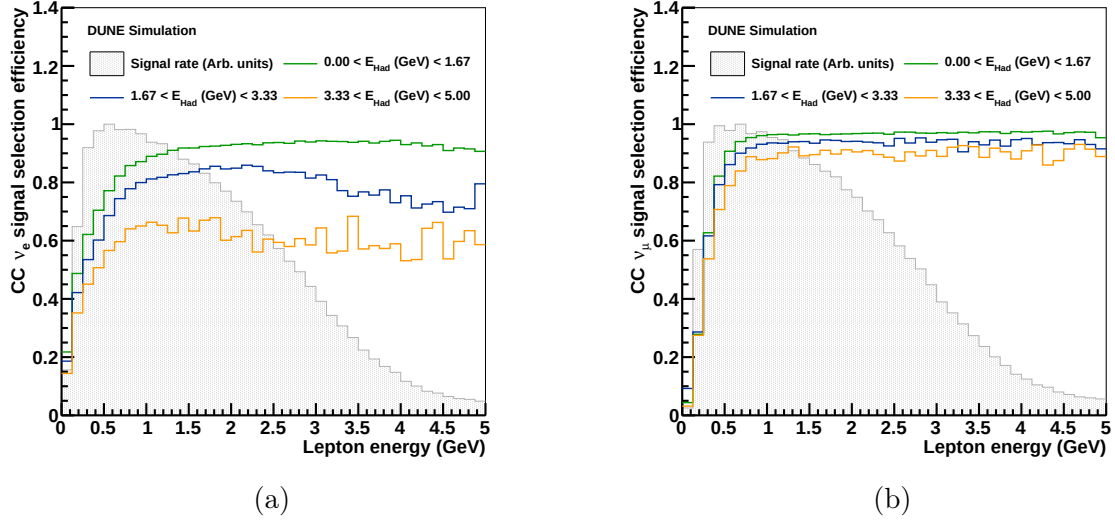


Fig. 3.12 The variation of the signal selection efficiency as a function of the charged lepton energy shown for three ranges of hadronic energy, E_{Had} , for the CC ν_e (a) and CC ν_μ (b) selections. In both cases the distribution of the unoscillated signal events before selection is shown by the filled grey histogram.

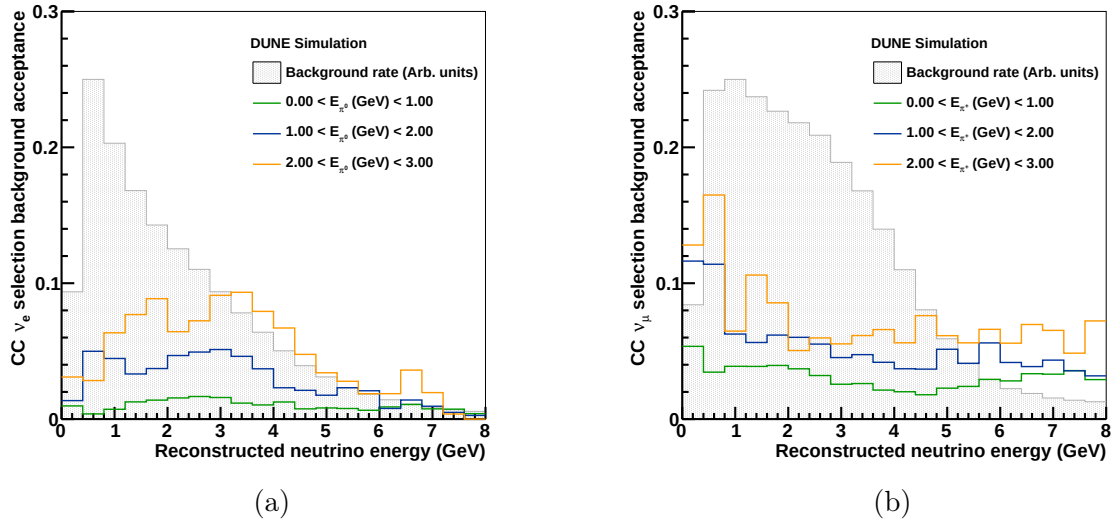


Fig. 3.13 a) the background acceptance in the CC ν_e selection for background interactions containing a final-state π^0 meson; b) the background acceptance in the CC ν_μ selection for background interactions containing a final-state π^+ meson. In both cases the energy distribution of the unoscillated backgrounds before selection is shown by the filled grey histogram.

The plot on the left of Fig. 3.13 shows the efficiency in the CC ν_e selection for background interactions containing a π^0 meson as a function of the reconstructed ν_e energy distribution for three ranges of π^0 energy, E_{π^0} . As expected, the selection efficiency is larger for the background interactions with higher energy π^0 mesons. Similarly, the selection efficiency for background interactions containing a π^+ meson in the CC ν_μ selection is shown on the right of Fig. 3.13 to be larger for higher energy mesons.

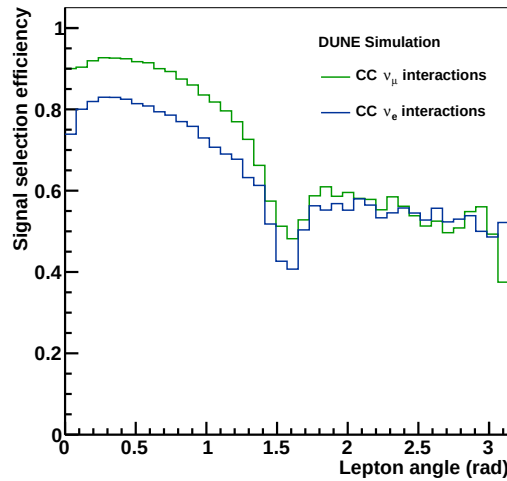


Fig. 3.14 The signal selection efficiency for CC ν_μ (green) and CC ν_e (blue) interactions as a function of the angle between the outgoing charged lepton and the parent neutrino.

Figure 3.14 shows the selection efficiency for CC ν_μ and CC ν_e interactions as a function of the charged lepton angle, defined with respect to the neutrino direction. This angle is defined in 3D, hence when the angle is 90° it corresponds to two cases where the efficiency is expected to be lower: the lepton is travelling almost perpendicular to the readout planes, or the lepton is travelling parallel to the collection plane (view 2) wires. In these two cases the CVN does not have clear images of the charged lepton in one or more readout views. This angle is also strongly correlated with the charged lepton energy, explaining the lower efficiency for events containing backward going, and hence lower energy, charged leptons.

Additional studies, not shown here, help to elucidate other features of these distributions. For example, a small fraction of events with very low energy leptons are still correctly identified. For these events it can be shown that they contain high energy pions which are likely responsible for their strong CVN flavour identification scores. Also of note are studies of the efficiency for other kinematic variables that showed no dependence other than those induced by their correlations with the leptonic and hadronic system energies. Finally, studies of CC ν_τ events showed that efficiencies were consistent with

the tau decay rates to muons and electrons. Roughly 17% of CC ν_τ events were classified as CC ν_μ , and about 17% as CC ν_e . The primary $\tau^\pm$ decays before leaving a track in the detectors, and though CC ν_τ event kinematics are different from CC ν_e and CC ν_μ events, these events are classified based on the visible charged lepton in the event.

The outcome of these studies provides confidence that the CVN classification is strongly tied to the charged lepton features: EM showers and muon tracks. The lowest performance is seen for indistinguishable intrinsic backgrounds, such as beam-induced electron neutrinos, and events with a misidentified hadron and no visible, lepton-induced track or shower.

3.4.2 Occlusion tests

An alternative way to prove the robustness of the network is to occlude some small parts of the input images and check how the CVN reacts to the changes. Small portions of the input images are withheld by setting the pixel values to zero and inference is performed on the modified images to check whether the new scores differ from the original ones: if the scores remain the same, or at least very close, it means the neural network is robust against small image variations. In order to have an overall view of the results, each score difference is placed into a separate map at the pixel corresponding to the centre of the patch; this procedure is repeated across the entire input image (see Fig. 3.15). Since the CVN has three different inputs, we only hide information of the collection plane view (third input) in our tests for simplicity. The occlusion tests presented in this section are inspired by similar work from NOvA [Howard Jr, 2019].

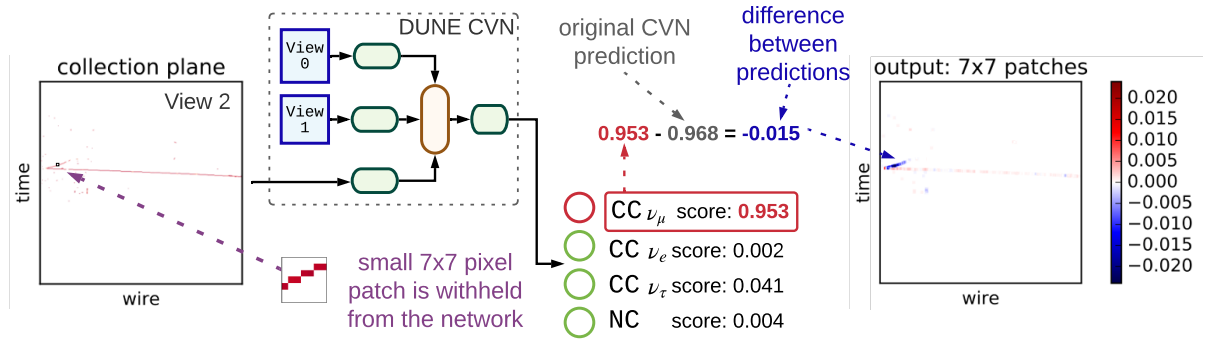


Fig. 3.15 Occlusion test generation for a CC ν_μ event. A small 7x7 pixel patch from the collection view is withheld from the network. The difference between the new and the original CVN predictions is calculated and placed into a separate map at the pixel corresponding to the centre of the patch. This procedure is repeated across the entire image with a stride of 1 until filling the output map with all score differences.

We ran the occlusion tests random sub-sample of 100 events from the dataset used for training; we used patches of size 7x7 pixels since it is the size of the filters in the first convolutional layer of each branch of the CVN. We also used patches of size 5x5 to check how the CVN behaves with filters of smaller size. Again, only the first CVN output was used (flavour of the neutrino interaction) since it is the only one used in the oscillation sensitivity analysis presented in Refs [Abi et al., 2020a] and [Abi et al., 2020d]. Figs. 3.16 to 3.20 show some selected examples of the occlusion tests performed, while Tab. 3.1 summarises the CVN scores for each specific test.

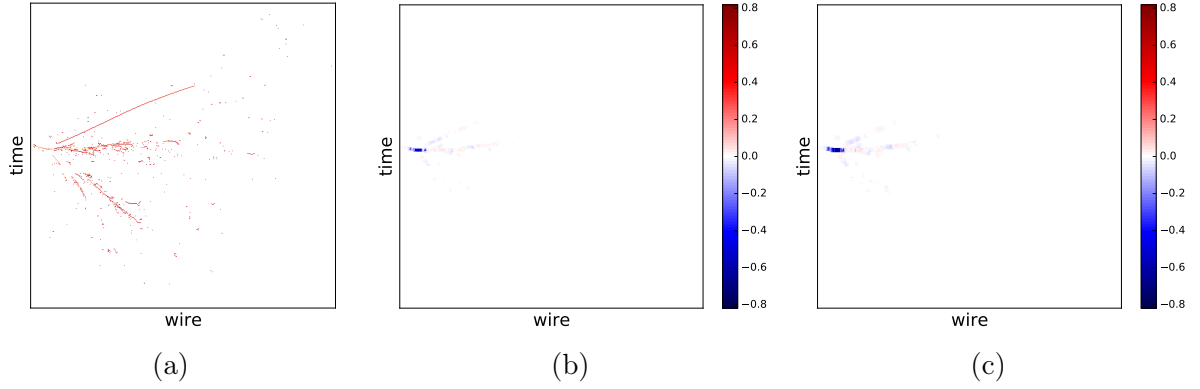


Fig. 3.16 Example of an occlusion test for a CC ν_e event. a) input: collection plane view b) output map after occluding patches of size 5x5 pixels; c) output map after occluding patches of size 7x7 pixels. The colour scale is the change in the CC ν_e score. When the network loses the hints in the patches near the vertex, the event looks much less CC ν_e like as there is a gap before the shower.

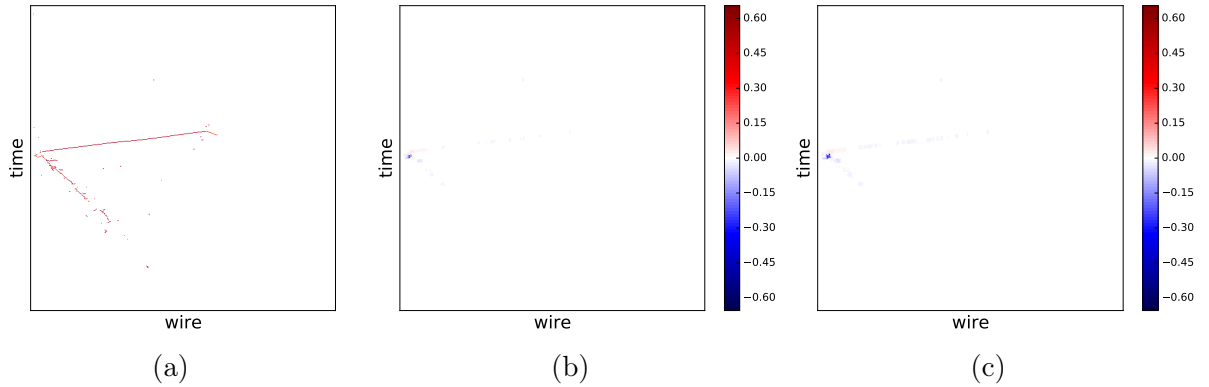


Fig. 3.17 Example of an occlusion test for a CC ν_e event. a) input: collection plane view b) output map after occluding patches of size 5x5 pixels; c) output map after occluding patches of size 7x7 pixels. The colour scale is the change in the CC ν_e score. Removing the start of the electron shower reduces the CC ν_e score, as expected.

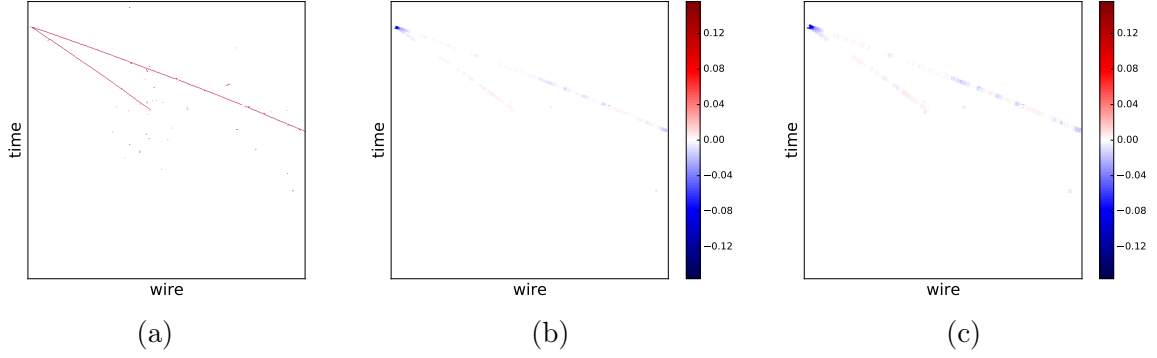


Fig. 3.18 Example of an occlusion test for a CC ν_μ event. a) input: collection plane view b) output map (5x5 pixel patches); c) output map (7x7 pixel patches). The colour scale is the change in the CC ν_μ score. A simple CC ν_μ event shows most degradation when excluding the vertex.

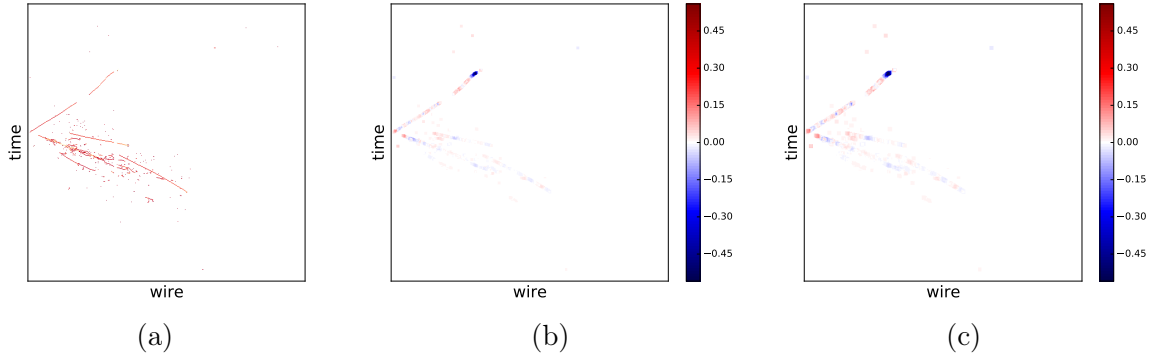


Fig. 3.19 Example of an occlusion test for a CC ν_μ event. a) input: collection plane view b) output map (5x5 pixel patches); c) output map (7x7 pixel patches). The colour scale is the change in the CC ν_μ score. The CVN finds the vertex a bit ambiguous, but it is using the end point of the muon to gain a handle on the event type.

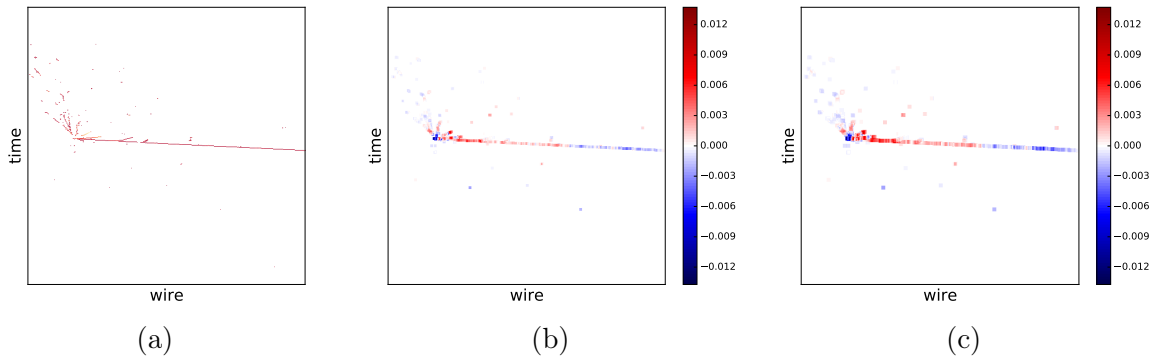


Fig. 3.20 Example of an occlusion test for a CC ν_μ event. a) input: collection plane view b) output map (5x5 pixel patches); c) output map (7x7 pixel patches). The colour scale is the change in the CC ν_μ score. This event has small changes, but we wanted to highlight an interesting feature: part of the muon track has a number of delta rays and diffuse activity close by; excluding these regions very slightly improves the classification since it is less ambiguously a track.

	Actual flavour	Subfigure	CVN scores			
			CC ν_μ	CC ν_e	CC ν_τ	NC
Fig. 3.16	CC ν_e	3.16a	0.0009	0.9184	0.0090	0.0717
		3.16b (largest difference)	0.0015	0.1003	0.0098	0.8884
		3.16c (largest difference)	0.0028	0.1872	0.0128	0.7972
Fig. 3.17	CC ν_e	3.17a	0.0007	0.9560	0.0185	0.0248
		3.17b (largest difference)	0.0026	0.3013	0.0234	0.6727
		3.17c (largest difference)	0.0027	0.4975	0.0358	0.4640
Fig. 3.18	CC ν_μ	3.18a	0.9672	0.0002	0.0258	0.0068
		3.18b (largest difference)	0.8112	0.0002	0.0953	0.0933
		3.18c (largest difference)	0.8112	0.0002	0.0953	0.0933
Fig. 3.19	CC ν_μ	3.19a	0.7142	0.0007	0.0750	0.2101
		3.19b (largest difference)	0.1551	0.0011	0.1552	0.6886
		3.19c (largest difference)	0.1854	0.0011	0.1550	0.6585
Fig. 3.20	CC ν_μ	3.20a	0.9614	0.0002	0.0372	0.0012
		3.20b (largest difference)	0.9477	0.0001	0.0511	0.0011
		3.20c (largest difference)	0.9478	0.0002	0.0510	0.0010

Table 3.1 Actual flavour and CVN scores for input events and output maps from occlusion tests shown in Figs. 3.16 to 3.20.

As can be seen in Tab. 3.1, occluding some relevant information (e.g., vertex) from the events shown in Figs. 3.16, 3.17 and 3.19 can make the CVN output a dramatically different score, even making the predicted flavour differ from the original prediction. However, the above events were manually selected and can be considered outliers since the largest score difference is usually close to zero for most of the events, as shown in Fig. 3.21.

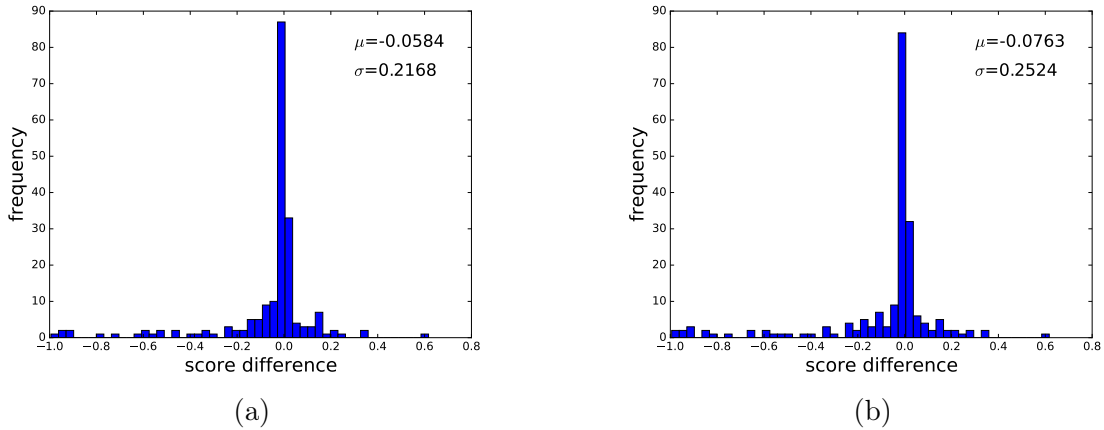


Fig. 3.21 Distribution of largest change in the true classification score of the event after running the occlusion tests. a) Patches of 5x5 pixels; b) patches of 7x7 pixels. Both distributions are peaked at zero, meaning that in the majority of cases, the CVN is robust against small changes to the input images. The distributions are also slightly skewed to the left, showing that removing small patches from the images usually damages the classification performance rather than improving it, as expected.

3.5 Summary

The DUNE CVN algorithm provides excellent neutrino flavour classification, reaching efficiencies of 90% for electron neutrinos and 95% for muon neutrinos, compared to the previously implemented CNN, a GoogLeNet-based architecture that gave a peak efficiency of $\sim 73\%$ and $\sim 82\%$ for electron neutrinos and muon neutrinos, respectively. These efficiencies have basic features that are consistent with those presented in the DUNE CDR [Acciarri et al., 2015]. The CVN outperforms the CDR estimates, exceeding the signal selection efficiency over most of the energy ranges shown, albeit with slightly decreased background rejection capability. The results presented here form a key part of the neutrino oscillation analysis sensitivities presented in the DUNE neutrino oscillation sensitivity analysis [Abi et al., 2020a,d]. A proof-of-principle demonstration of final-state particle counting showed a potential mechanism by which to subdivide the event selections to further improve the analysis sensitivity. The charged lepton and occlusion studies presented in this chapter give promising insights about the robustness of the algorithm.

Chapter 4

Hit-level identification using graph neural networks

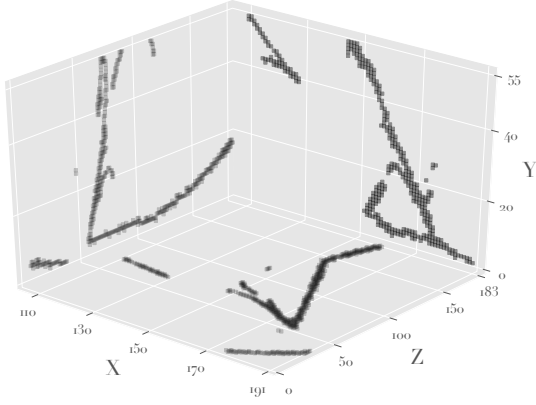
Chapter 3 demonstrated the power of deep-learning in classifying entire neutrino interactions. In this chapter, we consider the other extreme case: the classification of the individual hits that form the events. To do so, a graph neural network inspired by the GraphSAGE algorithm [Hamilton et al., 2017] is used to classify individual voxels in SuperFGD events.

4.1 Motivation

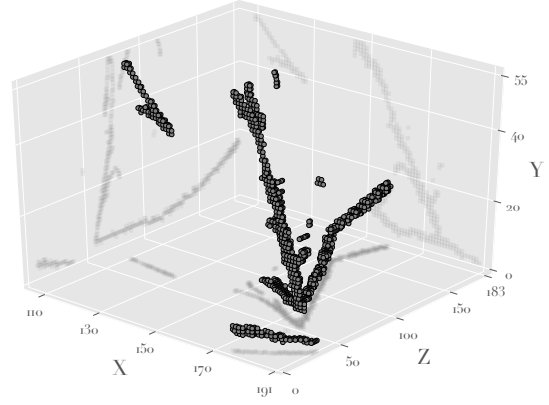
Neutrino interactions in the SuperFGD are imaged on three orthogonal readout views, as shown in Fig. 4.1a. The first step in the event reconstruction involves matching these hits in 2D into 3D voxels that represent the detector scintillator cubes (Fig. 4.1b). To accurately reconstruct neutrino interactions in scintillator-based neutrino detectors, it is crucial to be able to classify each voxel as one of the three types (see Sec. B.4.1 from Appendix B for more details):

- **Track:** a voxel whose energy deposit comes, partially or totally, from scintillation light generated in that same cube.
- **Crosstalk:** a voxel whose energy deposit comes exclusively from light-leakage from neighbouring cubes.
- **Ghost:** a voxel with no physical energy deposit with an apparent signal arising from ambiguities when matching the three 2D views into 3D.

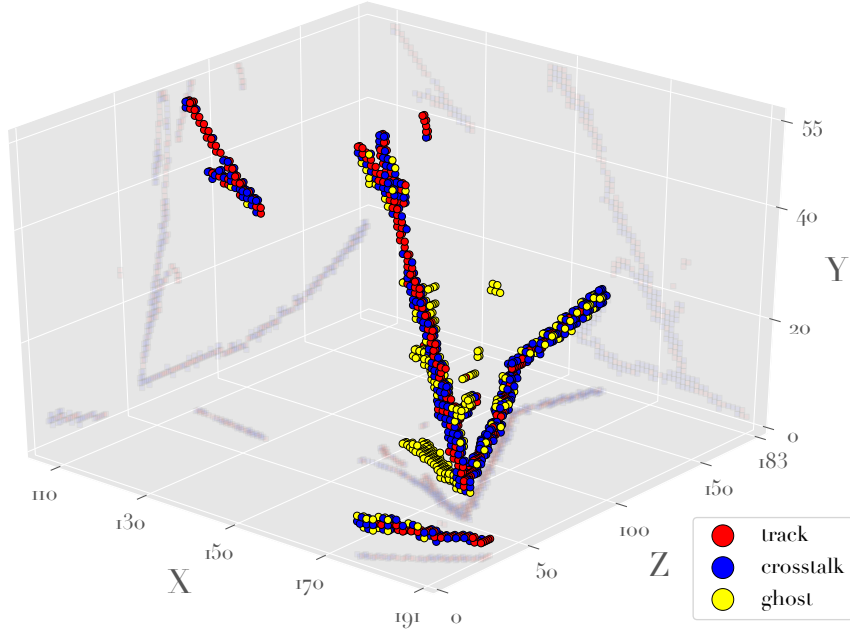
Figure 4.1c shows the three types of voxels using truth information after 3D matching has been performed for an example neutrino interaction.



(a) Projections of the observed neutrino interaction onto the three 2D detector views (XY, XZ, and YZ).



(b) 3D view of the neutrino interaction after the 3D matching of the three 2D views in Fig. 4.1a. The 3D voxels are shown as dark points. Projections of the observed neutrino interaction onto the three 2D detector views (XY, XZ, and YZ) are shown as shadow.



(c) 3D view of the neutrino interaction after the 3D matching of the three 2D views in Fig. 4.1a. The 3D voxels are labelled as track (red), crosstalk (blue), and ghost (yellow) according to the truth information from the simulation. Projections of the observed neutrino interaction onto the three 2D detector views (XY, XZ, and YZ) are shown as shadows.

Fig. 4.1 Visualisation of a neutrino interaction in a finely segmented 3D scintillator detector, demonstrating the relationship between the observed 2D projections onto the three orthogonal 2D views (Fig. 4.1a), the reconstructed 3D voxels (Fig. 4.1b), and the true classification of the voxels (Fig. 4.1c). The energy of the incoming neutrino is 4.754 GeV. The axes are in cm.

Once these voxels are properly labelled (either by the truth information or by a classification algorithm), the ghost voxels can be removed before the full event reconstruction proceeds, while simultaneously cleaning the particle tracks of crosstalk.

In this chapter, we represent the voxels as nodes in a graph and classify the signals using a deep-learning technique based on a graph neural network (GNN, see Sec. A.3.5 from Appendix A for more details). The abstract data representation provided by graphs makes this method very versatile and applicable to any experiment where the output data from the detector elements can be represented as a list of features with arbitrary dimensionality.

In the case study presented here, focused on the SuperFGD detector of the T2K experiment, this method shows great potential to assist reconstruction by assigning a probability to each voxel as being track, crosstalk or ghost. Detector response simulations show that about half of the reconstructed voxels in the SuperFGD will not be of track type. For neutrino physics studies, both in terms of cross-sections and oscillation measurements, correct event topology identification and kinematic reconstruction of outgoing tracks are paramount. Ghost and crosstalk voxels, if not dealt with, will smear the track range estimations, and hence the reconstructed momentum and angle. Similarly, more densely populated events in term of voxels will merge highly colinear tracks and will make it more difficult to identify short tracks that are key for correct topology assignment. The method presented here is therefore expected to benefit future physics measurements in T2K and in any other experiments with similar conceptual challenges as the ones here described.

4.2 Methodology

4.2.1 Data sample generation

The neutrino interactions for the training, validation and testing datasets were produced using the steps below. For each neutrino interaction:

1. Initial particle types and initial kinematics are specified for all final-state particles produced in the interaction.
2. Initial particles are propagated through the detector geometry producing further particles and leaving signals in the form of energy deposits.
3. Using particle energy deposits, the detector response is simulated.
4. The information is stored as a list of voxels with a unique integer known true label: track, crosstalk or ghost.

Initial particle types and kinematics

The initial particle types and their associated kinematics were simulated following two approaches. Firstly, GENIE datasets were created using **GENIE-G18.10b** neutrino interaction software [Andreopoulos et al., 2010]. For a given neutrino flux¹ and target geometry specification, it generates a list of realistic neutrino event interactions both in the number and type of outgoing particles, often referred to as event topologies, and in their individual initial kinematics. Secondly, particle-gun (Pgun) (1-track) and particle-bomb (P-Bomb) (multi-track) datasets were constructed to be as complementary as possible to the GENIE datasets by minimising the modelling dependency. The number of initial particles and their types were fixed in each of these datasets and their input kinematics were chosen to be randomly and uniformly distributed in the range 10-1000 MeV/c. As is standard procedure in machine learning, the datasets are divided into training, validation and testing samples in order to properly assess the validity of the model. A summary regarding the number of events and voxels in the two datasets, as well as of the class distribution is presented in Tab. 4.1.

GENIE dataset		Training	Validation	Testing
	# Events	6k	2k	11.5k
	# Voxels	1.83M	606.7k	3.58M
		Track	Crosstalk	Ghost
	Fraction	43%	37%	20%
P-Bomb dataset		Training	Validation	Testing
	# Events	6k	2k	39.5k
	# Voxels	1.84M	618k	12.3M
		Track	Crosstalk	Ghost
	Fraction	49%	38%	13%

Table 4.1 Descriptions of both GENIE and P-Bomb datasets, displaying the number of events and number of voxels used for training, validating and testing the models. Additionally the fractions of the different classes of voxels are shown, which are conserved through the training, validating, and testing sets.

Particle propagation simulation in the detector

The SuperFGD detector geometry was simulated as described in Ref. [Abe et al., 2019]. The particle propagation and physics simulation is done by means of **GEANT v4-10.6.1** [Agostinelli et al., 2003]. GEANT is a Monte Carlo based toolkit that provides realistic propagation of particles through matter. It outputs a list of energy deposits, with much finer spatial granularity than the detector scintillator cubes.

¹We used the T2K flux, which peaks at 600 MeV, see Ref. [Abe et al., 2013].

All energy deposits² occurring in the same detector cube, including the effect of Birks' quenching [Birks, 1951], are summed to form the list of *track voxels*. To simulate imperfect cube light-tightness, the 3D voxelised energy is then shared with the neighbouring cubes, creating a new set of voxels that originally had no energy deposits, the *crosstalk voxels* (see Fig. B.6). For the energy sharing, a fraction of the energy in the original cube is leaked into each of its six neighbours. The fraction that is shared is sampled from a Poisson distribution, with $\mu = 2.7\%$. Given that the probability for the energy to leak to the neighbour of a neighbouring cube is $O(\mu^2)$, only leakage to immediate neighbours is considered. The 3D voxelised energy of both track and crosstalk voxels is projected onto its three orthogonal planes where the detector 2D signals are simulated, converting the continuous energy deposit into discretised photons³, weighted by distance-dependent attenuation factors, which are detected with 35% probability. To mimic a minimum threshold detection sensitivity, only 2D hits with three or more detected photons are kept. SuperFGD thresholds at this level are expected to get rid of virtually all noise hits [Blondel et al., 2020], hence we have not included noise hits in our simulation. Then, the 2D hits are matched into 3D reconstructed voxels only if the same XYZ coordinate combination can be made using two different combinations of 2D planes. In this process, due to ambiguities some extra voxels are created, the *ghost voxels* (see Fig. B.7). Finally, those track and crosstalk voxels not reconstructed after the 3D matching are discarded from the original lists. An example of the 2D to 3D reconstruction is shown in Figs. 4.1a and 4.1b.

Simulation output

The resulting output from the simulation is a list of voxels and their associated energy deposits in the three planes, each with one of the following three labels that we want to classify, as described in Sec. 4.1: track, crosstalk or ghost voxel. Using the list of voxels of each event, further features are computed for each voxel as described in Section C.1 from Appendix C, as well as defining the graph's adjacency matrix utilising the position of each voxel, as will be detailed below. Both the new list of expanded voxel features plus the corresponding adjacency matrix of the event are fed into the GNN algorithm.

²Only signals recorded within 100 ns of the true interaction time are considered. Further delayed signals, such as decays, can be treated as independent graphs.

³No waveform processing is simulated. A single conversion factor is used from energy deposit to number of photons in the WLS fibre, based on laboratory data [Blondel et al., 2020].

4.2.2 Network architecture

GraphSAGE [Hamilton et al., 2017] is a technique that leverages the features of graph nodes $\mathcal{V}$ to generate efficient representations on previously unseen samples by learning aggregator functions from training nodes (see Sec. A.3.5 from Appendix A for more details). Each graph in GraphSAGE is constructed using the proximity of two voxels in that graph. If both voxels are spatially located within a radius of 1.75 cm^4 , then we consider them to be connected in the graph by an edge; we repeat the same procedure for each pair of voxels⁵. Additionally, we consider a neighbourhood depth of three, i.e., to produce the embedding of a voxel, we use the voxel features together with its first neighbours' features, the features of the neighbours of its neighbours, i.e, second neighbours' features, and the features of the neighbours of the neighbours of its neighbours, i.e., third neighbours' features. The aggregator used to combine the feature of the neighbours is the mean aggregator, which produces the average of the neighbours' values. This final embedding is then passed to an MLP consisting of two fully connected layers - each followed by a LeakyReLU activation function, and a final output layer followed by a softmax activation function. Figure 4.2 illustrates the GraphSAGE-based approach used, while Tab. 4.2 shows the architectural parameters chosen. Categorical cross-entropy is chosen as the cost function to minimise during training, as it is considered the standard one for multi-class classification problems, where each training example corresponds to a voxel:

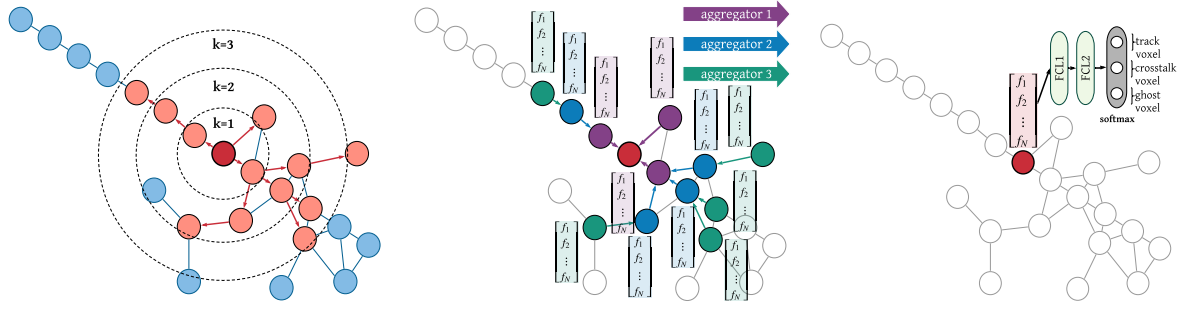
$$J = -\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^c y_j^{(i)} \log \hat{y}_j^{(i)}, \quad (4.1)$$

where:

- $\mathbf{y}^{(k)}$: true values corresponding to the k^{th} training example. $\mathbf{y}^{(k)}$ is a vector with all components equal to zero except for the class j , which is equal to one.
- $\hat{\mathbf{y}}^{(k)}$: predicted values corresponding to the k^{th} training example. $\hat{\mathbf{y}}^{(k)}$ is a vector with each component $\hat{y}_j^{(k)}$ denoting the score (continuous value from 0 to 1) of being of class j .
- m : number of training examples, equal to the total number of voxels in the training sample.
- c : number of classes/neurons corresponding to the output. In this case, the three classes are: track, crosstalk, and ghost.

⁴To link only those voxels within the $3 \times 3 \times 3$ cube of voxels centred on the target voxel (the maximum diagonal distance from the centre of this cube is $\sqrt{1^2 + 1^2 + 1^2} \approx 1.75$).

⁵If a voxel has no neighbours, it is discarded from the graph and cannot be classified; this happens for less than 0.6% of the total number of voxels.



(a) Sample neighbourhood (orange) for a depth $K = 3$. The red node indicates the target node to be classified; orange nodes are nodes taken into account for the classification given a depth of three; blue nodes are not taken into account.

(b) Aggregate feature information from neighbours. Each node has features $f_1, \dots, f_N$, with purple, blue, and green nodes information being aggregated through aggregators 1, 2, and 3, respectively.

(c) Use aggregated information as input for the fully connected layers and predict the label.

Fig. 4.2 Visual illustration of the GraphSAGE sample and aggregate approach with a depth of three [Hamilton et al., 2017].

Parameter	value
Encoding size	128
Depth	3
Aggregator	mean
Fully Connected Layer 1	128 neurons
Fully Connected Layer 2	128 neurons
Fully Connected Layer 3 (output)	3 neurons

Table 4.2 Architectural parameters; for more information about the meaning of the parameters, see Sec. A.3.5 from Appendix A.

The output layer of the model consists of three neurons, one for each of the three classes, with values v_i for $i = 1, 2, 3$. The sum of neuron values is given by $\sum_{i=1}^3 v_i = 1$ such that each neuron value gives a fractional score that can be used to classify voxels. In other words, the model returns scores for each voxel to be one of the three desired outputs, which can be interpreted as the probability: track-like, crosstalk-like, or ghost-like.

4.2.3 Training

The network was trained for 50 epochs using Python 3.6.9 and PyTorch 1.3.0 [Paszke et al., 2019] as the deep-learning framework, on an NVIDIA RTX 2080 Ti GPU. Adam [Kingma and Ba, 2014] is used as the optimiser, with a mini-batch size of 32, and an initial learning rate of 0.001 (divided by 10 when the error plateaus, as suggested in Ref. [He et al.,

2015a)]. The network has a total of 105,347 parameters. Figure 4.3 shows the validation results during the training process, measured by the F_1 -score metric:

$$F_1 = 2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \quad (4.2)$$

The precision and recall are defined as:

$$\text{precision} = \frac{\text{true}_{\text{positives}}}{\text{true}_{\text{positives}} + \text{false}_{\text{positives}}}, \quad (4.3)$$

$$\text{recall} = \frac{\text{true}_{\text{positives}}}{\text{true}_{\text{positives}} + \text{true}_{\text{negatives}}}, \quad (4.4)$$

where the labels are compared as one class vs. all the others. The model used later for inference on new data is the one that maximises the F_1 -score for the validation set, as it has the best generalisation for unseen data.

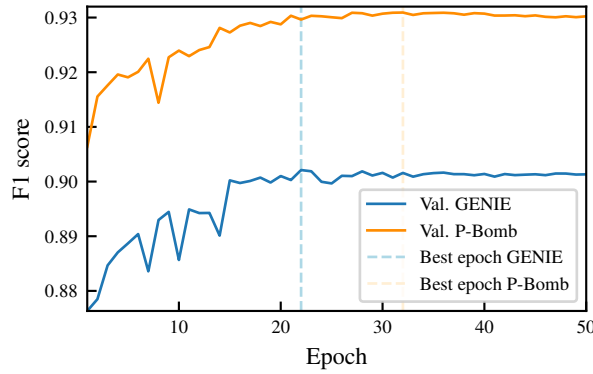


Fig. 4.3 Validation F1 results on GENIE and P-Bomb samples.

4.3 Results

The GNN voxel-type predictions are compared against the true labels to evaluate the network performance. Here, we choose the output class with the highest score as the predicted class of each voxel although, depending on the type of analysis, different selection criteria could be applied in the future. The signal efficiencies and purities of these predictions are calculated by two methods: per voxel and per event, the former are given by the following formulas for each type of voxels:

$$\text{efficiency}_i = \frac{\# \text{ voxels with label}_{\text{true}} = \text{label}_{\text{pred}}=i}{\# \text{ voxels with label}_{\text{true}}=i}, \quad (4.5)$$

$$\text{purity}_i = \frac{\# \text{ voxels with label}_{\text{true}} = \text{label}_{\text{pred}}=i}{\# \text{ voxels with label}_{\text{pred}}=i}. \quad (4.6)$$

The efficiencies and purities per event are defined as the mean of the efficiencies and purities of individual voxels for each event. The results of both methods for four sets of training/testing samples are shown in Tab. 4.3, showing that very similar performance is seen for all combinations of training and testing datasets used to train and test the GNN.

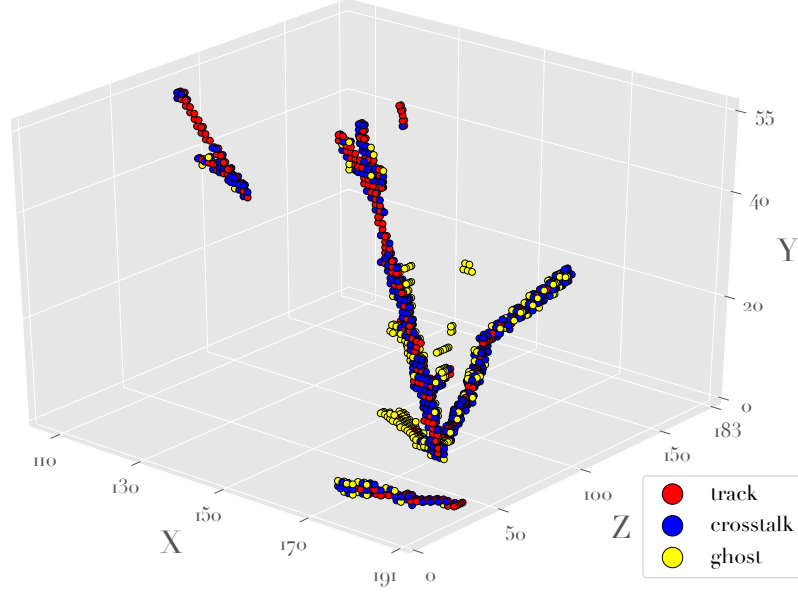
		GENIE Training				P-Bomb Training			
			Track	Crosstalk	Ghost		Track	Crosstalk	Ghost
GENIE Testing	Per Voxel	Efficiency	93%	90%	84%	Efficiency	93%	89%	80%
		Purity	93%	87%	91%	Purity	91%	86%	89%
	Per Event	Efficiency	94%	94%	88%	Efficiency	94%	93%	88%
		Purity	96%	91%	92%	Purity	95%	91%	91%
P-Bomb Testing	Per Voxel	Efficiency	94%	93%	87%	Efficiency	95%	93%	88%
		Purity	95%	90%	92%	Purity	95%	91%	92%
	Per Event	Efficiency	94%	94%	87%	Efficiency	95%	93%	88%
		Purity	96%	90%	92%	Purity	96%	91%	92%

Table 4.3 Mean efficiencies and purities of voxel classification, calculated for the whole sample (per voxel) and as a mean of the event-by-event efficiencies and purities (per event).

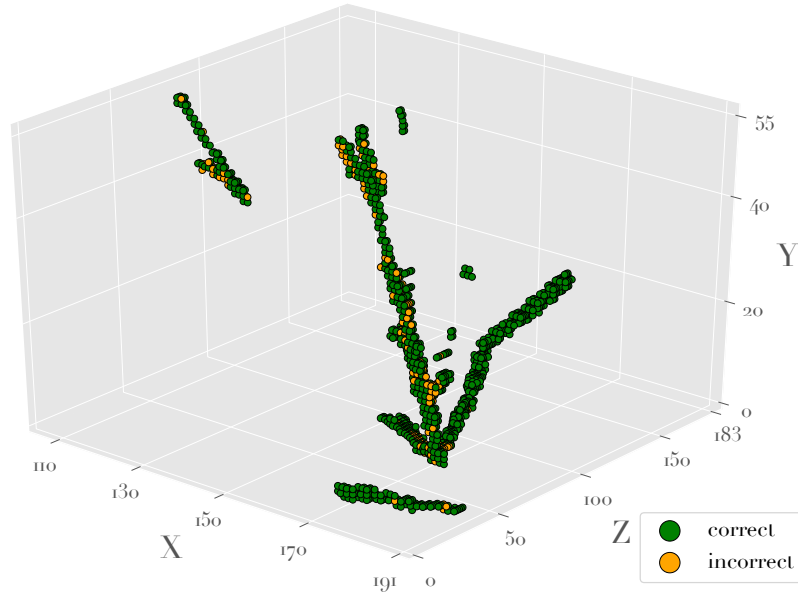
As an example, Fig. 4.4 shows the voxel prediction results from the GNN when applied to the event shown in Fig. 4.1, a GENIE event that features a track almost completely composed of ghost voxels. Figure 4.4a shows the class predicted for each voxel, while Fig. 4.4b displays which voxels were correctly/incorrectly classified.

A more in-depth analysis of the GNN performance can be carried out by studying the effects of different event properties on the efficiencies and purities of the predictions. For these studies, given the demonstrated invariance of the GNN to the choice of training and test samples, the results of the GNN trained and tested on the GENIE dataset are used.

One of the factors expected to affect these predictions is the number of voxels in the event. Figure 4.5 shows the relationship between the mean efficiency and purity per event for each type of voxel as a function of the total number of voxels in the event. The figure also shows the mean number of events in each bin (in light blue). It is clear that both the efficiencies and purities of the three types of voxels decrease as the number of voxels in the event increases. This decrease is coupled with an increase of the fraction of ghost



(a) Prediction: voxels are coloured based on the GNN predictions.

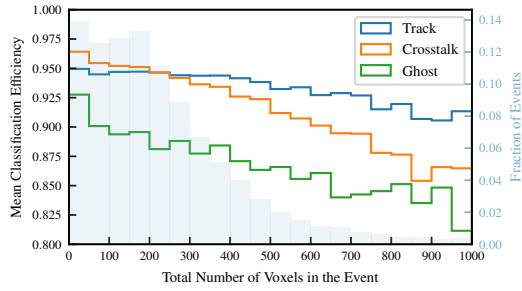


(b) Accuracy: voxels correctly classified by the GNN are shown in green.

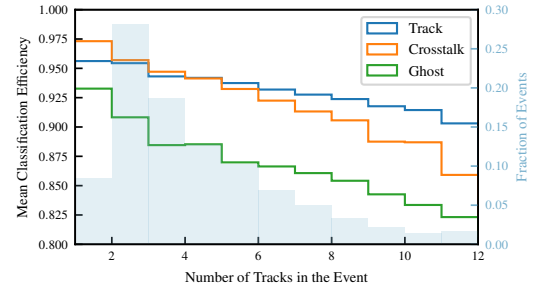
Fig. 4.4 Example GNN prediction results for the interaction shown in Fig. 4.1. In this event, 86.3% of the voxels were correctly identified. The GNN is able to almost entirely reject the fake track travelling on the XZ plane and stopping near to the vertex at $X \sim 160$ cm and $Z \sim 70$ cm.

voxels as the total number of voxels increases, which are the hardest for the GNN to classify.

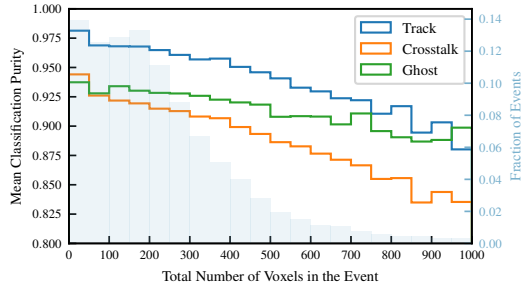
The number of tracks in the event is an estimate of the complexity of its topology. According to Fig. 4.6, the classification efficiencies and purities drop as the number of tracks increases. This behaviour is also correlated with the increasing fraction of ghost voxels in the events. In all cases, the efficiencies and purities remain above 80% even for the most complex events.



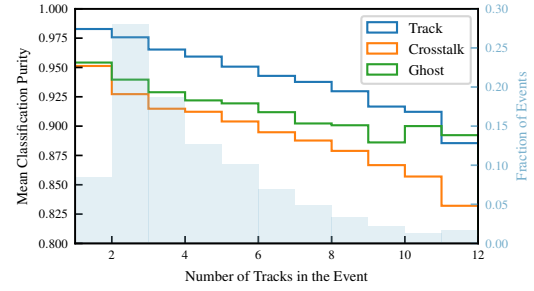
(a) Efficiency.



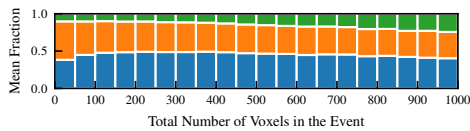
(a) Efficiency.



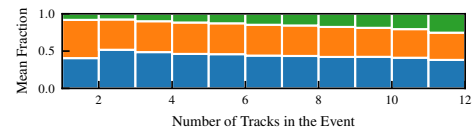
(b) Purity.



(b) Purity.



(c) Mean fraction of each type of voxel as a function of the number of voxels in the event (blue = track, orange = crosstalk, green = ghost).



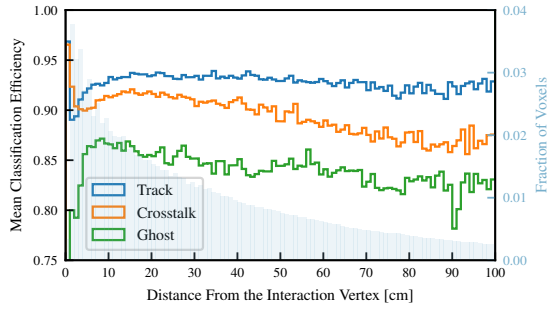
(c) Mean fraction of each type of voxel as a function of the number of tracks in the event (blue = track, orange = crosstalk, green = ghost).

Fig. 4.5 Efficiency and purity as a function of the number of voxels in the event for a sample trained and tested on GENIE simulated data.

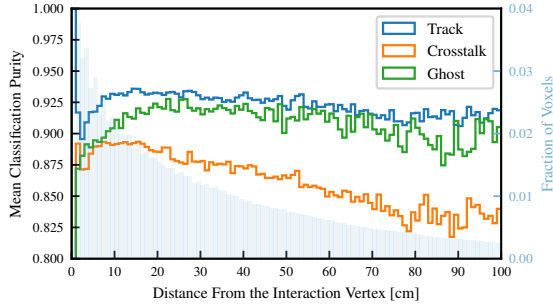
Fig. 4.6 Efficiency and purity as a function of the number of tracks in the event for a sample trained and tested on GENIE simulated data.

The region around the interaction vertex is of particular interest in the event. It is expected that a high spatial density of voxels within a certain volume of the detector may pose a challenge for the GNN to correctly identify the voxel type. This can be observed

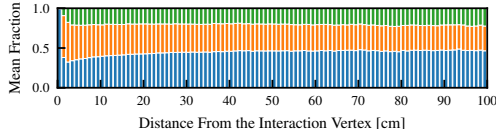
by studying the efficiencies and purities as a function of the distance to the interaction vertex, as shown in Fig. 4.8. At the interaction vertex itself, it is clear that there are only track voxels and the GNN can identify them with over 96% efficiency and 100% purity. The following 2 cm exhibit only a small fraction of ghost voxels, mainly due to the high spatial density of voxels with real signals in that volume, which is mainly occupied by track and crosstalk voxels. As we go further from the vertex, the efficiencies and purities increase up to 10 cm, after which we expect the effect of vertex-proximity on the purities and efficiencies to subside, and the effects of the number of voxels and tracks to be more relevant.



(a) Efficiency.



(b) Purity.



(c) Mean fraction of each type of voxel as a function of the distance to the vertex (blue = track, orange = crosstalk, green = ghost).

Fig. 4.8 Efficiency and purity as a function of the distance to the neutrino interaction vertex for a sample trained and tested on GENIE data.

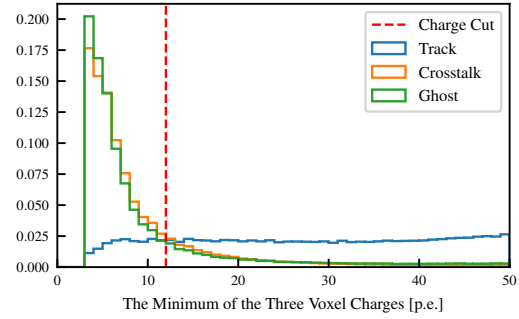


Fig. 4.9 The distribution of the minimum charge among the three voxel charges for the GENIE sample.

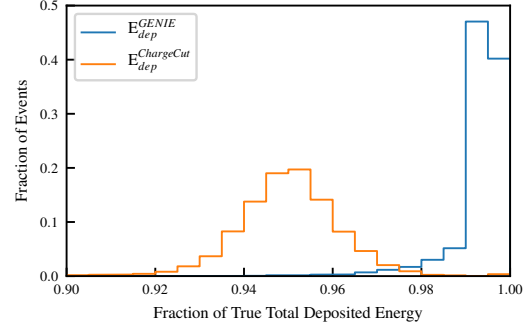


Fig. 4.10 The fraction of the true total deposited energy obtained when using the GNN (trained on GENIE) or the charge cut as a classification method.

As the main goal of this GNN is to identify ghost voxels in order to eliminate them from the events, it is important to make sure that true track and crosstalk voxels are not lost in the process. According to the GENIE sample results, only 1.1% of all true track voxels and 3.3% of crosstalk voxels are incorrectly classified as ghost voxels by the GNN. In addition, it is important not to miss ghost voxels: the GNN correctly identified 84.5% of all ghost voxels, where 72.1% of those classified incorrectly were predicted as crosstalk. Therefore, although not ideal, this issue is not critical as crosstalk voxels have a smaller influence on future studies than track voxels.

Lastly, we compare the results of the GNN against a conventional method of voxel classification which relies on a charge cut. As described in Section C.1 from Appendix C, each voxel has three charges that correspond to the signals from the three fibres passing through it. Since other voxels along the same fibre may have signals causing a larger amplitude to be recorded, we consider the smallest of these three charges to be the most accurate estimation of the true voxel charge. Hence, this minimum charge is used for the purposes of this charge cut. Since, by definition, we expect higher energy deposition in track voxels compared to crosstalk and ghost voxels, we set a lower limit for the minimum charge in a voxel such that any voxels with a higher minimum charge than the threshold are classified as track voxels. Fig. 4.9 shows the distribution of the minimum voxel charge for the three types of voxels. From this figure, it is clear that it is not possible to separate ghost from crosstalk voxels using the simple charge cut method. Thus, this classification is only binary such that we have two categories: track or other. We decide to place this cut at 12 p.e., where the track and non-track voxel curves intersect.

To compare the results of this cut with those of our GNN, we combine the predictions of the crosstalk and ghost categories. Table 4.4 shows the efficiency and purity of the classifications for the two methods. It is evident that using only a charge cut can still yield a comparable track voxel classification efficiency to the GNN. However, it struggles to correctly classify non-track voxels which, in turn, reduces the purity of the predicted track voxels.

GNN			Charge Cut		
	Track	Other		Track	Other
Efficiency	94%	96%	Efficiency	93%	80%
Purity	96%	95%	Purity	80%	91%

Table 4.4 Mean efficiencies and purities of voxel classification for the GNN and a simple charge cut.

Another advantage of the GNN over the charge cut is the improved capability of rejecting “fake” tracks, i.e. a cluster of ghost voxels that closely resembles the structure of a real particle track. Since fake tracks are usually produced by the shadowing of real

tracks, the corresponding number of p.e. measured in the three readout views can be higher than 12 p.e., hence the charge cut cannot reject them easily. The superiority of the GNN in reducing ghost tracks is shown in Section C.2 from Appendix C for a number of neutrino interactions in comparison to the charge cut method.

Figure 4.10 shows the advantage of the three-fold classification of the GNN over the binary classification of the charge cut when comparing the fraction of true total deposited energy obtained using each method. In the case of the GNN, the total deposited energy in an event is the sum of the true energy deposited in all non-ghost voxels. For the charge cut, only the energy deposited in track voxels is used. This causes an average energy loss of 5% per event when using a method that also excludes the crosstalk voxels, compared to less than 1% when using the GNN that can isolate ghost voxels.

4.4 Systematic uncertainty considerations

The results presented in Sec. 4.3 show that the GNN is a very powerful technique for removing ghost voxels and identifying optical crosstalk in 3D-reconstructed neutrino interactions. It is important to demonstrate that this technique is robust and does not introduce new systematic uncertainties, potentially given by a sub-optimal choice of the training sample. As described in the previous section, different training samples, (GENIE or P-Bomb) were generated and the results were summarised in Tab. 4.3, demonstrating that the performance is still very good even when the samples used for training and testing were different. The robustness of the GNN against model dependencies can be verified by training different neural networks on different event samples and applying them to the same set of neutrino interactions. A difference in the observables used in the physics measurement, such as particle momenta, energy deposit, etc., obtained by the different training can be assigned as a systematic uncertainty introduced by the method.

A study was performed to evaluate the impact of the method on the total true energy deposited in the detector. The difference between the total energy deposit computed after rejecting the voxels classified as ghosts for both network trainings was computed. Figure 4.11 shows the distribution of the total true deposited energy before and after discarding the voxels classified as ghosts. Both GENIE- and P-Bomb- trained GNNs give very similar results over the full range of total deposited energy. The total true deposited energy computed with and without ghost rejection differ on average by less than 1 MeV with a standard deviation of approximately 5.5 MeV, mainly due to a few outlier entries, and 68% of the events with a difference better than 0.192 MeV, as shown in Fig. 4.12. Hence, it is expected to be improved by increasing the statistics of the training samples.

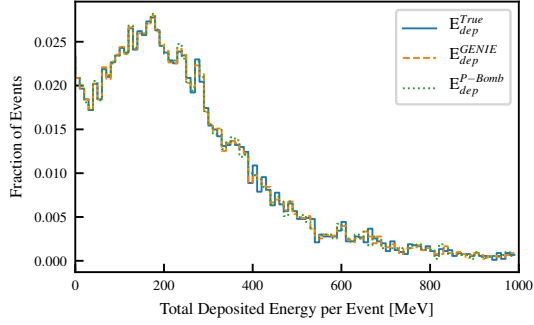


Fig. 4.11 Distribution of the total true deposited energy after rejecting the ghost voxels classified either with GENIE- (dashed orange) or P-Bomb- (dotted green) trained GNNs and without any ghost rejection (solid blue). The mean total deposited energy per event is about 288 MeV.

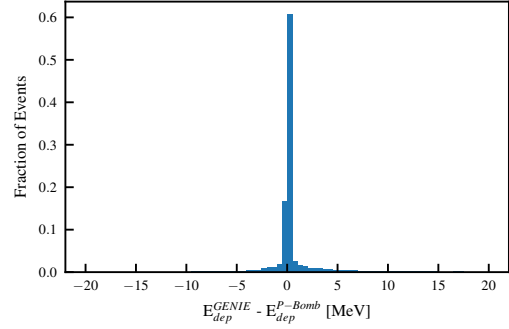


Fig. 4.12 Difference between the total true deposited energy computed after rejecting the ghost voxels classified with GENIE- and P-Bomb- trained GNNs. The mean is 0.78 MeV while the standard deviation is 5.5 MeV. About 40% of events show no difference between P-Bomb and GENIE, 68% have a difference within ± 0.192 MeV, while only 5% of the events have a difference outside the range ± 6.35 MeV.

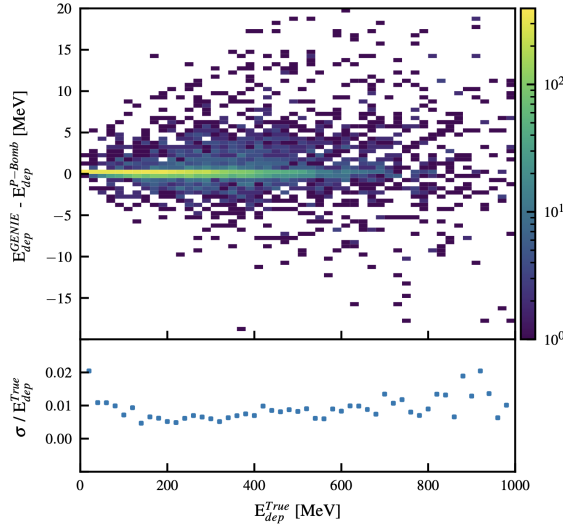


Fig. 4.13 Top: difference between the total true deposited energy computed after rejecting the ghost voxels classified with GENIE- and P-Bomb- trained GNNs as a function of the total true deposited energy. Bottom: fractional standard deviation of the difference of the total true deposited energy computed after rejecting the ghost voxels classified with GENIE- and P-Bomb- trained GNNs as a function of the total true deposited energy.

This corresponds to less than 2% of the mean total deposited energy per event. In Fig. 4.13 the impact of the different training sample is shown as a function of the total deposited energy. The fractional standard deviation, defined as the standard deviation of the difference between deposited energy computed from different GNN trainings and divided by the true deposited energy, shown in the bottom panel, is less than 2% and almost constant as a function of the deposited energy. This means that the performance of the method is about the same irrespective of the total deposited energy. This study confirms that GNN can be used for classifying 3D voxels potentially with limited systematic uncertainties in the deposited energy, while drastically improving the tracking capability.

Nominal		Track	Crosstalk	Ghost
Crosstalk 2.7%	Efficiency	93%	90%	84%
	Purity	92%	87%	91%
Crosstalk 2%		Track	Crosstalk	Ghost
	Efficiency	92%	89%	81%
	Purity	94%	83%	89%
Crosstalk 5%		Track	Crosstalk	Ghost
	Efficiency	94%	89%	88%
	Purity	86%	91%	93%

Table 4.5 Mean efficiencies and purities of voxel classification, per voxel, for different crosstalk values, i.e. 2.7% (nominal) 2% and 5%. The GNN was trained with GENIE training samples with nominal crosstalk and tested on the same GENIE sample with different crosstalk values to study its robustness.

Another potential issue could be given by a mismodelling of the amount of crosstalk. In addition to the nominal optical crosstalk (2.7%), two further datasets were simulated using 2% and 5% crosstalk and the voxel classification was performed using the GNN trained with nominal crosstalk. As shown in Tab. 4.5, the efficiency and the purity is relatively stable even in the case where the crosstalk model is wrong, in particular for identifying track voxels. Whilst a drop in purity for track voxels is observed with 5% crosstalk, such a large mismodelling is highly unlikely given that crosstalk can be measured with small prototypes to sub-percent precision [Blondel et al., 2020]. Hence, crosstalk mismodelling is not considered to be a source of additional systematic uncertainty given that the GNN method is robust to small crosstalk variations.

4.5 Summary

A graph neural network inspired by GraphSAGE was developed and tested on simulated neutrino interactions in a 3D voxelised fine-granularity plastic-scintillator detector with three 2D readout views with the same geometry as SuperFGD, a detector that will be installed in the near detector (ND280) of T2K. The advantage of this neural network is that the graph data structures provide a natural representation of the neutrino interactions.

The neural network was able to identify ambiguities and scintillation light leakage between neighbouring active scintillator detector volumes as well as real signatures left by particles with efficiencies and purities in the range of 94-96% per event, with a clear improvement with respect to less sophisticated methods. In particular, it can reject fake tracks produced by the shadowing of real tracks observed in the 2D readout views. The performance was tested for neutrino events with different number of voxels, number of tracks and voxels at different distances from the vertex, variables that could hint to interaction model dependencies of the method. Efficiencies and purities were found to be relatively stable and the trends were consistent with the expectation. The robustness of the neural network against possible systematic uncertainties introduced by the method was tested. The results were obtained using neural networks trained on different samples, produced either with the GENIE event generator or by randomising the number of final state particles and relative momentum to obtain a more generic sample that does not belong to any particular theoretical model. It was found that the bias introduced on the total deposited energy of the event by arbitrarily choosing a different training sample is, on average, less than 1 MeV, or less than 2% for true deposited energies in the range 0.0-1.0 GeV. The impact of potential mismodelling of the light leakage between neighbouring scintillator volumes was tested. Results show that the performance of the neural network is robust to expected changes in the crosstalk modelling.

Chapter 5

Fast simulation and model parameter optimisation using generative adversarial networks

Accurate simulations are a critical to HEP experiments, but they are typically computationally expensive, hence there is great interest in fast simulations. In this chapter we propose and demonstrate the use of a model-assisted generative adversarial network to efficiently produce fake images that accurately match true images through the variation of the parameters of the model that describes the features of the images. The generator learns the model parameter values that produce fake images that best match the true images.

5.1 Model-assisted generative adversarial networks

Generative adversarial networks (GANs) [Goodfellow et al., 2014] have been shown to produce fake images indistinguishable from true images, but these images are manipulated in an arbitrary way to match the true image (see Sec. A.3.6 from Appendix A for more details). However, there are many experiments, for example in experimental physics, where the features of the true images can be described by a model that contains a number of parameters (hereafter referred to as the model parameters). In these cases, the GAN should only be able to manipulate the images in a way described by the model parameters to produce accurate and physically motivated fake images. Considering a set of true images produced by such an experiment, with fake images produced by an implementation of the model called the simulation, it is clear that an arbitrary approach to manipulating the fake images to match the true images is not well motivated as it

disregards any knowledge of the model and the model parameters. Note that in a real experiment the true images are experimental data images, but here we will henceforth refer to them as true images.

We propose the model-assisted GAN (MAGAN) as a solution to this problem. The approach varies the model parameters $\mathbf{p} = (p_0, \dots, p_N)$ that cause well-defined changes in the images, providing new fake images that better match the true images. A set of true images are produced with true model parameter vectors $\mathbf{p}_t$ drawn from a distribution such that $\mathbf{p}_t \sim p_{data}(\mathbf{p}_t)$, since a single choice of $\mathbf{p}_t$ would produce a set of identical images. The goal is for the MAGAN to generate a set of model parameters $\mathbf{p}_{bm} \sim p_{generator}(\mathbf{p}_{bm})$ to produce simulated images that best match the true images such that $p_{generator}(\mathbf{p}_{bm}) = p_{data}(\mathbf{p}_t)$. The default simulation for an experiment will typically have default model parameter values that do not exactly match the true model parameter values. The parameters $\mathbf{p}_{bm} \sim p_{generator}(\mathbf{p}_{bm})$ can be extracted to update the default simulation model parameters so that the fake images will more accurately reproduce the true images. Furthermore, the difference between $p_{generator}(\mathbf{p}_{bm})$ and the default parameters distribution gives physical insight into the understanding of the model and the model parameter values that were not correct in the default simulation. The key advantage of the MAGAN is that the simulation only needs to be run once, and then any number of model parameter optimisations can be performed with different true data sets very quickly. Another advantage is that any additional number of fake images can be produced efficiently using the emulator from the MAGAN.

5.1.1 Algorithm description

A schematic of the MAGAN architecture is shown in Fig. 5.1. The three data types (true, simulated and emulated), the four neural networks (Generator, Emulator, Discriminator, and Siamese) and the Simulator that form the architecture are described in the following sections. The training and implementation of the network are also discussed.

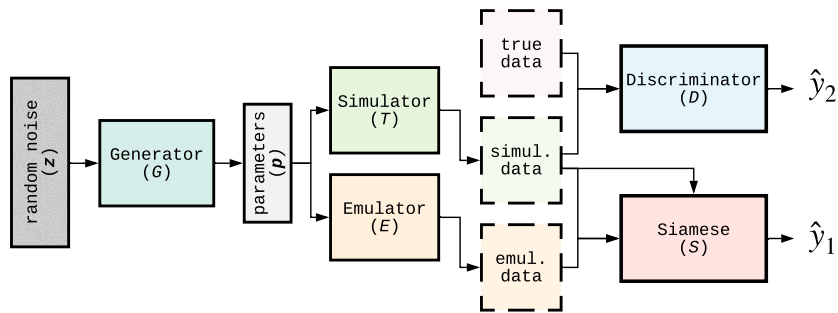


Fig. 5.1 An overview of the model-assisted GAN.

Data types

There are three $m \times m$ pixel image data types:

- *True data*: a set of simulated data with a chosen set of true parameters $\mathbf{p}_t$ used to represent an experiment. In a real experiment this would be the experimental data with unknown parameter values that we want to measure.
- *Simulated data*: the output from the custom simulation used to simulate the experiment.
- *Emulated data*: the output from the emulator that learns to mimic the custom simulation.

Generator

The goal of the generator G is to produce parameters such that the simulator can use them to create images that cannot be distinguished from true data images. Contrary to the generator from a traditional GAN, this generator outputs only the set of parameters $\mathbf{p}$ rather than the completed image. These parameters form the input to both the simulator and the emulator. In our implementation, the last layer of G uses a tanh activation function.

Simulator

The Simulator T , which is specific to each use-case, can be any set of operations that performs a well-defined transformation from the input model parameters $\mathbf{p}$ to an $m \times m$ pixel image. The Simulator is not a neural network (or any other type of machine learning algorithm) but, for example, a Monte Carlo simulation of an experiment with some default set of model parameter values that do not necessarily produce fully-accurate fake images due to uncertainties associated with the understanding of the experiment. Two simulation choices are used in the case studies described in Secs. 5.2.1 and 5.2.2.

Emulator

The Emulator E is a neural network, similar to a generator from a conditional GAN [Mirza and Osindero, 2014], that learns to mimic the simulator. In other words, its aim is to generate identical images to those of the Simulator T when both E and T are fed with the same input parameters $\mathbf{p}$. It is a necessary component of the architecture as it provides the correct back-propagation (it is not generally possible to calculate the derivatives of the simulator) needed by the generator to learn the required model parameter variations. Once the components of the MAGAN have been trained to produce the optimal set of

parameters, the emulator can be used as a fast simulation technique since it produces an accurate emulation of the full simulation running in the simulator step. The specific architecture of the emulator depends on the complexity of the simulation and the number of parameters. In our implementation, which is identical for both case studies, E is a Convolutional Neural Network (CNN) that maps an input parameter vector $\mathbf{p}$ to a two-dimensional output that represents an emulated image. The last layer of E uses a tanh activation function.

Discriminator

Like in regular GANs, the goal of the discriminator D is to distinguish between true data images and images produced by the simulator (or images produced by the emulator to speed up the training process). In our implementation, D is a CNN, and its last layer uses a sigmoid activation function.

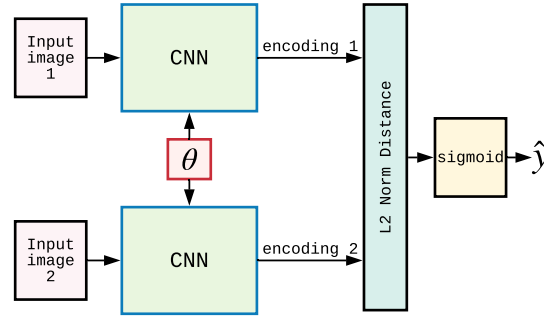


Fig. 5.2 The architecture of the Siamese network, containing two identical CNNs that share the same parameters θ , used to determine the similarity of two input images. Each CNN generates an encoding from a different input image; the CNNs are identical and share the same parameters θ . Then, an L2-norm of the differences of both encodings is applied and used to feed a sigmoid activation function, which outputs the similarity of the input images as a value between 0 and 1.

Siamese network

The Siamese network S [Bromley et al., 1993; Chopra et al., 2005] determines the similarity between the simulated images and the emulated images (both the simulator and the emulator are fed with the parameters $\mathbf{p}$ from the generator) and is used to ensure that the images are as identical as possible. It includes two CNNs that share all their parameters so that they give the same output for a given input. For the i^{th} training example, each CNN generates an encoding $f(\mathbf{x}^{(i)})$ (a 128-length vector) from an input sample $\mathbf{x}^{(i)}$; then, the L2-norm of the differences of both encodings is applied (see Fig. 5.2). Unlike in face-recognition deep-learning tasks [Chopra et al., 2005; Taigman et al., 2014], this

network outputs the probability of two input images to be identical (same pixel map), so there is no need to use a triplet loss¹ [Schroff et al., 2015] unless the simulator T had the ability to output different images (e.g., by introducing some randomness) from the same parameters $\mathbf{p}$. The last layer of the Siamese network uses a sigmoid activation function.

5.1.2 Training details

The two main stages of training the MAGAN, each consisting of two steps, are shown visually in Fig. 5.3: pre-training and training.

- In the pre-training stage, the goal is to learn an emulator distribution $p_E(\mathbf{x})$ that matches the simulator distribution $p_T(\mathbf{x})$. The model learns an emulator network E that generates samples from the emulator distribution p_E by transforming random parameter set variables $p_{\text{random}}(\mathbf{r})$ into samples $E(\mathbf{r})$ such that $E(\mathbf{r}) = T(\mathbf{r})$, where $T(\mathbf{r})$ are samples that the custom simulation technique T generates from the simulator distribution p_T by transforming the same random parameter set variables $p_{\text{random}}(\mathbf{r})$ into the samples. To achieve this, S and E play the following two-player minimax game [Goodfellow et al., 2014] with value function $V_1(E, S)$ ²:

$$\min_E \max_S V_1(S, E) = \mathbb{E}_{\mathbf{r} \sim p_{\text{random}}(\mathbf{r})} [\log S(T(\mathbf{r}), T(\mathbf{r}))] + \mathbb{E}_{\mathbf{r} \sim p_{\text{random}}(\mathbf{r})} [\log(1 - S(T(\mathbf{r}), E(\mathbf{r})))] \quad (5.1)$$

The two pre-training steps using randomised input parameters, normalised between -1 and 1, are:

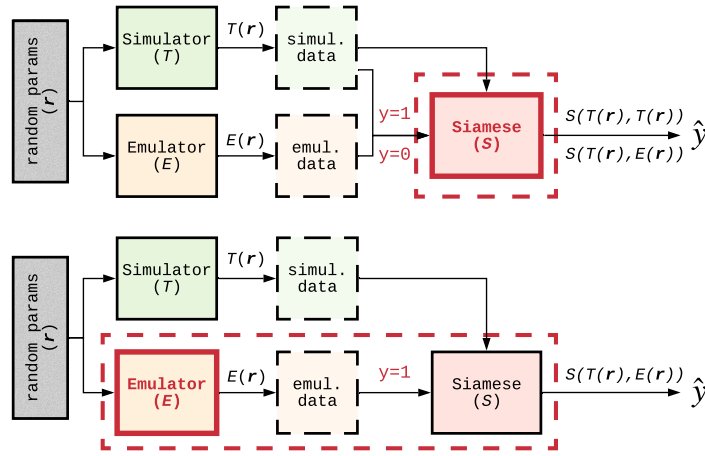
1. The Siamese network is trained to learn the similarity of the simulated and emulated images.
 2. The emulator is trained to learn to create emulated images that mimic simulated images using the Siamese network as trained in step one. The goal is that the emulator produces an image identical to the image created by the simulation for all possible input parameters $\mathbf{p}$.
- In the training stage, the goal is to learn a generator distribution p_G over parameters $\mathbf{p}$, such that the parameters $\mathbf{p}$ predicted by the generator G can be used by the custom simulation technique T to match the true data distribution $p_{\text{data}}(\mathbf{x})$. The

¹Triplet loss: loss function where a baseline (anchor) input is compared to a positive (true) input and a negative (false) input.

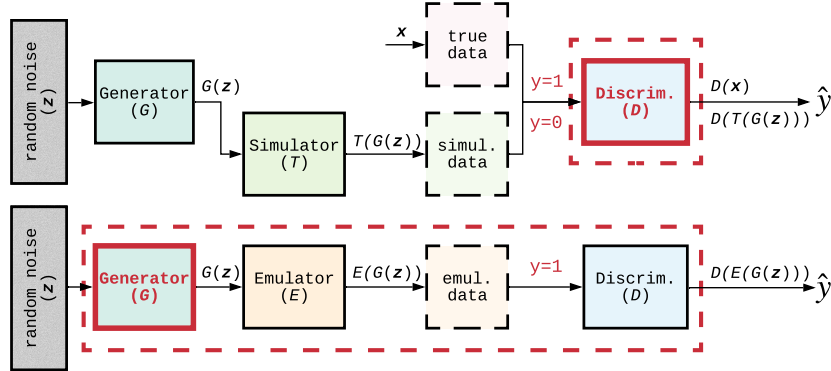
²“The expectation $\mathbb{E}$, or expected value, of some function $f(x)$ with respect to a probability distribution $p(x)$ is the average, or mean value, that f takes on when x is drawn from p .” [Goodfellow et al., 2016]

generator network G generates parameter samples from the generator distribution p_G by transforming random noise variables $p_{noise}(\mathbf{z})$ into samples $G(\mathbf{z})$; then, the pre-trained network E generates samples from $G(\mathbf{z})$ in the form of $E(G(\mathbf{z}))$. To achieve this, D and G play the following two-player minimax game [Goodfellow et al., 2014] with value function $V_2(G, D)^2$:

$$\min_G \max_D V_2(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{noise}(\mathbf{z})} [\log(1 - D(E(G(\mathbf{z}))))]. \quad (5.2)$$



(a) Adversarial pre-training: the Siamese network learns the similarity between the simulator and the emulator images; the emulator learns to make emulated data to mimic simulated data.



(b) Adversarial training: the discriminator learns to distinguish true (or experimental) data from simulated data (or emulated data to speed up the training); the generator learns to predict parameters $\mathbf{p}$ to produce emulated data that best match the true data.

Fig. 5.3 The two training phases of the MAGAN. The red dashed lines enclose the neural network that is trained in each step and the inner red boxes correspond to the trainable modules within the neural network (thus, the weights of the other modules, if any, are frozen).

The two training steps are:

1. The discriminator is trained to distinguish between true data images and simulated images. In the standard configuration shown in Fig. 5.3, $E(G(\mathbf{z}))$ in Eq. 5.2 is replaced by $T(G(\mathbf{z}))$. In order to speed up the training process and assuming the pre-training ended successfully, the simulated images may be replaced by emulated images without damaging the results.
2. The generator learns to predict model parameters such that the images from the emulator cannot be distinguished from the true data.

The architectures of the networks presented above were inspired by DCGANs [Radford et al., 2015]. The MAGAN has been implemented in Keras [Chollet et al., 2015] on top of TensorFlow [Abadi et al., 2015, 2016], and the code is publicly available³.

In order to enhance the training phase, we use some suggestions from [Chintala et al., 2016]. We normalise the image pixel values between -1 and 1 , and use a tanh activation function as the last layer of both the generator and the emulator. To stabilise the training and to provide robustness, we use the label smoothing technique described in Ref. [Salimans et al., 2016] (the label y is set with random values between 0.7 and 1.2 for simulated images, and random values from 0.0 to 0.3 for emulated images), and randomly flip a fraction of simulated and emulated image labels when training the different networks [Arjovsky and Bottou, 2017]. We use the Stochastic Gradient Descent (SGD) optimiser [Bottou et al., 2018] for training the discriminator and the Siamese, and the Adam optimiser [Kingma and Ba, 2014] for training the generator and the emulator as suggested in Refs. [Radford et al., 2015].

5.2 Case studies

5.2.1 Case study 1: first order polynomial images

Consider an experiment that produces 28×28 pixel images with a signal consisting of a first order polynomial with gradient m , constant c and extent in the x-direction x_{steps} defined by the simple equation $y = mx + c$.

The model parameters describing the images are $\mathbf{p} = (m, x_0, c, x_{steps})$, where x_0 is the initial x value, and the effect of each parameter on the simulated images is easily understood. We choose a set of true parameter values $\mathbf{p}_t$ drawn from Gaussian distributions with means μ_i and standard deviation σ_i , where i is the index of the parameter in the range 0 to 3 . The ranges of the parameters are used to represent

³The full MAGAN code is available at <https://gitlab.cern.ch/salonsom/model-assisted-gan>.

variations from different processes that can lead an experiment to have image-to-image variations in the data sample. The mean and standard deviation of the parameters chosen for the true data images are listed in Tab. 5.1.

Parameter	m	x_0	c	x_{steps}
True μ	1.5	10.0	0.5	9.0
True σ	0.3	0.5	0.1	0.5
Best match μ	1.501	9.988	0.512	9.003
Best match σ	0.266	0.418	0.129	0.432

Table 5.1 The mean and standard deviation of the model parameters used to make the true data images and those from the best match parameters learned by the generator in the first case study.

The MAGAN was trained with a mini-batch size of 256 on a single 16GB NVIDIA Tesla V100 GPU. The pre-training stage was trained for 500K iterations, and the training stage for 30K iterations. The best match parameters $\bar{p}_{bm}$, defined as the mean of the best match parameter values from the generator, and the corresponding standard deviations are shown in the bottom two rows of Tab. 5.1 to be in very good agreement with the true data parameters.

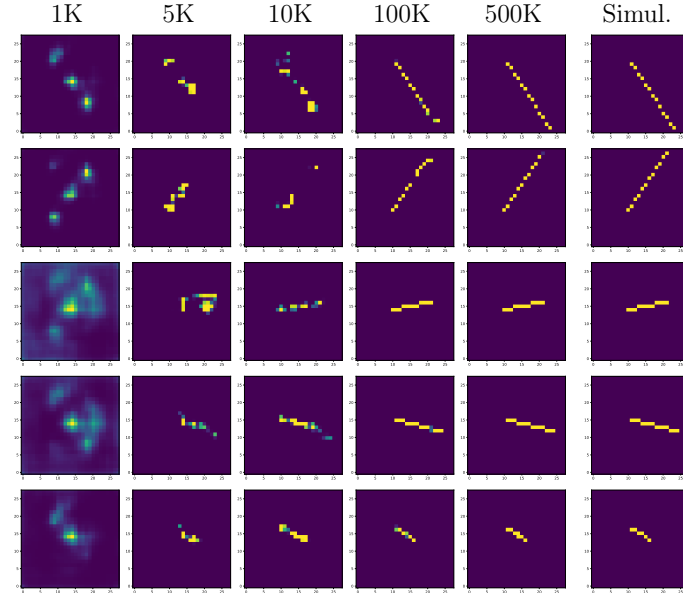


Fig. 5.4 Emulator output (the columns 1-5 show images after 1K, 5K, 10K, 100K, and 500K pre-training iterations, respectively) vs simulator output (column 6). The rows show the images generated from five arbitrary parameter sets.

Example images from the simulator and emulator are shown in Fig. 5.4 for five random parameter sets after 1K, 5K, 10K, 100K and 500K pre-training iterations and demonstrate

the ability of the emulator to mimic the simulator. Figure 5.5 shows simulator images at five stages throughout the training process. In each case three simulated images from randomly chosen sets of generated parameters are shown and compared to three randomly selected true images. These simulated images should not be identical to the true images as they do not have identical parameters, but they are representative of the sample.

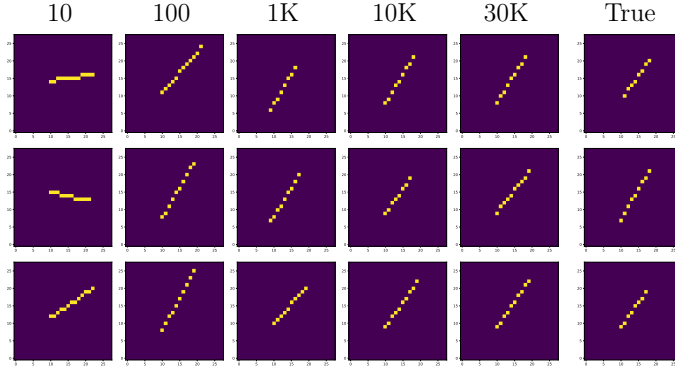


Fig. 5.5 Generator-Simulator output from five random noise vectors (the columns 1-5 show images after 10, 100, 1K, 10K, and 30K training iterations, respectively) vs some random true images (column 6).

5.2.2 Case study 2: circular signal with noise and amplitude variation

We now consider a more complex example containing model parameters that include brightness and noise manipulations as well as topological changes to 28×28 pixel images. The topological description of the signal in the images is:

$$x^2 + y^2 = r^2. \quad (5.3)$$

where r is the radius of the circle. The five model parameters are $\mathbf{p} = (x_0, y_0, r, n, b)$, where: (x_0, y_0) is the centre of the circle; the white noise scale, n , that varies the brightness of the white noise; and the signal brightness, b . Both n and b are defined as a fraction of the maximum image brightness. The set of true data model parameter values $\mathbf{p}_t$ is produced by drawing from Gaussian distributions for each model parameter, as described in Sec. 5.2.1. The mean and standard deviation of the parameter values chosen for the true data images are given in Tab. 5.2.

We used the same training and testing infrastructure described in Sec. 5.2.1. Since the simulator is more complex than in case study one, 1M iterations were needed in the pre-training step. The training step required only 30K iterations, as before.

Parameter	x_0	y_0	r	n	b
True μ	18	18	8	0.15	0.9
True σ	0.8	0.8	0.5	0.05	0.05
Best match μ	18.082	18.035	7.950	0.152	0.892
Best match σ	0.833	0.795	0.411	0.072	0.083

Table 5.2 The mean and standard deviation of the model parameters used to produce the true data images and the corresponding values from the best match parameters from the generator in the second case study.

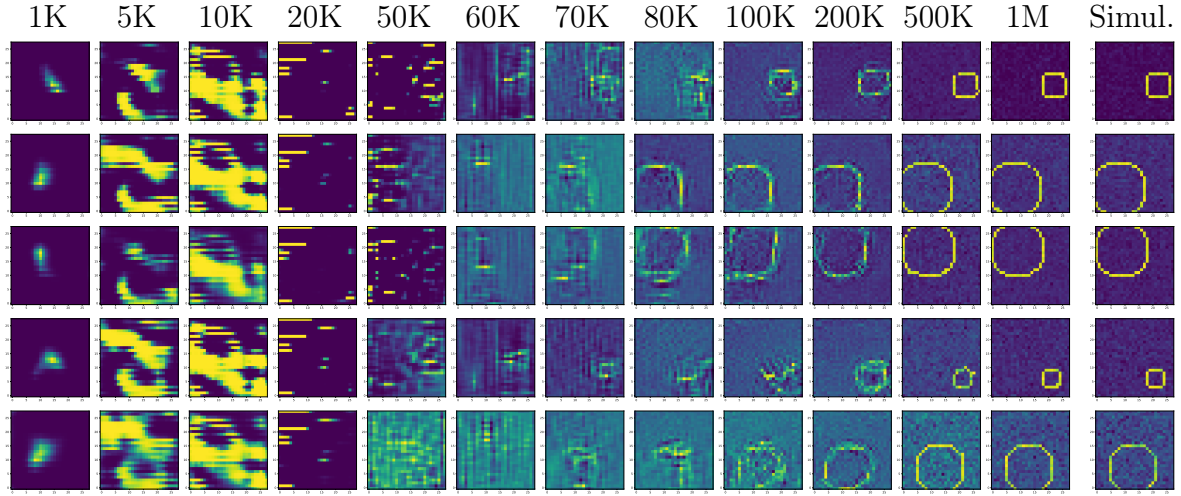


Fig. 5.6 Emulator output (the columns 1-12 show images after 1K, 5K, 10K, 20K, 50K, 60K, 70K, 80K, 100K, 200K, 500K, and 1M pre-training iterations, respectively) vs simulator output (column 13). The rows show the images generated from five arbitrary parameter sets.

The mean and standard deviation of the best match parameters from the trained generator are shown in Tab. 5.2 in comparison to the true data parameters. Excellent agreement is seen for all five parameters showing that the MAGAN performs equally well on this more complex example.

Figure 5.6 shows a comparison of the emulator images for five sets of arbitrarily chosen parameters compared at twelve points in the pre-training process to the simulated image with the same parameters. The emulated images are shown to accurately reproduce the simulated images in this more complex scenario. Figure 5.7 shows three randomly chosen simulated images from different points in the training stage compared to three randomly chosen true images.

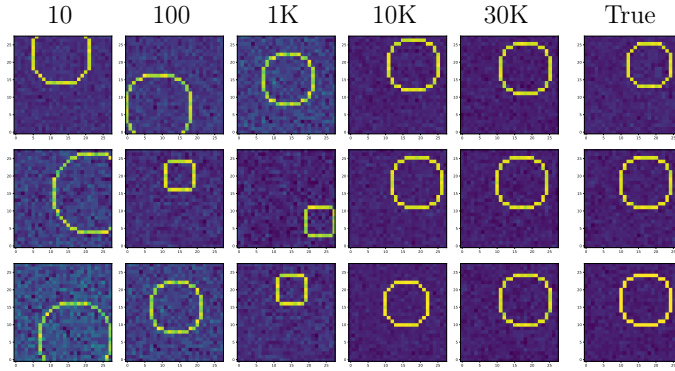


Fig. 5.7 Generator-Simulator output from some random noise vectors (the columns 1-5 show images after 10, 100, 1K, 10K, and 30K training iterations, respectively) vs some random true images (column 6).

5.2.3 Fast simulation with the emulator

Once trained, the emulator produces images very similar to the simulation in considerably less time for the same set of input model parameters $\mathbf{p}$. Table 5.3 shows that in both case studies the emulator running on the GPU is much faster than the simulation, and that the emulator execution time is independent of the complexity of the simulation for a given emulator architecture. The emulator is also considerably quicker using the CPU for case study two, and it is only slower for the first case study since the simulation is very simple, which will not be the case for a real-world experiment. The emulator could hence be used in place of the simulation to allow for the rapid development of analyses that can be performed without needing the exact simulated images.

Image source			Number of generated images		
			10K	100K	1M
Case Study 1	Simulator ¹		2.143 s	21.368 s	213.798 s
	Emulator CPU ¹		6.539 s	66.536 s	672.633 s
	Emulator GPU ²		1.625 s	4.901 s	36.656 s
Case Study 2	Simulator ¹		47.367 s	471.842 s	4,773.626 s
	Emulator CPU ¹		6.540 s	66.541 s	671.628 s
	Emulator GPU ²		1.623 s	4.907 s	36.671 s

Table 5.3 Image production time comparison of the simulator and the emulator for each case study. Each value shows the average time of 10 executions.

¹The experiments were run on an Intel Xeon CPU E5-2698 v4 @ 2.20GHz.

²The experiments were run on a 16GB NVIDIA Tesla V100 GPU.

5.3 Application to the DUNE photon-detector simulation

The DUNE photon-detection (PD) system aims to measure scintillation light resulting from particle interactions within the Far Detector LArTPCs [Abi et al., 2020c]. The simulation of the PD system will guide the design of the experiment; it is a complex task that involves the simulation of the readout electronics, as well as the PD system response to the photons. Light simulations can be memory and CPU intensive. In the PD system simulation, a look-up photon pre-simulated library divides the detector volume into three million voxels, and it is used to access the entire geometry. The current look-up library is stored in memory, which makes the simulation process highly time-consuming [Fermilab Redmine, 2020]. On top of that, the idea is to have higher resolution and cover a larger volume, both of which will make the simulation impossibly large. Figure 5.8 shows an example of the PD geometry.



Fig. 5.8 Anode plane assembly (APA) example. The 10 photon detectors are highlighted with thick horizontal purple lines. Source: <https://cds.cern.ch/record/2280729>.

The approach we present in this section is to try the fast-simulation segment (adversarial pre-training) from our MAGAN to speed things up. In this case, we try to reproduce the look-up table method for the photon simulation.

5.3.1 Problem description

We aim to train the GAN so that it can emulate the library at each 3D-position of the photon emission point. The detector volume is divided into $100 \times 100 \times 300$ voxels, where each voxel has a size of 7.6 cm x 13.6 cm x 5.8 cm. These three million voxels form the training sample, where x, y, z are the input model parameters (Tab. 5.4 shows the geometric extent of the inputs).

Dimension	Minimum (cm)	Maximum (cm)
X	-379.66	379.66
Y	-658.09	699.59
Z	-302.95	1443.50

Table 5.4 Geometric extent of the model parameters.

The test detector geometry⁴ used contains 120 photon detectors embedded in the detector readout planes. There are 12 readout planes in the centre of the detector (two planes high, six planes long), and each plane contains 10 photon detectors. The problem can be simplified into:

- Input (model parameters): (x, y, z) voxel coordinates.
- Output: photon detector system as a 20×6 pixel image, where each pixel gives the visibility of one photon detector (i.e., amount of light observed by each photon detector). Figure 5.9 shows a graphical view of the image format.

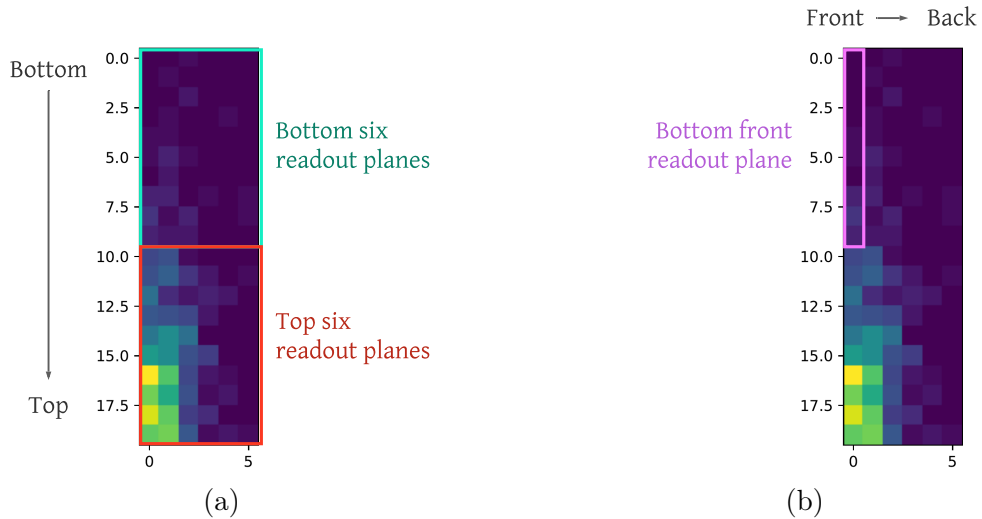


Fig. 5.9 Image format of the light pattern seen across the photon detectors. The images are 20×6 pixels: two readout planes high (a), and six readout planes wide (b).

⁴This is a smaller test geometry used by the DUNE collaboration, not the full far detector geometry.

5.3.2 Evaluation

We ran the pre-training stage of the MAGAN on 3M voxel images for 17K iterations such that the emulator learns to mimic the PD system look-up library. An example of the emulated images compared to the actual library images is shown in Fig. 5.10.

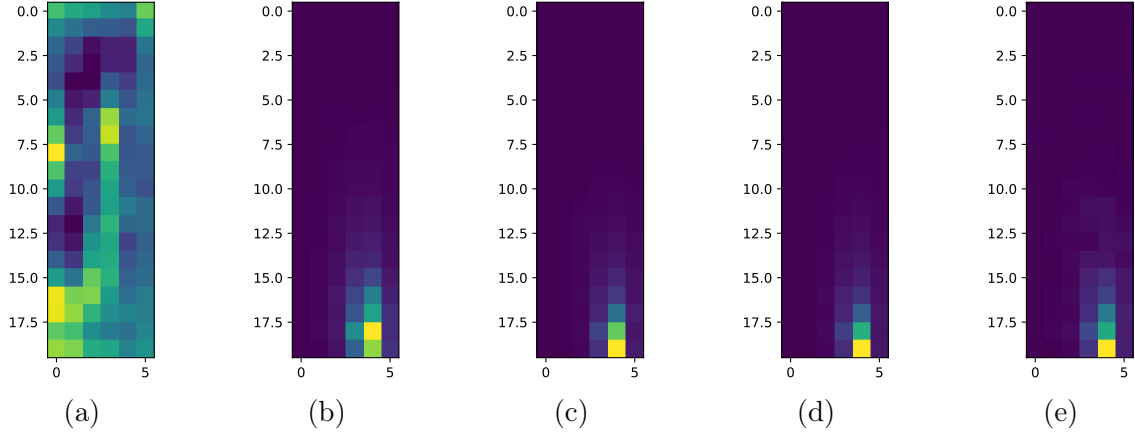


Fig. 5.10 Arbitrary training example. a) Emulated image after 1 iteration, b) 1K iters., c) 10K iters., d) 17K iters; e) library image. There is a close agreement between the final emulated image (d) and the light map from the photon library (e).

Once the MAGAN is trained, the emulator can be used as a fast-simulation technique, as explained in Sec. 2. Table 5.5 shows the time the emulator needs to produce 10K, 100K, and 1M images, respectively; these results exhibit how efficient the emulator of the MAGAN is compared to the original simulation, where a few days are needed to produce 1M of simulated voxel images using the current software.

	Number of generated images		
	10K	100K	1M
CPU	5.919 s	52.088 s	520.017 s
GPU	2.509 s	5.558 s	55.053 s

Table 5.5 Image production time using the emulator. The CPU and GPU are the same ones used for the tests in Sec. 5.2.3. Each value shows the average time of 10 executions.

5.4 Summary

We have proposed and demonstrated the use of model-assisted GANs (MAGANs) to produce physically-motivated manipulations of simulated images through variation of underlying model parameters. In two case studies presented in this chapter, the MAGAN

produces best match parameters $\mathbf{p}_{bm} \sim p_{generator}(\mathbf{p}_{bm})$ in excellent agreement with the true data parameters $\mathbf{p}_t \sim p_{data}(\mathbf{p}_t)$ and hence generates simulated and emulated images that accurately match the true images. In a situation with experimental data instead of true data, the best match parameters $\mathbf{p}_{bm}$ would be used to update the default simulation to produce more accurate images that reproduce the experimental data images. This is critical to minimise biases and ensure similar performance of image-recognition techniques applied to simulated images and data images in experimental situations.

The emulator that is trained as part of the MAGAN can be used as a conditional generator to very quickly produce images very similar to the simulation for a given set of model parameters. The advantages of this method of image production become increasingly clear for complex simulations such as the DUNE photon-detector simulation, and preliminary studies suggest that the first step of the training shown in Fig. 5.3b could use the emulator and emulated data in order to reduce the training time of the MAGAN accumulated from using a very complex simulation.

Chapter 6

Performance study of deep-learning workloads

Being able to run computationally efficient deep-learning workloads is becoming key for both science and industry. On one hand, this chapter studies an approach based on data locality for those neural networks that cannot benefit from scaling their computation due to a significant bottleneck in the data I/O. In addition, we present DaloFlow, an environment for easily deploying distributed trainings using both Horovod and TensorFlow. On the other hand, the plan of this chapter aims to go further and test the implementation of deep-learning techniques on an accelerator board.

6.1 Analysing the distributed training of deep-learning models via data locality

In this section, we study the idea of using a solution based on data locality, ideal for deep-learning applications that have scalability limitations due to data I/O (Sec. 6.1.1). We also present DaloFlow, an easy-to-use environment for deploying distributed trainings using Horovod and TensorFlow (Sec. 6.1.2). DaloFlow is the deployment environment used to perform the evaluation shown in Sec. 6.1.3.

6.1.1 I/O overload study

A typical HPC cluster used for scaling the computation of deep-learning models is composed of several nodes (either CPU or GPU), and the storage system is generally distributed among the nodes, with some local disk space available in each node in most

of the cases. Regarding the storage of training data and the way the nodes can access the training examples, three different scenarios are studied:

1. The training data is stored on only a single node, so all processes must access this node in order to read the input data.
2. The data is accessible through a distributed storage system (such as a Hadoop Distributed File System (HDFS), Gluster File System (GlusterFS), or Lightning Memory-Mapped Database (LMDB)).
3. The data is stored locally in each node, so there is no need to use a distributed storage filesystem or database. In this scenario, the training data should be mirrored among the different nodes.

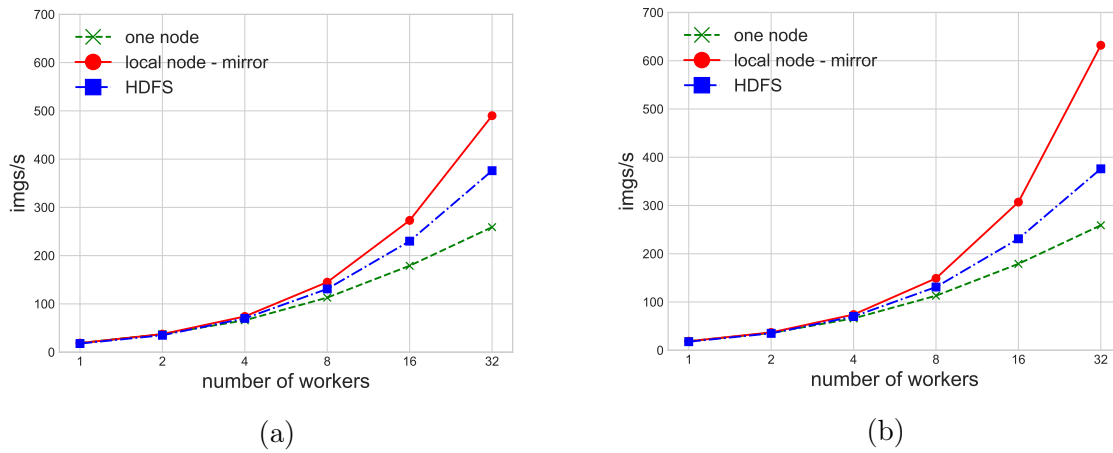


Fig. 6.1 Simulated scenarios using SimGrid. Results show the images processed per second on a cluster consisting of 4 nodes (a), and 16 nodes (b). HDFS is chosen as the distributed storage system.

In order to set up the cluster infrastructure, it is crucial to understand the impact of the above alternatives. Thus, we wanted to identify the data I/O limitations ourselves and study the viability of the approach by simulating the three proposed scenarios using the SimGrid toolkit [Casanova et al., 2014]. We simulated two cluster environments, composed of 4 and 16 nodes, respectively. Each node has a computing speed of 1 GFLOPS, the disk bandwidth is 80 MB/s, and all nodes are connected through a 125 MBps network. The input dataset consists of 1M images of 128x128 pixels each. For simplicity, we choose HDFS as the underlying distributed filesystem for the second scenario. Figure 6.1 shows the results of the simulation performed: alternatives 1) and 2) (green and blue curves in Fig. 6.1, respectively) experience a lower throughput (in terms of images processed per second) compared to alternative 3) (red curve), where workers benefit from data locality and avoid I/O bottlenecks, yielding better scalability.

6.1.2 DaloFlow: a deployment environment for distributed deep-learning computation

An intuitive and easy way to deploy the associated software stack on one computer is needed to alleviate the problem of the rapidly evolving deep-learning software stack. In this way, the reproducibility of experiments might be easily achieved. Docker containers provide a solution to the earlier issues, where a Dockerfile includes all the instructions to install all the software needed. Although Docker containers greatly simplify maintenance tasks, the user needs to know about Docker container management to use deep learning with on-premise cloud platforms. Besides, not all the current the deep-learning software stack is “container-aware”; for example, Horovod documentation points out that OpenMPI might hang due to non-routed network interfaces.

In order to solve the problems above, we propose DaloFlow, a deployment environment to easily run distributed trainings using both Horovod and TensorFlow. More details about DaloFlow are provided in Appendix D. DaloFlow hides the Docker complexity in a transparent way to the user as shown in Fig. 6.2, and provides developers with the following advantages:

- It is based on well-known Horovod Dockerfiles (for both GPU and CPU). The required software can be updated, and different software might be tailored to be used on various containers.
- It is open source (with a GitHub repository¹).

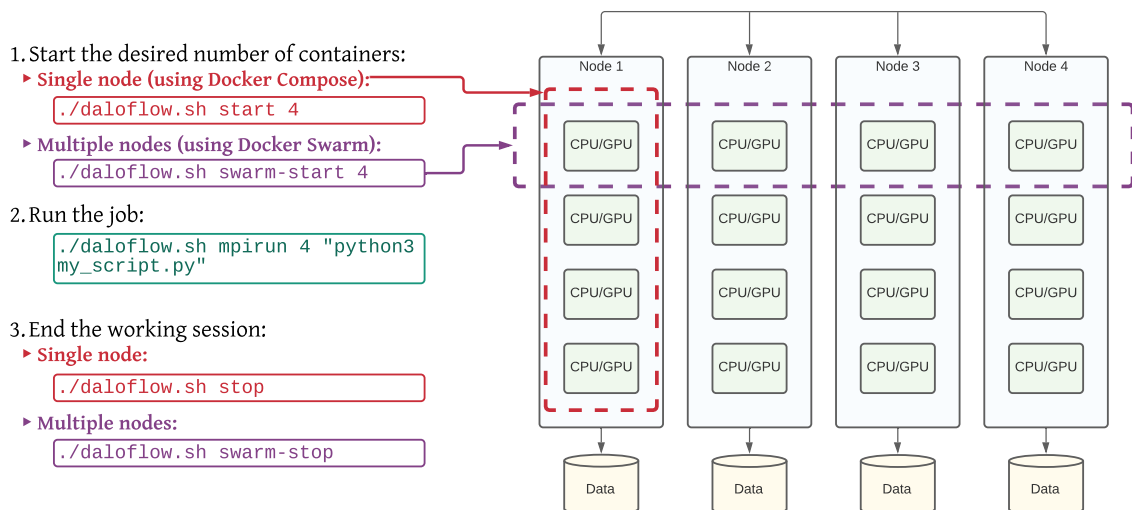


Fig. 6.2 DaloFlow usage example: deployment of 4 containers on a single node (using Docker Compose) or on multiple nodes (using Docker Swarm).

¹The code is available at <https://github.com/saulam/daloflow>.

- It can deploy several containers on a single node or several containers on several nodes (using Docker Swarm).
- It can be used from a single terminal to control a typical work session in a simple way.

6.1.3 Evaluation

The experimental evaluation has been carried out on two clusters (one for CPU and one for GPU) with the following configurations:

- CPU cluster (4 nodes):
 - Hardware: each node has a 12-core Intel(R) Xeon(R) CPU E5-2603 v4 processor with 126 GiB DDR4 of RAM. Additionally, each node has two hard disks, one with a capacity of 256 GiB to store the operating system and applications, and another of 2 TiB of capacity for data. All nodes are connected through a 1 Gbps network.
 - Software: All nodes use an Ubuntu 18.04.5 LTS Linux distribution operating system, Docker version 19.03.13, and docker-compose version 1.27.4. Additional software included are Python 3.6, Tensorflow 2.3.0, Open MPI 4.0.5, Horovod 0.20.3, GlusterFS 3.13.2.
- GPU cluster (4 nodes):
 - Hardware: each node is equipped with 4 NVIDIA Tesla V100-SXM2 with 16 GiB of memory, as well as with an Intel(R) Xeon(R) CPU E5-2698 v4 processor with 140 GiB of RAM. All nodes are connected through a 1 Gbps network.
 - Software: All nodes use an Ubuntu 18.04.5 LTS Linux distribution operating system, Docker version 19.03.13, and docker-compose version 1.27.4. Additional software included are Python 3.6, Tensorflow 2.3.0, Open MPI 4.0.5, Horovod 0.20.3, GlusterFS 3.13.2.

The experimental evaluation has been performed using DaloFlow as the deployment environment. We ran all the experiments over two random datasets of 1M images each, with 32x32 and 128x128 single-precision floating-point pixel images, respectively. We want to compare the number of images processed per second by varying the number of processes for both CPU and GPU in two scenarios: using a distributed filesystem (GlusterFS, in this case) or having the data stored locally. We used three different networks for the

evaluation: a “shallow” network, with a single convolutional layer; a “medium” network, with 10 convolutional layers; and a “deep” network, with 50 convolutional layers. The experimental tests distribute 1, 2, 4 and 8 processes across 1, 2 and 4 nodes. Notice that 1 process cannot be distributed across 2 nodes, and 8 processes across 4 nodes implies each node executes 2 processes.

Image size	Nodes	Model	Scenario	1 proc. (imgs/s)		2 proc. (imgs/s)		4 proc. (imgs/s)		8 proc. (imgs/s)	
				CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
32x32	1	Shallow	Dist. FS	383.0	1,034.5	534.3	1,323.0	971.7	2,199.2	1,523.6	3,712.9
			Data locality	912.4	2,179.7	1,349.0	2,460.0	1,782.9	4,293.9	2,452.5	7,301.3
		Medium	Dist. FS	74.9	405.4	114.1	628.8	143.6	950.7	175.5	1,507.3
			Data locality	179.2	802.9	232.4	1,034.8	292.5	1,675.7	333.6	2,875.3
		Deep	Dist. FS	11.4	35.1	19.9	67.2	29.0	144.1	43.2	266.9
			Data locality	11.7	36.0	21.1	69.0	29.7	146.8	45.3	277.7
	2	Shallow	Dist. FS	-	-	475.6	1,123.5	711.8	1,814.5	1,001.3	2,899.0
			Data locality	-	-	1,124.9	2,157.6	1,563.1	3,822.6	2,156.1	6,133.0
		Medium	Dist. FS	-	-	87.4	572.3	120.6	808.7	165.1	1,442.2
			Data locality	-	-	244.0	912.5	277.2	1522.4	312.0	2566.6
		Deep	Dist. FS	-	-	21.1	65.2	26.0	139.9	40.4	255.2
			Data locality	-	-	21.2	66.1	25.9	141.1	41.9	268.1
	4	Shallow	Dist. FS	-	-	-	-	477.2	1,345.0	743.9	2,562.1
			Data locality	-	-	-	-	1,488.3	3,672.1	2,092.1	6,155.4
		Medium	Dist. FS	-	-	-	-	96.7	734.6	155.7	1,209.5
			Data locality	-	-	-	-	245.2	1,510.2	319.0	2,502.1
		Deep	Dist. FS	-	-	-	-	26.0	125.3	45.5	251.9
			Data locality	-	-	-	-	33.1	135.9	50.5	259.0
128x128	1	Shallow	Dist. FS	16.2	343.8	19.2	248.4	30.1	267.5	10.9	195.3
			Data locality	43.3	785.9	46.2	455.7	53.8	492.8	53.2	514.1
		Medium	Dist. FS	1.5	45.4	1.9	79.9	2.2	91.3	2.5	92.7
			Data locality	3.1	93.3	4.0	145.4	4.2	174.4	4.6	191.2
		Deep	Dist. FS	0.4	17.2	0.7	25.3	0.7	31.8	0.7	32.4
			Data locality	0.6	18.9	0.8	27.4	0.9	42.4	1.0	57.1
	2	Shallow	Dist. FS	-	-	20.2	467.3	21.4	364.0	23.4	375.4
			Data locality	-	-	60.7	1,120.8	61.4	830.2	62.7	823.4
		Medium	Dist. FS	-	-	4.1	75.0	5.1	89.9	5.3	102.6
			Data locality	-	-	5.9	151.2	7.2	242.5	7.2	311.1
		Deep	Dist. FS	-	-	1.1	25.7	1.0	45.4	1.2	75.4
			Data locality	-	-	1.0	27.1	1.2	54.9	1.4	82.6
	4	Shallow	Dist. FS	-	-	-	-	24.7	534.2	35.5	595.5
			Data locality	-	-	-	-	92.0	1,689.9	107.5	1,451.2
		Medium	Dist. FS	-	-	-	-	5.4	101.0	7.9	144.9
			Data locality	-	-	-	-	10.1	333.4	12.2	355.7
		Deep	Dist. FS	-	-	-	-	1.4	56.6	2.7	88.1
			Data locality	-	-	-	-	1.5	68.5	2.8	101.7

Table 6.1 Comparison between a distributed filesystem (GlusterFS, “Dist. FS” in the table) and the solution leveraging data locality (“Data locality”). Results show the average number of processed images per second (throughput) using 1, 2, 4, and 8 processes for both CPU and GPU in 1, 2, and 4 nodes. Each dataset consist of 1M images. The batch size was set to 32 for every run. Model configurations: Shallow (1 conv layer), Medium (10 conv layers), Deep (50 conv layers).

Table 6.1 shows the results of the evaluation performed, from which the following conclusions can be extracted:

- For the case of 32x32 pixel images, the input data is small enough that we do not appreciate big overloads due to I/O, even for the shallow network (see Fig. 6.3a).
- For the case of the 128x128 pixel images, we identify a bottleneck due to I/O, where the throughput does not improve after parallelising the computation, especially for the shallow network (see Fig. 6.3b).
- The two previous items lead to the following statement: the larger the image size and the lower the complexity of the model, the greater the probability of facing I/O limitations.
- The data locality approach is much more efficient than the distributed filesystem one for most of the cases.
- The best speedup is achieved when distributing the computation of the deep models across multiple nodes. This happens because very deep models need more computing time and the I/O rarely becomes a problem.
- GPUs, as expected, are much more efficient processing units than CPUs for deep-learning tasks. However, excellent scalability results on CPU are also shown.

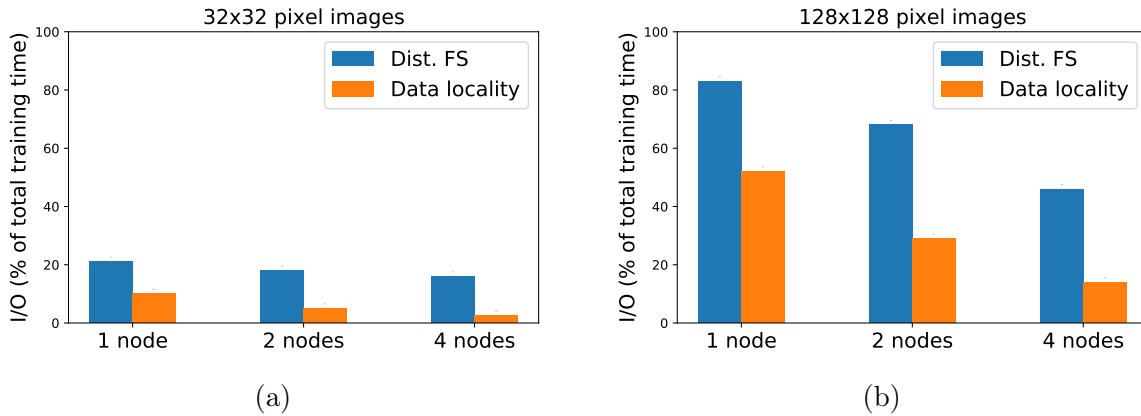


Fig. 6.3 Percentage of the total training time used for I/O in some scenarios from Tab. 6.1: 4 processes running the training of the shallow network on 32x32 (a) or 128x128 (b) pixels images across 1, 2, and 4 GPU nodes using either a distributed filesystem (GlusterFS, “Dist. FS” in the figure) or the data locality solution (“Data locality”). For the distributed filesystem, the percentage of I/O time includes the overload caused by the internal network of the cluster.

6.2 Fast inference using FPGAs for DUNE data reconstruction

The data acquisition (DAQ) system for the DUNE FD gathers beam-related interactions, as well as cosmic-ray muons and atmospheric neutrino interactions; added together, recording their activity will dominate the data rate. Before triggering, the data rate for each 10-kt module is expected to be as much as 1.5 TB/s. The ultimate limit on the output data rate of the DAQ is set by the available permanent storage capacity; this limit is estimated to be about 30 PB/year. Extrapolating to four detector modules, this requires a DAQ data reduction factor of almost four orders of magnitude. In order to meet these demands, new technologies will need to be developed, including high throughput front-end electronics as well as additional FPGA and CPU resources.

Deep-learning techniques, such as deep neural networks (DNN) or convolutional neural networks (CNN), have demonstrated to be extremely useful in particle physics experiments [Adam-Bourdarios et al., 2015; Baldi et al., 2014; Carminati et al., 2017], also in neutrino experiments [Aurisano et al., 2016; Hertel et al., 2017]. However, standard computing infrastructure, i.e., CPUs, is usually not suitable for this ever-increasing technology, so other concrete solutions are needed.

6.2.1 The Micron Deep Learning Accelerator technology

The Micron Deep Learning Accelerator (DLA) is an FPGA-based unit from Micron (SB852) that has been designed for running neural networks with high efficiency, high speed, low power consumption and low latency even with small batches. It has a Xilinx Virtex Ultrascale+ UV9P FPGA, 64 GB of DDR4, 2 GB of HMC memory, 2 QSFP transceiver connectors and a PCIe x16 Gen3 interface. The FPGA contains custom firmware that turns the FPGA into a dedicated processor, with 2 clusters (cores) containing 1024 MAC units each. The MACs are divided among various sub-units (matrix-matrix, matrix-vector and vector-vector) with several parallel connections to internal maps (2MB/cluster) and kernel (512KB/cluster) buffers and the memory interface for optimal access to memory. All operations are performed on 16-bit fixed point values with intermediate results kept in a 32-bit accumulator. This implies a reduction in precision compared to floating-point that has to be considered when designing and deploying neural networks.

The DLA comes with a complete framework that allows quick deployment of existing neural networks designed with common deep-learning frameworks such as Pytorch, TensorFlow, Keras and others. The Micron SDK has a compiler that will compile

networks exported to ONNX (a common neural networks interchange format) into a binary code that the accelerator can run. The compiled code will stay in the accelerator DDR4 memory, which is shared between the FPGA and the host, so different networks can be quickly switched on the accelerator, without programming newer firmware onto the FPGA. Examples are provided with both C and Python code and turning CPU or GPU based code into Micron accelerator code requires modifications to only a few lines of code.

6.2.2 Benchmark using the DUNE CVN

In this section, we will describe the benchmark ran consisting of three independent tests to characterise the Micron DLA. Since the performance of the DUNE CVN is already known and described in Chapter 3, we aimed to check whether we could have the same results using the Micron DLA and obtain an increase of performance. For this purpose, we tested the DLA on three different scenarios, using the DUNE CVN² for all of them.

Micron DLA uses Q8.8 to represent numbers, which is a 16-bit fix-point representation format (8 integer bits, 8 fractional bits), in order to use fractional numbers on low-cost integer hardware. Before compiling the TensorFlow CVN model with the DLA software, it needs to be converted into Open Neural Network Exchange (ONNX) format. ONNX [Tang, 2019] is an open-source neural network format that makes it easier to access hardware optimisations. Once the model is converted into ONNX format, it can be compiled into DLA instructions using the Micron DLA SDK [Micron, 2020].

For the first test, we ran inference continuously over ~ 2 million images, using the SB852. Then we compared the results with the ground truth. Table 6.2 shows the classification report. To fully understand the table, some metrics have to be defined. We define $C_{i,j}$ as the number of elements predicted as category i actually belonging to the category j with $i, j = 1, 2, \dots, n$, where n is the number of categories:

Precision: it measures the number of correctly classified items in a category over all items predicted as this category.

$$Precision(i) = \frac{C_{i,i}}{\sum_{j=1}^n C_{i,j}} \quad (6.1)$$

²The architecture used for the studies showed in this section corresponds to a previous implementation of the CVN, simpler than the network presented in Chapter 3

Recall: is the number of correctly predicted elements in a category divided by the number of actual elements in the category.

$$Recall(i) = \frac{C_{i,i}}{\sum_{j=1}^n C_{j,i}} \quad (6.2)$$

F1-Score: it acts as a weighted average of precision and recall. The F1 score is limited between 0 and 1, where 0 is the worst value, and 1 is the best.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (6.3)$$

Support: is the total number of elements, $\sum C_{i,j} \forall i$, for each category, j .

The results presented in Tab. 6.2 are the expected results for the DUNE CVN, proving that the NN performs correctly in the inference engine.

Interaction	Precision	Recall	F1-score	Support
ν_μ CC QE	0.79	0.80	0.80	113213
ν_μ CC Res	0.59	0.67	0.62	157227
ν_μ CC DIS	0.70	0.77	0.73	203583
ν_μ CC other	0.71	0.24	0.36	54752
ν_e CC QE	0.78	0.79	0.79	110484
ν_e CC Res	0.61	0.70	0.65	154098
ν_e CC DIS	0.68	0.75	0.72	197268
ν_e CC other	0.59	0.43	0.50	54252
ν_τ CC QE	0.56	0.17	0.26	21447
ν_τ CC Res	0.42	0.06	0.10	23373
ν_τ CC DIS	0.50	0.29	0.37	46824
ν_τ CC other	0.49	0.05	0.09	10262
NC	0.91	0.94	0.92	773217

Table 6.2 Classification report for each of the 13 different categories.

The set of $C_{i,j}$ values can be illustrated as a matrix, where the predicted categories, i , correspond to the rows and the actual labels, j , to the columns. This matrix is called “confusion matrix” and helps to interpret the reported results. Figure 6.4 shows the confusion matrix for the classification report. The colour scale of the matrix works the following way: the lightest colour represents cells with no classified events, while the darkest colour represents cells with more than 25k classified events. The elements in the main diagonal show the number of correctly predicted samples.

The highest values tend to cluster around the same neutrino flavour, and that is intrinsic to the neutrino interactions topology. It is easier to distinguish between neutrino flavours than interactions; therefore, sometimes the network mixes the different

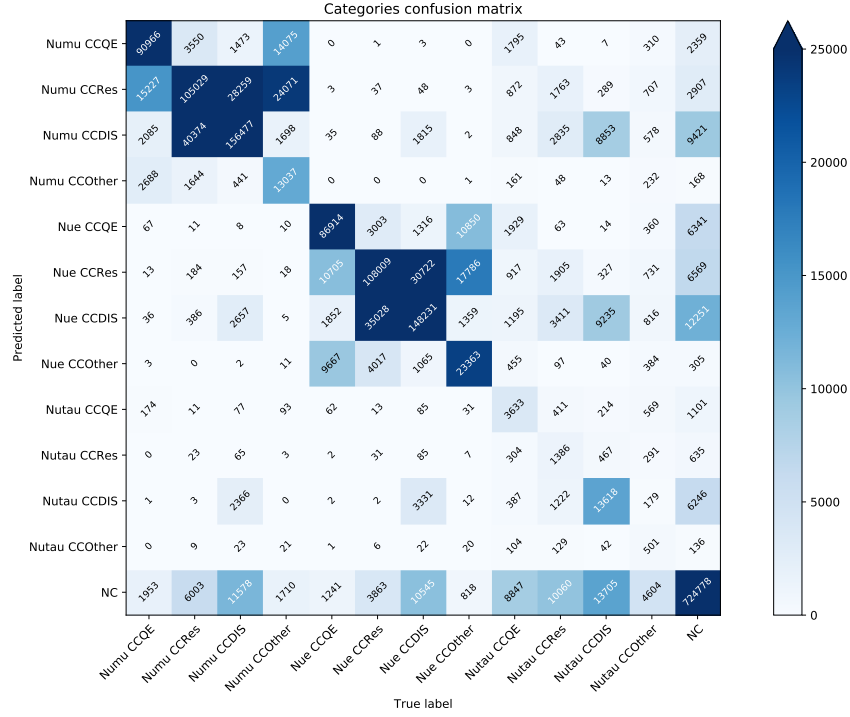


Fig. 6.4 Confusion matrices for the classification task of the different neutrino interaction. A confusion matrix shows the number of elements, $C_{i,j}$, predicted as category i , in rows, belonging to the category j , in columns.

interactions within the same flavour. As mentioned in Chapter 3, the DUNE analysis is focused on the CC ν_μ and CC ν_e selections. Table 6.3 shows the classification report after summing together the scores of the four sub-categories for each neutrino flavour. With an F1-score of 0.94 and 0.93 for CC ν_μ and CC ν_e , respectively, this network maximises the sensitivity of the experiment for the neutrino classification analysis.

Interaction	Precision	Recall	F1-score	Support
CC ν_μ	0.93	0.95	0.94	528775
CC ν_e	0.89	0.96	0.93	516102
CC ν_τ	0.58	0.31	0.40	101906
NC	0.92	0.92	0.92	773217

Table 6.3 Classification report for each neutrino flavour.

For the second test, we reran the same network on a smaller dataset using 1,500 of randomly chosen images. This time, we deployed it on a NVIDIA Tesla V100 GPU and on the SB852 to compare their outputs. The goal is to check if there is any discrepancy due to the loss of precision due to the lack of floating-point arithmetic as mentioned in

Sec. 6.2.1. Figure 6.5 shows the histogram of the absolute error for each of the outputs for all samples. With a standard deviation of 0.0416 and a mean in the order of magnitude of 10^{-10} , we can conclude that the loss of precision is negligible on this test.

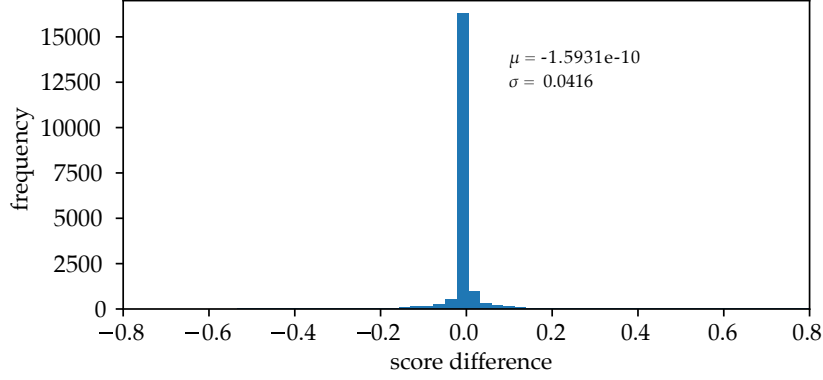


Fig. 6.5 Histogram for the absolute error between the GPU and the SB852 board. The error is calculated as the difference in GPU and SB852 score for the 13 categories over a total of 1500 events.

The aim of the third test carried out is to measure the performance of the SB852 compared to a traditional processor unit. For this test, we used an Intel Core i7-8750H 8th Gen CPU using the Keras package with TensorFlow as the backend deep-learning framework. We enabled multithreading pools in TensorFlow to get the maximum performance of the CPU. On the SB852 side, we used the 4th Gen DLA firmware with 512 MACs running at 250MHz. We ran the inference on a loop of 145 samples and eliminated the first 20 iterations until we reached a steady state. Table 6.4 depicts the results. The average inference time in CPU is 264.85 ms. The SB852 is almost 2.6 times faster, with an average inference time of 103.61 ms.

Processor	Average time (ms)	STD	Min	Max
SB852	103.6074	0.5505	102.4658	105.0381
CPU (i7-8750H)	264.8545	0.8653	262.1692	267.2548

Table 6.4 Inference time comparison between the Micron SB852 and an Intel CPU processor.

6.3 Summary

The work presented in the first half of this chapter shows an efficient way of distribute the training of deep-learning models in an straightforward and transparent way for the

user. The studied method leverages data locality, in the sense that each node maximises the I/O reads from their local hard disks. We also present DaloFlow, a dockerised environment for deploying distributed trainings for either GPU or CPU across multiple processes or nodes. The results presented show that deep-learning applications with I/O load imbalance (i.e., some workers have to contend with others for accessing the hard disk resources) can benefit from the data-locality approach by avoiding I/O bottlenecks.

Also in this chapter, we presented an efficient way to run a deep neural network on FPGAs using the Micron DLA. Due to the amount of data that DUNE will produce per year, approaches that can reduce the data volume are crucial for its smooth operation. We successfully implemented a convolutional neural network conceived to classify neutrino interactions into the Micron DLA SB852. We tested its behaviour over ~ 2 million images with a negligible error compared to its original implementation. Once we characterised the DLA for neutrino physics applications, we plan to move to a more detector-specific scenario, with extremely tight constraints where efficiency in data management and operation is critical. Deep-learning techniques can do the work, but only if they can be deployed efficiently on hardware accelerators that can meet these constraints.

Chapter 7

Conclusion and future work

This thesis has presented a number of deep-learning-based solutions to various open challenges in neutrino physics, which can be valuable to achieving the goals of the next-generation neutrino experiments. At the end of the thesis, the status of the objectives listed in Sec. 1.2 is the following:

- O1 **Proposing algorithms for maximising the accuracy of neutrino event classification.** Chapter 3 presents an algorithm based around a convolutional neural network developed for classifying events in the DUNE far detector. The algorithm provides excellent neutrino flavour classification, reaching efficiencies of 90% for electron neutrinos and 95% for muon neutrinos.
- O2 **Boosting the reconstruction of three-dimensional events by developing rigorous mechanisms for cleaning charged particle tracks.** Chapter 4 proposes the use of a graph neural network for assisting the 3D reconstruction of particle tracks through the rejection of non-physical hits in the SuperFGD detector of the T2K experiment, with efficiencies and purities of 94-96% per event in voxel classification.
- O3 **Providing fast alternatives to standard simulations and trying to minimise the gap between simulated and experimental data.** Chapter 5 introduces model-assisted generative adversarial networks, a technique for performing image-based physics parameter optimisation that produces simulated values in excellent agreement with true data parameters. The emulator segment generates emulated images that accurately match the true images without being time-consuming, and hence it can be used to produce fast simulations.
- O4 **Analysing the computational performance of the techniques studied to end up with more efficient algorithms.** Chapter 6 examines a solution based

on data locality to leverage the performance of distributed deep-learning trainings. It also studies the performance of convolutional neural networks on FPGAs.

7.1 Contributions

Altogether, this thesis provides the following contributions:

- **A convolutional neural network, named CVN, for classifying neutrino interactions in the DUNE far detector;** the CVN provides highly efficient and pure selections of electron neutrino and muon neutrino charged-current interactions, maximising the sensitivity of the experiment to CP -violating effects. This enabled the experiment to demonstrate for the first time that the design CP -violation sensitivity will be achievable, and perhaps exceeded.
- **A graph neural network for 3D classification of ambiguities and optical crosstalk in scintillator-based neutrino detectors;** the network, inspired by the GraphSAGE algorithm, has been tested on the SuperFGD detector of the T2K experiment, reporting promising results and being robust against systematic effects.
- **A new type of generative adversarial network for producing fast simulations and physically-motivated manipulations of simulated images through variation of underlying model parameters;** the approach produces fake images that accurately match true images through the variation of the parameters of the model, whilst the emulator module can be used as a conditional generator for fast fake image production.
- **An analysis of the performance of deep-learning trainings via data locality;** the approach is focused on neural networks that cannot benefit from scaling their computation due to a significant bottleneck in the data I/O. A deployment environment, DaloFlow, for easily run distributed trainings using both Horovod and TensorFlow, is also presented.
- **An evaluation of running a preliminary version of the DUNE CVN on a hardware accelerator board;** the experimental results were run on an FPGA from Micron, outperforming more traditional processing units. The FPGA-based approach is planned to be eventually used for online reconstruction in DUNE.

7.2 Dissemination

The results of this thesis have been published as a number of scientific articles and presented at different events.

7.2.1 Publications

Journals

- **Neutrino interaction classification with a convolutional neural network in the DUNE far detector:** B. Abi *et al.* (DUNE) (**main author**), Physical Review D (PRD), ISSN: 2470-0029, November 2020 [IF 2019: 4.833, Q1] [DOI:10.1103/PhysRevD.102.092003, arXiv:2002.03008].
- **Long-baseline neutrino oscillation physics potential of the DUNE experiment:** B. Abi *et al.* (DUNE) (**significant contributor**), The European Physical Journal C (EPJ C), ISSN: 1434-6052, October 2020 [IF 2019: 4.389, Q1] [DOI:10.1140/EPJC/S10052-020-08456-Z, arXiv:2006.16043].
- **Image-based model parameter optimization using Model-Assisted Generative Adversarial Networks:** S. Alonso-Monsalve, L. H. Whitehead, IEEE Transactions on Neural Networks and Learning Systems (TNNLS), ISSN: 2162-237X, March 2020 [IF 2019: 8.793, Q1] [DOI:10.1109/TNNLS.2020.2969327, arXiv:1812.00879].
- **A heterogeneous mobile cloud computing model for hybrid clouds:** S. Alonso-Monsalve, F. García-Carballeira, A. Calderón, Future Generation Computer Systems (FGCS), ISSN: 0167-739X, April 2018 [IF 2018: 5.768, Q1] [DOI:10.1016/J.FUTURE.2018.04.005].
- **(Submitted) Graph neural network for 3D classification of ambiguities and optical crosstalk in scintillator-based neutrino detectors:** S. Alonso-Monsalve, D. Douqa, C. Jesús-Valls, T. Lux, S. Pina-Otey, F. Sánchez, D. Sgalaberna, L. H. Whitehead, Physical Review D (PRD), ISSN: 2470-0029, 2020 [IF 2019: 4.833, Q1] [arXiv:2009.00688].

Conferences

- **Fast inference using FPGAs for DUNE data reconstruction:** M. J. Rodríguez for the the DUNE Collaboration (**significant contributor**), 24th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2019), Adelaide, South Australia, November 4–8, 2019. [DOI:10.1051/epjconf/202024501030].
- **(Submitted) Analyzing the distributed training of deep-learning models via data locality:** S. Alonso-Monsalve, F. García-Carballeira, A. Calderón, J. Rivadeneira,

29th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2021), March 10–21, 2021.

Workshops

- **Graph neural network for 3D node classification in scintillator-based neutrino detectors:** S. Alonso-Monsalve, D. Douqa, C. Jesús-Valls, T. Lux, S. Pina-Otey, F. Sánchez, D. Sgalaberna, L. H. Whitehead, Physical Sciences workshop at the 34th Conference on Neural Information Processing Systems (NeurIPS 2020), December 11, 2020.

Posters

- **DL-based particle identification system for DUNE (Second prize winner):** M. Rodríguez, S. Alonso-Monsalve, P. Sala, CERN openlab Technical Workshop, CERN, Geneva, Switzerland, January 22-23, 2020.
- **Model-Assisted Generative Adversarial Networks:** S. Alonso-Monsalve, L. H. Whitehead, PHYSTAT-nu 2019, CERN, Geneva, Switzerland, January 22-25, 2019.

7.2.2 Presentations

Conferences

- **Machine learning for neutrino identification:** Session: Plenary, Symposium on Artificial Intelligence for Science, Industry and Society (AISIS 2019), UNAM, Mexico City, Mexico, October 21–25, 2019.

Workshops

- **Deep learning in DUNE:** Session: Collaboration/Project Summary Talks, Neutrino Physics and Machine Learning workshop (NPML), July 10-24, 2020.
- **Graph neural networks for 3D voxel classification in scintillator-based trackers:** Session: Individual talks, Neutrino Physics and Machine Learning workshop (NPML), July 10-24, 2020.
- **Event classification using neural networks in DUNE:** Session: Machine Learning, Reconstruction and Machine Learning in Neutrino Experiments (RML 2020), DESY Hamburg, Germany, September 16–18, 2019.
- **Model-Assisted GANs for the optimisation of simulation parameters and as an algorithm for fast Monte Carlo production:** Session: Submitted contributions, 3rd IML Machine Learning Workshop, CERN, Geneva, Switzerland, April 15–18, 2019.

Invited seminars

- **Deep learning in neutrino experiments:** Seminar, Institute for Particle Physics and Astrophysics, ETH Zurich, Zurich, Switzerland, November 3, 2020.
- **Machine learning in neutrino experiments:** Seminar, Department of Particle Physics and Particle Astrophysics, University of Sheffield, Sheffield, South Yorkshire, England, November 18, 2019.
- **Machine learning in neutrino experiments:** Seminar, Department of Physics and Astronomy, Stony Brook University, New York, USA, October 28, 2019.

Physics meetings

- **Reconstruction with neural networks in DUNE and T2K:** NEUTRINO Research network (GDR), November 24, 2020.
- **3D hit identification in SFGD using neural networks:** Session: Parallel, ND 280 Upgrade Meeting, CERN, Geneva, Switzerland, January 20–22, 2020.
- **Machine Learning:** T2K Cross-Section Workshop, CERN, Geneva, Switzerland, October 16–18, 2019.
- **Neutrino event classification in DUNE:** Session: Machine Learning, PHYSTAT-nu, CERN, Geneva, Switzerland, January 22–25, 2019.

Experiment collaboration meetings

- **CVN occlusion tests:** Session: ProtoDUNE SP: Physics: LBL, DUNE Collaboration Meeting, September 21–25, 2020.
- **Track/shower separation with graph networks:** Session: ProtoDUNE SP: Joint between ProtoDUNE-SP DRA and FD Sim/Reco, DUNE Collaboration Meeting, CERN, Geneva, Switzerland, January 27–31, 2020.
- **Machine Learning in SuperFGD:** Session: ND280 Software+Reconstruction pre-meeting, T2K Collaboration Meeting, Ecole Polytechnique, Paris, France, July 23–27, 2019.
- **CVN Update:** Session: Long-baseline and Simulation/Reconstruction, DUNE Collaboration Meeting, Fermi National Accelerator Laboratory (Fermilab), Illinois, USA, September 24–28, 2018.
- **Event selection for long-baseline analyses:** Session: Plenary, DUNE Collaboration Meeting, Fermi National Accelerator Laboratory (Fermilab), Illinois, USA, May 15–18, 2018.
- **CVN Update:** Session: LBL+Far Detector Sim/Reco, DUNE Collaboration Meeting, Fermi National Accelerator Laboratory (Fermilab), Illinois, USA, May 15–18, 2018.

7.3 Future work

There are several future lines of research resulting from the work presented in this thesis:

- Regarding the work presented in Chapter 3, future studies of possible systematic biases arising from physics models are planned to ensure the robustness of the particle counting outputs. Improvements to the efficiency above 5 GeV may be achieved through the inclusion of more relevant training data, but requires more study. In parallel, the idea is to upgrade the architecture of the CVN to being able to handle sparse representations more efficiently. Finally, there is already some work in progress to adapt the CVN to enhance the neutrino event classification of the ArgoNeuT experiment [Guenette, 2011].
- Similar results to the ones presented in Chapter 4 may be expected for other types of detectors. In particular, the same approach is planned to be applied to detectors that aim to a 3D reconstruction of the neutrino event from 2D projections and that share analogous features like ambiguities and leakage of signal between detector voxels.
- In Chapter 5, we already applied the fast-simulation segment of the MAGAN to the DUNE photon detector simulation, with promising preliminary results. In the future, we will explore fully realistic applications of the MAGAN to real experiments in a number of scientific disciplines, including high energy physics, for both simulation parameter optimisation and fast simulations.
- Concerning the results shown in Chapter 6, future work will focus on ensuring the model convergence of distributed models (compared to the non-distributed alternative) by distributing not only the training computation but the data in a more innovative way. We are also interested in analysing the data locality approach over different scenarios, and propose a mechanism for cases where local storage is not a feasible option.

References

- Aartsen, M. et al. (2017). The IceCube Neutrino Observatory: Instrumentation and Online Systems. *JINST*, 12(03):P03012.
- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.
- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). TensorFlow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.
- Abe, K., Abe, K., Aihara, H., Aimi, A., Akutsu, R., Andreopoulos, C., Anghel, I., Anthony, L. H. V., Antonova, M., Ashida, Y., et al. (2018). Hyper-Kamiokande Design Report.
- Abe, K., Abgrall, N., Aihara, H., Ajima, Y., Albert, J., Allan, D., Amaudruz, P.-A., Andreopoulos, C., Andrieu, B., Anerella, M., et al. (2011a). The T2K experiment. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 659(1):106 – 135.
- Abe, K., Abgrall, N., Ajima, Y., Aihara, H., Albert, J. B., Andreopoulos, C., Andrieu, B., Aoki, S., Araoka, O., Argyriades, J., et al. (2011b). Indication of electron neutrino appearance from an accelerator-produced off-axis muon neutrino beam. *Physical Review Letters*, 107(4).
- Abe, K., Adam, J., Aihara, H., Akiri, T., Andreopoulos, C., Aoki, S., Ariga, A., Assylbekov, S., Autiero, D., Barbi, M., et al. (2015). Measurements of neutrino oscillation in appearance and disappearance channels by the T2K experiment with 6.6×10^{20} protons on target. *Phys. Rev. D*, 91:072010.
- Abe, K. et al. (2013). T2K neutrino flux prediction. *Phys. Rev.*, D87(1):012001. [Addendum: Phys. Rev.D87,no.1,019902(2013)].
- Abe, K. et al. (2019). T2K ND280 upgrade - technical design report.
- Abe, K. et al. (2020). Constraint on the matter–antimatter symmetry-violating phase in neutrino oscillations. *Nature*, 583(7803):339–344.
- Abi, B., Acciarri, R., Acero, M., Adamov, G., Adams, D., Adinolfi, M., Ahmad, Z., Ahmed, J., Alion, T., Alonso Monsalve, S., et al. (2020a). Far detector technical design report, volume II: DUNE physics. *arXiv*.

- Abi, B., Acciarri, R., Acero, M., Adamov, G., Adams, D., Adinolfi, M., Ahmad, Z., Ahmed, J., Alion, T., Alonso Monsalve, S., et al. (2020b). Neutrino interaction classification with a convolutional neural network in the DUNE far detector. *Physical Review D*, 102(9).
- Abi, B., Acciarri, R., Acero, M., Adamov, G., Adams, D., Adinolfi, M., Ahmad, Z., Ahmed, J., Alion, T., Alonso Monsalve, S., et al. (2020c). Volume IV. the DUNE far detector single-phase technology. *Journal of Instrumentation*, 15(08):T08010–T08010.
- Abi, B., Acciarri, R., Acero, M. A., Adamowski, M., Adams, C., Adams, D. L., Adamson, P., Adinolfi, M., Ahmad, Z., Albright, C. H., et al. (2017). The Single-Phase ProtoDUNE Technical Design Report. *arXiv*.
- Abi, B. et al. (2020d). Long-baseline neutrino oscillation physics potential of the DUNE experiment. *The European Physical Journal C*, 80(10):1–34.
- Acciarri, R., Acero, M. A., Adamowski, M., Adams, C., Adamson, P., Adhikari, S., Ahmad, Z., Albright, C. H., Alion, T., Amador, E., et al. (2015). Long-Baseline Neutrino Facility (LBNF) and Deep Underground Neutrino Experiment (DUNE) Conceptual Design Report. *arXiv*.
- Acciarri, R., Adams, C., An, R., Asaadi, J., Auger, M., Bagby, L., Baller, B., Barr, G., Bass, M., Bay, F., et al. (2017a). Convolutional neural networks applied to neutrino events in a liquid argon time projection chamber. *Journal of Instrumentation*, 12(03):P03011–P03011.
- Acciarri, R. et al. (2017b). Design and construction of the MicroBooNE detector. *Journal of Instrumentation*, 12(02):P02017–P02017.
- Acero, M. et al. (2019). First Measurement of Neutrino Oscillation Parameters using Neutrinos and Antineutrinos by NOvA. *Phys. Rev. Lett.*, 123(15):151803.
- Acero, M. A. et al. (2018). New constraints on oscillation parameters from ν_e appearance and ν_μ disappearance in the NOvA experiment. *Phys. Rev.*, D98:032012.
- Adam-Bourdarios, C., Cowan, G., Germain, C., Guyon, I., Kégl, B., and Rousseau, D. (2015). The Higgs boson machine learning challenge. In *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, pages 19–55.
- Adams, C., Alvarez, V., Arazi, L., Arnquist, I. J., Azevedo, C. D. R., Bailey, K., Ballester, F., Benlloch-Rodriguez, J. M., Borges, F. I. G. M., Byrnes, N., et al. (2020a). Sensitivity of a tonne-scale NEXT detector for neutrinoless double beta decay searches.
- Adams, C., Del Tutto, M., Asaadi, J., Bernstein, M., Church, E., Guenette, R., Rojas, J. M., Sullivan, H., and Tripathi, A. (2020b). Enhancing neutrino event reconstruction with pixel-based 3D readout for liquid argon time projection chambers. *JINST*, 15(04):P04009.
- Adams, C. et al. (2019). Deep neural network for pixel-level electromagnetic particle identification in the MicroBooNE liquid argon time projection chamber. *Phys. Rev. D*, 99:092001.

- Adamson, P. et al. (2014). Combined analysis of ν_μ disappearance and $\nu_\mu \rightarrow \nu_e$ appearance in MINOS using accelerator and atmospheric neutrinos. *Phys. Rev. Lett.*, 112:191801.
- Agostinelli, S. et al. (2003). GEANT4: A Simulation toolkit. *Nucl. Instrum. Meth. A*, 506:250–303.
- Aharmim, B. et al. (2005). Electron energy spectra, fluxes, and day-night asymmetries of B-8 solar neutrinos from measurements with NaCl dissolved in the heavy-water detector at the Sudbury Neutrino Observatory. *Phys. Rev. C*, 72:055502.
- Ahmad, Q. R., Allen, R. C., Andersen, T. C., D’Anglin, J., Barton, J. C., Beier, E. W., Bercovitch, M., Bigu, J., Biller, S. D., Black, R. A., et al. (2002). Direct evidence for neutrino flavor transformation from neutral-current interactions in the sudbury neutrino observatory. *Phys. Rev. Lett.*, 89:011301.
- Ahn, M. H. et al. (2006). Measurement of Neutrino Oscillation by the K2K Experiment. *Phys. Rev. D*, 74:072003.
- Aizman, A., Maltby, G., and Breuel, T. (2019). High performance I/O for large scale deep learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 5965–5967.
- Alonso-Monsalve, S., Douqa, D., Jesús-Valls, C., Lux, T., Pina-Otey, S., Sánchez, F., Sgalaberna, D., and Whitehead, L. H. (2020a). Graph neural network for 3D classification of ambiguities and optical crosstalk in scintillator-based neutrino detectors.
- Alonso-Monsalve, S., Douqa, D., Jesús-Valls, C., Lux, T., Pina-Otey, S., Sánchez, F., Sgalaberna, D., and Whitehead, L. H. (2020b). Graph neural network for 3D node classification in scintillator-based neutrino detectors. In *Third Workshop on Machine Learning and the Physical Sciences (NeurIPS 2020)*.
- Alonso-Monsalve, S., García-Carballeira, F., and Calderón, A. (2018). A heterogeneous mobile cloud computing model for hybrid clouds. *Future Generation Computer Systems*, 87:651 – 666.
- Alonso-Monsalve, S. and Whitehead, L. H. (2020). Image-based model parameter optimisation using Model-Assisted Generative Adversarial Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 31(12):5645–5650.
- Amazon (2020). Deep learning in aws.
- An, F. P. et al. (2012). Observation of electron-antineutrino disappearance at Daya Bay. *Phys. Rev. Lett.*, 108:171803.
- Andreopoulos, C., Bell, A., Bhattacharya, D., Cavanna, F., Dobson, J., Dytman, S., Gallagher, H., Guzowski, P., Hatcher, R., Kehayias, P., Mereaglia, A., Naples, D., Pearce, G., et al. (2010). The genie neutrino monte carlo generator. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 614(1):87 – 104.
- Andrews, M., Alison, J., An, S., Burkle, B., Gleyzer, S., Narain, M., Paulini, M., Poczos, B., and Usai, E. (2020). End-to-end jet classification of quarks and gluons with the CMS open data. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 977:164304.

- Antipov, G., Baccouche, M., and Dugelay, J. (2017). Face aging with conditional generative adversarial networks. *CoRR*, abs/1702.01983.
- Araki, T. et al. (2005). Measurement of neutrino oscillation with KamLAND: Evidence of spectral distortion. *Phys. Rev. Lett.*, 94:081801.
- Arjovsky, M. and Bottou, L. (2017). Towards principled methods for training generative adversarial networks. *CoRR*, abs/1701.04862.
- Auger, M. et al. (2012). The EXO-200 detector, part I: Detector design and construction. *JINST*, 7:P05010.
- Aurisano, A., Radovic, A., Rocco, D., Himmel, A., Messier, M. D., Niner, E., Pawloski, G., Psihas, F., Sousa, A., and Vahle, P. (2016). A convolutional neural network neutrino event classifier. *Journal of Instrumentation*, 11(09):P09001.
- Baldi, P., Bauer, K., Eng, C., Sadowski, P., and Whiteson, D. (2016). Jet substructure classification in high-energy physics with deep neural networks. *Phys. Rev. D*, 93:094034.
- Baldi, P., Bian, J., Hertel, L., and Li, L. (2019). Improved energy reconstruction in NOvA with regression convolutional neural networks. *Phys. Rev. D*, 99:012011.
- Baldi, P., Sadowski, P., and Whiteson, D. (2014). Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5(1):4308.
- Barnard, J., Dawe, E. N., Dolan, M. J., and Rajcic, N. (2017). Parton shower uncertainties in jet substructure analyses with deep neural networks. *Phys. Rev. D*, 95:014018.
- Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.
- Beringer, J. et al. (2012). Review of particle physics. *Phys. Rev. D*, 86:010001.
- Bhimji, W., Farrell, S. A., Kurth, T., Paganini, M., Prabhat, and Racah, E. (2018). Deep neural networks for physics analysis on low-level whole-detector data at the LHC. *Journal of Physics: Conference Series*, 1085:042034.
- Birks, J. B. (1951). Scintillations from Organic Crystals: Specific Fluorescence and Relative Response to Different Radiations. *Proceedings of the Physical Society. Section A*, 64(10):874–877.
- Blondel, A., Cadoux, F., Fedotov, S., Khabibullin, M., Khotjantsev, A., Korzenev, A., Kostin, A., Kudenko, Y., Longhin, A., Mefodiev, A., Mermoud, P., Mineev, O., Noah, E., Sgalaberna, D., Smirnov, A., and Yershov, N. (2018). A fully-active fine-grained detector with three readout views. *Journal of Instrumentation*, 13(02):P02006–P02006.
- Blondel, A. et al. (2020). The SuperFGD Prototype Charged Particle Beam Tests.
- Bottou, L., Curtis, F., and Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311.

- Bridle, J. S. (1990a). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In Soulié, F. F. and Hérault, J., editors, *Neurocomputing*, pages 227–236, Berlin, Heidelberg. Springer Berlin Heidelberg.
- Bridle, J. S. (1990b). Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 211–217. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Brock, A., Donahue, J., and Simonyan, K. (2019). Large scale GAN training for high fidelity natural image synthesis.
- Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1993). Signature verification using a "siamese" time delay neural network. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pages 737–744, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Buetti-Dinh, A., Galli, V., Bellenberg, S., Ilie, O., Herold, M., Christel, S., Boretska, M., Pivkin, I. V., Wilmes, P., Sand, W., Vera, M., and Dopson, M. (2019). Deep neural networks outperform human expert's capacity in characterizing bioleaching bacterial biofilm composition. *Biotechnology Reports*, 22:e00321.
- Canziani, A., Paszke, A., and Culurciello, E. (2016). An analysis of deep neural network models for practical applications. *CoRR*, abs/1605.07678.
- Carminati, F., Khattak, G., Pierini, M., Farbin, A., Hooberman, B., Wei, W., Zhang, M., Pacela, V. B., Vallecorsafac, S., Spiropulu, M., and Vlimant, J.-R. (2017). Calorimetry with Deep Learning: Particle Classification, Energy Regression, and Simulation for High-Energy Physics. In *Calorimetry with Deep Learning: Particle Classification, Energy Regression, and Simulation for High-Energy Physics*.
- Casanova, H., Giersch, A., Legrand, A., Quinson, M., and Suter, F. (2014). Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917.
- Chau, L.-L. and Keung, W.-Y. (1984). Comments on the Parametrization of the Kobayashi-Maskawa Matrix. *Phys. Rev. Lett.*, 53:1802.
- Chen, C., Zhang, P., Zhang, H., Dai, J., Yi, Y., Zhang, H., and Zhang, Y. (2020). Deep learning on computational-resource-limited platforms: A survey. *Mobile Information Systems*, 2020.
- Chen, J., Pan, X., Monga, R., Bengio, S., and Jozefowicz, R. (2017). Revisiting distributed synchronous SGD.
- Chen, J. and Ran, X. (2019). Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107(8):1655–1674.
- Chen, X., Duan, Y., Houthoofd, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems 29*, NIPS'16, page 2180–2188. Curran Associates Inc., Red Hook, NY, USA.

- Chilimbi, T. M., Suzue, Y., Apacible, J., and Kalyanaraman, K. (2014). Project Adam: Building an efficient and scalable deep learning training system. In *OSDI*.
- Chintala, S., Denton, E., Arjovsky, M., and Mathieu, M. (2016). How to train a GAN? tips and tricks to make GANs work. <https://github.com/soumith/ganhacks>.
- Chollet, F. (2016). Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357.
- Chollet, F. et al. (2015). Keras. <https://github.com/keras-team/keras>.
- Choma, N., Monti, F., Gerhardt, L., Palczewski, T., Ronaghi, Z., Prabhat, P., Bhimji, W., Bronstein, M. M., Klein, S. R., and Bruna, J. (2018). Graph neural networks for IceCube signal classification. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 386–391.
- Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546 vol. 1.
- Chowdhury, F., Zhu, Y., Heer, T., Paredes, S., Moody, A., Goldstone, R., Mohror, K., and Yu, W. (2019). I/O characterization and performance evaluation of BeeGFS for deep learning. In *Proceedings of the 48th International Conference on Parallel Processing, ICPP 2019, New York, NY, USA*. Association for Computing Machinery.
- Cogan, J., Kagan, M., Strauss, E., and Schwartzman, A. (2015). Jet-images: computer vision inspired techniques for jet tagging. *Journal of High Energy Physics*, 2015(2):118.
- Cowan, C. L., Reines, F., Harrison, F. B., Kruse, H. W., and McGuire, A. D. (1956). Detection of the free neutrino: a confirmation. *Science*, 124(3212):103–104.
- Creswell, A. and Bharath, A. A. (2018). Inverting the generator of a generative adversarial network. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–8.
- Danby, G., Gaillard, J.-M., Goulianos, K., Lederman, L. M., Mistry, N., Schwartz, M., and Steinberger, J. (1962). Observation of high-energy neutrino reactions and the existence of two kinds of neutrinos. *Phys. Rev. Lett.*, 9:36–44.
- de Oliveira, L., Kagan, M., Mackey, L., Nachman, B., and Schwartzman, A. (2016). Jet-images — deep learning edition. *Journal of High Energy Physics*, 2016:69.
- de Oliveira, L., Paganini, M., and Nachman, B. (2017). Learning particle physics by example: Location-aware generative adversarial networks for physics synthesis. *Computing and Software for Big Science*, 1(1):4.
- Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., aurelio Ranzato, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., and Ng, A. Y. (2012). Large scale distributed deep networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1223–1231. Curran Associates, Inc.
- deeplearning.ai (2015). Deep Learning Specialization. <https://www.coursera.org/specializations/deep-learning>.

- DeepLearnPhysics (2020). DeepLearnPhysics: Research collaboration. <http://deeplearnphysics.org>.
- Delaquis, S. et al. (2018). Deep Neural Networks for Energy and Position Reconstruction in EXO-200. *JINST*, 13(08):P08023.
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.
- Dogo, E. M., Afolabi, O. J., Nwulu, N. I., Twala, B., and Aigbavboa, C. O. (2018). A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks. In *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*, pages 92–99.
- Dominé, L., de Soux, P. C., Drielsma, F., Koh, D. H., Ran Itay, Q. L., Terao, K., Tsang, K. V., and Usher, T. L. (2020). Point proposal network for reconstructing 3D particle positions with sub-pixel precision in liquid argon time projection chambers. *ArXiv*, abs/2006.14745.
- Dominé, L. and Terao, K. (2020). Scalable deep convolutional neural networks for sparse, locally dense liquid argon time projection chamber data. *Phys. Rev. D*, 102:012005.
- Drielsma, F., Lin, Q., de Soux, P. C., Dominé, L., Itay, R., Koh, D. H., Nelson, B. J., Terao, K., Tsang, K. V., and Usher, T. L. (2020a). Clustering of electromagnetic showers and particle interactions with graph neural networks in liquid argon time projection chambers data.
- Drielsma, F., Lin, Q., and Terao, K. (2020b). Particle clustering and flow reconstruction for particle imaging neutrino detectors using graph neural networks.
- Duarte, J., Han, S., Harris, P., Jindariani, S., Kreinar, E., Kreis, B., Ngadiuba, J., Pierini, M., Rivera, R., Tran, N., and Wu, Z. (2018). Fast inference of deep neural networks in FPGAs for particle physics. *Journal of Instrumentation*, 13(7).
- Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12(null):2121–2159.
- Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional Networks on Graphs for Learning Molecular Fingerprints. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 2224–2232. Curran Associates, Inc.
- Escudero, L. (2016). Initial probe of δCP by the T2K experiment with ν_μ disappearance and ν_e appearance. *Nuclear and Particle Physics Proceedings*, 273-275:1814 – 1819. 37th International Conference on High Energy Physics (ICHEP).
- Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., Cui, C., Corrado, G., Thrun, S., and Dean, J. (2019). A guide to deep learning in healthcare. *Nature medicine*, 25(1):24–29.

- Evans, L. and Bryant, P. (2008). LHC machine. *Journal of Instrumentation*, 3(08):S08001–S08001.
- Farrell, S., Calafiura, P., Mudigonda, M., Prabhat, Anderson, D., Vlimant, J.-R., Zheng, S., Bendavid, J., Spiropulu, M., Cerati, G., Gray, L., Kowalkowski, J., Spentzouris, P., and Tsaris, A. (2018). Novel deep learning methods for track reconstruction.
- Feng, X., Jiang, Y., Yang, X., Du, M., and Li, X. (2019). Computer vision algorithms and hardware implementations: A survey. *Integration*, 69:309 – 320.
- Fermi, E. (1934). An attempt of a theory of beta radiation. 1. *Z. phys*, 88(161):19.
- Fermilab Redmine (2020). DUNE computing and software.
- Fukuda, Y. et al. (1998a). Evidence for oscillation of atmospheric neutrinos. *Phys. Rev. Lett.*, 81:1562–1567.
- Fukuda, Y., Hayakawa, T., Ichihara, E., Inoue, K., Ishihara, K., Ishino, H., Itow, Y., Kajita, T., Kameda, J., Kasuga, S., et al. (1998b). Evidence for oscillation of atmospheric neutrinos. *Phys. Rev. Lett.*, 81:1562–1567.
- Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterton, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.
- Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA. PMLR.
- Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc.
- Google (2020). Google cloud TPU system architecture.
- Graham, B., Engelcke, M., and van der Maaten, L. (2017). 3D semantic segmentation with submanifold sparse convolutional networks.
- Graham, B. and van der Maaten, L. (2017). Submanifold sparse convolutional networks.
- Grossberg, S. (1988). Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks*, 1(1):17 – 61.
- Gu, J., Chowdhury, M., Shin, K. G., Zhu, Y., Jeon, M., Qian, J., Liu, H., and Guo, C. (2019). Tiresias: A GPU cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 485–500, Boston, MA. USENIX Association.

- Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., and Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354 – 377.
- Guenette, R. (2011). The ArgoNeuT experiment.
- Guest, D., Cranmer, K., and Whiteson, D. (2018). Deep learning and its application to LHC physics. *Annual Review of Nuclear and Particle Science*, 68(1):161–181.
- Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Inductive Representation Learning on Large Graphs.
- Haskins, G., Kruger, U., and Yan, P. (2020). Deep learning in medical image registration: a survey. *Machine Vision and Applications*, 31(1):8.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015a). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015b). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Identity mappings in deep residual networks. *CoRR*, abs/1603.05027.
- Hertel, L., Li, L., Baldi, P., and Bian, J. (2017). Convolutional neural networks for electron neutrino and electron shower energy reconstruction in the NOvA detectors. In *Convolutional Neural Networks for Electron Neutrino and Electron Shower Energy Reconstruction in the NOvA Detectors*.
- Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.
- Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558.
- Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861.
- Howard Jr, B. L. (2019). *Toward a Precision Era of Neutrino Oscillation Physics: Liquid Argon Scintillation Detector Development for DUNE and Neutrino Oscillation Studies with NOvA*. PhD thesis, Indiana University.
- Hu, J., Shen, L., and Sun, G. (2017). Squeeze-and-excitation networks. *CoRR*, abs/1709.01507.
- Hu, Z., Yang, Z., Salakhutdinov, R., Liang, X., Qin, L., Dong, H., and Xing, E. (2018). Deep generative models with learnable knowledge constraints. *arXiv:1806.09764*.
- Huennefeld, M. (2018). Deep Learning in Physics exemplified by the Reconstruction of Muon-Neutrino Events in IceCube. *PoS, ICRC2017*:1057.

- Huenefeld, M. (2019). Reconstruction Techniques in IceCube using Convolutional and Generative Neural Networks. *EPJ Web Conf.*, 207:05005.
- Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally, W. J., and Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR*, abs/1602.07360.
- Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.
- Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*.
- Jin, P. H., Yuan, Q., Iandola, F., and Keutzer, K. (2016). How to scale distributed deep learning?
- Jouppi, N. (2020). Google supercharges machine learning tasks with TPU custom chip.
- Józefowicz, R., Richter-Was, E., and Was, Z. (2016). Potential for optimizing the Higgs boson CP measurement in $h \rightarrow \tau\tau$ decays at the LHC including machine learning techniques. *Phys. Rev. D*, 94:093001.
- Ju, X., Farrell, S., Calafiura, P., Murnane, D., Prabhat, Gray, L., Klijsma, T., Pedro, K., Cerati, G., Kowalkowski, J., Perdue, G., Spentzouris, P., Tran, N., Vlimant, J.-R., Zlokapa, A., Pata, J., Spiropulu, M., An, S., Aurisano, A., Hewes, J., Tsaris, A., Terao, K., and Usher, T. (2020). Graph Neural Networks for Particle Reconstruction in High Energy Physics detectors.
- Justesen, N., Bontrager, P., Togelius, J., and Risi, S. (2020). Deep learning for video game playing. *IEEE Transactions on Games*, 12(1):1–20.
- Kayser, B. (1981). On the quantum mechanics of neutrino oscillation. *Phys. Rev. D*, 24:110–116.
- Kekic, M. et al. (2020). Demonstration of background rejection using deep convolutional neural networks in the NEXT experiment.
- Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.
- Kipf, T. N. and Welling, M. (2016). Semi-Supervised Classification with Graph Convolutional Networks.
- Kodama, K., Ushida, N., Andreopoulos, C., Saoulidou, N., Tzanakos, G., Yager, P., Baller, B., Boehnlein, D., Freeman, W., Lundberg, B., et al. (2001). Observation of tau neutrino interactions. *Physics Letters B*, 504(3):218 – 224.
- Koh, D. H., de Soux, P. C., Dominé, L., Drielsma, F., Itay, R., Lin, Q., Terao, K., Tsang, K. V., and Usher, T. (2020). Scalable, proposal-free instance segmentation network for 3D pixel clustering and particle trajectory reconstruction in liquid argon time projection chambers.

- Komiske, P. T., Metodiev, E. M., and Schwartz, M. D. (2017). Deep learning in color: towards automated quark/gluon jet discrimination. *Journal of High Energy Physics*, 2017(1):110.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA. Curran Associates Inc.
- Kronmueller, M. and Glauch, T. (2020). Application of Deep Neural Networks to Event Type Classification in IceCube. *PoS, ICRC2019*:937.
- Kuutti, S., Bowden, R., Jin, Y., Barber, P., and Fallah, S. (2020). A survey of deep learning applications to autonomous vehicle control. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–22.
- Lateef, F. and Ruichek, Y. (2019). Survey on semantic segmentation using deep learning techniques. *Neurocomputing*, 338:321 – 348.
- LeCun, Y., Bengio, Y., and Hinton, G. E. (2015). Deep learning. *Nature*, 521(7553):436–444.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- LeCun, Y., Kavukcuoglu, K., and Farabet, C. (2010). Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256.
- Li, Z. et al. (2019). Simulation of charge readout with segmented tiles in nEXO. *JINST*, 14(09):P09020.
- Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *CoRR*, abs/1312.4400.
- Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., and Pietikäinen, M. (2020). Deep learning for generic object detection: A survey. *International journal of computer vision*, 128(2):261–318.
- Louppe, G., Kagan, M., and Cranmer, K. (2016). Learning to pivot with adversarial networks. *CoRR*, abs/1611.01046.
- Lukas, W. (2012). Fast simulation for ATLAS: Atlfast-II and ISF. *Journal of Physics: Conference Series*, 396(2):022031.
- Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on International Conference on Machine Learning, series = ICML'13*,, page 3.
- Maki, Z., Nakagawa, M., and Sakata, S. (1962). Remarks on the Unified Model of Elementary Particles. *Progress of Theoretical Physics*, 28(5):870–880.

- Martin, G. and Smith, G. (2009). High-level synthesis: Past, present, and future. *IEEE Design and Test of Computers*, 26(4):18–25.
- Mayer, R. and Jacobsen, H.-A. (2020). Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools. *ACM Comput. Surv.*, 53(1).
- McCullagh, P. and Nelder, J. (1989). Generalized linear models.
- McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.
- MicroBooNE Collaboration (2020). Semantic segmentation with sparse convolutional neural network for event reconstruction in MicroBooNE.
- Micron (2020). Micron DLA SDK. <https://github.com/FWDNXT/SDK>.
- Milioto, A., Lottes, P., and Stachniss, C. (2018). Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in cnns. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2229–2235.
- Minaee, S., Boykov, Y., Porikli, F., Plaza, A., Kehtarnavaz, N., and Terzopoulos, D. (2020). Image segmentation using deep learning: A survey.
- Mineev, O. et al. (2019). Beam test results of 3D fine-grained scintillator detector prototype for a T2K ND280 neutrino active target. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 923:134–138.
- Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *CoRR*, abs/1411.1784.
- Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML’10*, pages 807–814, USA. Omnipress.
- Ng, A. Y. (2004). Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-First International Conference on Machine Learning, ICML’04*, page 78, New York, NY, USA. Association for Computing Machinery.
- Otter, D. W., Medina, J. R., and Kalita, J. K. (2020). A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21.
- Paganini, M., de Oliveira, L., and Nachman, B. (2018a). Accelerating Science with Generative Adversarial Networks: An Application to 3D Particle Showers in Multilayer Calorimeters. *Phys. Rev. Lett.*, 120(4):042003.
- Paganini, M., de Oliveira, L., and Nachman, B. (2018b). CaloGAN: Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks. *Phys. Rev. D*, 97:014021.
- Pamela Vagata, K. W. (2014). Scaling the Facebook data warehouse to 300 PB.

- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.
- Patterson, R. (2013). The NOvA experiment: status and outlook. *Nuclear Physics B - Proceedings Supplements*, 235-236:151 – 157. The XXV International Conference on Neutrino Physics and Astrophysics.
- Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 701–710, New York, NY, USA. Association for Computing Machinery.
- Pontecorvo, B. (1957). Mesonium and anti-mesonium. *Sov. Phys. JETP*, 6:429.
- Pontecorvo, B. (1960). Electron and Muon Neutrinos. *Sov. Phys. JETP*, 10:1236–1240.
- Pontecorvo, B. (1968). Neutrino Experiments and the Problem of Conservation of Leptonic Charge. *Sov. Phys. JETP*, 26:984–988. [*Zh. Eksp. Teor. Fiz.* 53, 1717 (1967)].
- Prechelt, L. (1996). Early stopping-but when? In Orr, G. B. and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 55–69. Springer.
- Pumma, S., Buono, D., Checconi, F., Que, X., and Feng, W. (2020). Alleviating load imbalance in data processing for large-scale deep learning. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 262–271.
- Pumma, S., Si, M., Feng, W., and Balaji, P. (2017). Towards scalable deep learning via I/O analysis and optimization. In *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 223–230.
- Pumma, S., Si, M., Feng, W.-C., and Balaji, P. (2019). Scalable deep learning via I/O analysis and optimization. *ACM Trans. Parallel Comput.*, 6(2).
- Qasim, S. R., Kieseler, J., Iiyama, Y., and Pierini, M. (2019). Learning representations of irregular particle-detector geometry with distance-weighted graph networks. *The European Physical Journal C*, 79(7).
- Racah, E., Ko, S., Sadowski, P., Bhimji, W., Tull, C., Oh, S., Baldi, P., and Prabhat (2016). Revealing fundamental physics from the Daya Bay neutrino experiment using deep neural networks. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 892–897.
- Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434.
- Radovic, A., Williams, M., Rousseau, D., Kagan, M., Bonacorsi, D., Himmel, A., Aurisano, A., Terao, K., and Wongjirad, T. (2018). Machine learning at the energy and intensity frontiers of particle physics. *Nature*, 560(7716):41–48.

- Reines, F. and Cowan, C. L. (1953). Detection of the free neutrino. *Phys. Rev.*, 92:830–831.
- Ren, S., He, K., Girshick, R., and Sun, J. (2017). Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149.
- Renner, J. et al. (2017). Background rejection in NEXT using deep neural networks. *JINST*, 12(01):T01004.
- Rodriguez, M. J. (2020). Fast inference using fpgas for dune data reconstruction. *EPJ Web Conf.*, 245:01030.
- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386.
- Rubbia, C. (1977). The liquid-argon time projection chamber: a new concept for neutrino detectors. Technical Report CERN-EP-INT-77-8, CERN, Geneva.
- Ruder, S. (2017). An overview of gradient descent optimization algorithms.
- Sadowski, P., Radics, B., Ananya, Yamazaki, Y., and Baldi, P. (2017). Efficient antihydrogen detection in antimatter physics by deep learning.
- Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training GANs. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS’16, pages 2234–2242, USA. Curran Associates Inc.
- Schawinski, K., Zhang, C., Zhang, H., Fowler, L., and Santhanam, G. K. (2017). Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit. *MNRAS*, 467(1):L110–L114.
- Schroff, F., Kalenichenko, D., and Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823.
- Seong, I., Hertel, L., Collado, J., Li, L., Nayak, N., Bian, J., and Baldi, P. (2019). Convolutional neural networks for energy and vertex reconstruction in DUNE. In *Second Workshop on Machine Learning and the Physical Sciences (NeurIPS 2019)*.
- Sergeev, A. and Balso, M. D. (2018). Horovod: fast and easy distributed deep learning in tensorflow.
- Serpa, M. S., Krause, A. M., Cruz, E. H. M., Navaux, P. O. A., Pasin, M., and Felber, P. (2018). Optimizing machine learning algorithms on multi-core and many-core architectures using thread and data mapping. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 329–333.
- Shelhamer, E., Long, J., and Darrell, T. (2017). Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):640–651.

- Shen, Y., Li, H., Yi, S., Chen, D., and Wang, X. (2018). Person Re-identification with Deep Similarity-Guided Graph Neural Network. In *The European Conference on Computer Vision (ECCV)*.
- Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60.
- Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.
- Sjöstrand, T. (1994). High-energy-physics event generation with PYTHIA 5.7 and JETSET 7.4. *Computer Physics Communications*, 82(1):74 – 89.
- Smithing, N. (1999). Supervised learning in feedforward artificial neural networks.
- Sperduti, A. and Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.
- Strigl, D., Kofler, K., and Podlipnig, S. (2010). Performance and scalability of GPU-based convolutional neural networks. In *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-Based Processing, PDP 2010*, pages 317–324.
- Sturman, O., von Ziegler, L., Schläppi, C., Akyol, F., Privitera, M., Slominski, D., Grimm, C., Thieren, L., Zerbi, V., Grewe, B., et al. (2020). Deep learning-based behavioral analysis reaches human accuracy and is capable of outperforming commercial solutions. *Neuropsychopharmacology*, 45(11):1942–1952.
- Sun, R. (2019). Optimization for deep learning: theory and algorithms.
- Sword-Fehlberg, S. (2017). Multi-particle identification using convolutional neural networks in MicroBooNE. *Bulletin of the American Physical Society*, 62.
- Szegedy, C., Ioffe, S., and Vanhoucke, V. (2016). Inception-v4, Inception-ResNet and the impact of residual connections on learning. *CoRR*, abs/1602.07261.
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, abs/1409.4842.
- Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567.
- Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. (2014). DeepFace: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708.
- Tanabashi, M. et al. (2018). Review of Particle Physics. *Phys. Rev. D*, 98(3):030001.
- Tang, Y. (2019). ONNX: Open Neural Network Exchange. <https://onnx.ai>.

- Teichmann, M., Weber, M., Zöllner, M., Cipolla, R., and Urtasun, R. (2018). MultiNet: Real-time joint semantic reasoning for autonomous driving. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1013–1020.
- Teodorescu, L. (2008). Artificial neural networks in high-energy physics. In *Inverted CERN School of Computing (iCSC2006)*, pages 13–22.
- Terao, K. (2020). ML-based data reconstruction for particle imaging neutrino detectors. IML Workshop 2020.
- The ATLAS Collaboration (2017). Quark versus gluon jet tagging using jet images with the ATLAS detector. Technical Report ATL-PHYS-PUB-2017-017, CERN, Geneva.
- Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688.
- Tieleman, T. and Hinton, G. (2012). Lecture 6.5-RMSprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.
- Toronto, I. (2018). The computing power needed to train ai is growing alarmingly.
- Trimberger, S. M. (2015). Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology. *Proceedings of the IEEE*, 103(3):318–331.
- Vallecorsa, Sofia, Carminati, Federico, and Khattak, Gulrukh (2019). 3D convolutional GAN for fast simulation. *EPJ Web Conf.*, 214:02010.
- Vasudev, R. (2019). Why distributed deep learning is going to gain paramount importance? why should you care about HPC for deep learning?
- Voulodimos, A., Doulamis, N., Doulamis, A., and Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018.
- Wang, X., Han, Y., Leung, V. C. M., Niyato, D., Yan, X., and Chen, X. (2020). Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 22(2):869–904.
- Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. (2018). Dynamic Graph CNN for Learning on Point Clouds.
- Whiteson, S. and Whiteson, D. (2009). Machine learning for event selection in high energy physics. *Engineering Applications of Artificial Intelligence*, 22(8):1203 – 1217.
- Wielgosz, M., Skoczeń, A., and Mertik, M. (2017). Using LSTM recurrent neural networks for monitoring the LHC superconducting magnets. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 867:40 – 50.
- Wu, C., Herranz, L., Liu, X., wang, y., van de Weijer, J., and Raducanu, B. (2018). Memory replay GANs: Learning to generate new categories without forgetting. In *Advances in Neural Information Processing Systems 31*, pages 5962–5972. Curran Associates, Inc.

- Xie, S., Girshick, R. B., Dollár, P., Tu, Z., and He, K. (2016). Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431.
- Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network.
- Yang, G. (2019). 3D Projection Scintillator Tracker in the DUNE Near Detector. *PoS, ICHEP2018*:868.
- Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. (2018). Hierarchical Graph Representation Learning with Differentiable Pooling. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 4800–4810. Curran Associates, Inc.
- You, Y., Zhang, Z., Hsieh, C., Demmel, J., and Keutzer, K. (2019). Fast deep neural network training on distributed systems and cloud TPUs. *IEEE Transactions on Parallel and Distributed Systems*, 30(11):2449–2462.
- Yousef, M., Hussain, K. F., and Mohammed, U. S. (2020). Accurate, data-efficient, unconstrained text recognition with convolutional neural networks. *Pattern Recognition*, 108:107482.
- Yu, S. (2019). Electron neutrino energy reconstruction in NOvA using CNN particle IDs.
- Zeiler, M. D. (2012). Adadelta: An adaptive learning rate method.
- Zhang, Z., Huang, L., Pauloski, J. G., and Foster, I. (2019). Aggregating local storage for scalable deep learning I/O. In *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)*, pages 69–75.
- Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2018). Graph neural networks: A review of methods and applications.
- Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2017). Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012.

Appendix A

Introduction to deep learning

A.1 Notation and definitions

The notation presented in this appendix is based on the notation used in the book *Deep Learning* [Goodfellow et al., 2016] and the *Coursera Deep Learning Specialization* [deeplearning.ai, 2015].

Numbers and arrays

a	A scalar (integer or real).
$\mathbf{a}$	A vector.
$\mathbf{A}$	A matrix.
$\mathbf{A}$	A tensor.
$\mathbf{a}$	A scalar random variable.
$\mathbf{a}$	A vector-valued random variable.
$\mathbf{A}$	A matrix-valued random variable.

Sets and graphs

$\mathcal{G}$	A graph.
$\mathcal{V}$	A set of vertices from a graph.
$\mathcal{E}$	A set of edges from a graph.

Indexing

a_i	Element i of vector $\mathbf{a}$, with indexing starting at 1.
$A_{i,j}$	Element i, j of matrix $\mathbf{A}$.
$\mathbf{a}_i$	Element i of the random vector $\mathbf{a}$.

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x .
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x .

Probability and information theory

$P(\mathbf{a})$	A probability distribution over a discrete variable.
$p(\mathbf{a})$	A probability distribution over a continuous variable, or over a variable whose type has not been specified.
$\mathbf{a} \sim P$	Random variable $\mathbf{a}$ has distribution P .
$\mathbb{E}_{\mathbf{x} \sim P}[f(\mathbf{x})]$ or $\mathbb{E}f(\mathbf{x})$	Expectation of $f(\mathbf{x})$ with respect to $P(\mathbf{x})$.

Functions

$f(\mathbf{x}; \boldsymbol{\theta})$	A function of $\mathbf{x}$ parametrised by $\boldsymbol{\theta}$. (Sometimes we write $f(\mathbf{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation).
$\log x$	Natural logarithm of x .
$\sigma(x)$	Logistic sigmoid, $\frac{1}{1 + e^{-x}}$

Datasets and Distributions

p_{data}	The data generating distribution.
$\hat{p}_{\text{data}}$	The empirical distribution defined by the training set.
$\mathbb{X}$	A set of training examples.
$x^{(i)}$ or $\mathbf{x}^{(i)}$	The i -th example (input) from a dataset.
$y^{(i)}$ or $\mathbf{y}^{(i)}$	The target associated with the i -th input example for supervised learning.
$\hat{y}^{(i)}$ or $\hat{\mathbf{y}}^{(i)}$	The prediction given for the i -th input example for supervised learning.

Definitions

Sample	a dataset.
Input/training example	one element of a dataset (e.g., one image is an input example in a convolutional neural network, whilst one audio file is an input example for a speech-recognition model).
Batch	a set of n input examples (also called mini-batch). The input examples in a batch are processed independently, in parallel. During training, a batch results in only one update to the model (one forward pass and one backward pass).
Iteration	one pass using a batch of input examples. Thus, the number of iterations consists of the number of forward/backward passes, each pass using <i>batch_size</i> number of examples.
Epoch	one forward pass and one backward pass of all the training examples. In other words, an epoch is one pass over the entire dataset, and it is used to separate training into distinct phases. For a dataset consisting of m training examples and a batch size of n , then it will take m/n iterations to complete one epoch.

A.2 Machine-learning concepts

Arthur Samuel defined the term *machine learning* in 1959 as “*the field of study that gives computers the ability to learn without being explicitly programmed*”. Tom Mitchell updated Arthur’s definition in 1998: “*A computer program is said to learn from experience E with respect to some task T and some performance measure P , if its performance on T , as measured by P , improves with experience E* ”. To summarise, machine learning (ML) is an interdisciplinary field within the mathematical and computational sciences that aims to use and develop algorithms that have the ability to learn from input data, being able to give accurate predictions on unseen data. There are three primary ML paradigms: supervised, unsupervised, and reinforcement learning.

A.2.1 Supervised learning

Supervised learning algorithms are those that try to associate every input with a certain output, given a set of examples $\mathbf{x}$ and outputs $\mathbf{y}$ (also known as targets, labels, or just tags) to learn from. In supervised learning, there is supposed to be a relationship between the input and the output and the goal of the algorithm is to learn this relationship. Supervised learning problems are categorised into *classification* and *regression* problems.

Classification problems

In classification problems, every input example x must be associated with a discrete output y . Thus, the predicted output $\hat{y}$ must be able to be represented in a discrete way, and the goal of the algorithm is to make $\hat{y}$ match the true label y . Examples of classification problems are: tagging photos of pets as being *cats*, or *dogs*, or guessing the flavour of a neutrino event. The former would be a *binary classification problem*, since for each input example there are only two possible labels that can be encoded as 0 or 1; the latter, on the other hand, would be a *multi-class classification problem*, since the number of possible labels is larger than two for each input example. There is a third set of classification problems, called *multi-label classification problems*, which might be seen as combination of the other two. In multi-label classification problems, each input example can have either zero or more labels associated.

Regression problems

In regression problems, every input example x must be associated with a continuous output y . Thus, the predicted output $\hat{y}$ must be able to be represented in a continuous way, and the goal of the algorithm is to make $\hat{y}$ as close as possible as the true label y .

Examples of regression problems are: predicting house prices based on house features or predicting the energy of a neutrino event.

A.2.2 Unsupervised learning

Unsupervised learning algorithms are those that try to approach problems with little or no idea what results should look like, given a set of examples $\mathbf{x}$ to learn from. The main difference between supervised and unsupervised learning problems is that, for the latter, there are no labels $\mathbf{y}$ available for training. Examples of unsupervised-learning problems are: identifying meaningful patterns in 2D data or splitting Amazon costumers into three groups based on their activity in the last month.

A.2.3 Reinforcement learning

In reinforcement learning algorithms, a software agent learns to achieve a goal by facing a game-like situation, getting either rewards or penalties depending on the actions taken. The agent plays a trial-and-error-based approach in order to solve the problem. Examples of reinforcement-learning problems are: training a robot to escape from a maze or coherent transport of quantum states.

A.3 Neural networks

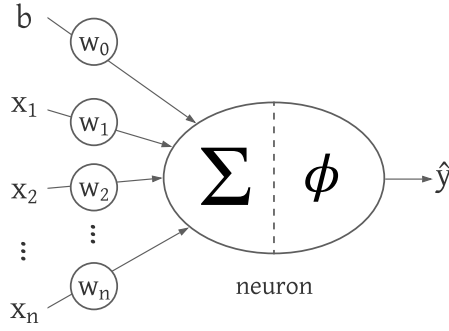
Artificial neural networks are algorithms inspired by the way the human brain works. Deep learning, for its part, is a family ML algorithms based on artificial neural networks [LeCun et al., 2015]. The first time neural activity was accounted as being computational was back in 1943 [McCulloch and Pitts, 1943], where simple neural networks were shown to be able to compute almost any kind of logic or arithmetic function.

The human brain is made up of around 86 billion neurons, and each neuron can have up to thousands of synapses¹, through which other neurons transmit electrical impulses. When this electrical impulse exceeds the threshold of the neuron, the neuron transmits the impulse through its axon². Artificial neurons try to recreate the behaviour of biological neurons. Among the mathematical models of neurons, one of the simplest is the perceptron [Rosenblatt, 1958]. Figure A.1 shows the structure of a perceptron. The

¹In biology, the transmission of the electrical nerve impulse from one cell to another is known as the synapse. Each neuron can connect - and be connected - to several neurons in such a way that networks of neurons can be created.

²The most common biological neurons have a cell body, many short prolongations called dendrites, which receive the nerve impulse and carry it to the body of the neuron, and a long prolongation called the axon, that transmits the impulse outward toward another neuron or an organ.

artificial neuron receives a vector of inputs $\mathbf{x}$ (which are outputs from other neurons or the actual input data) weighted by a weight vector $\mathbf{w}$. The perceptron is only activated if the weighted sum of all inputs exceeds a threshold. The output $\hat{y}$ can be interpreted as the transmission of the electrical impulse by the axon of the neuron. Mathematically, this behaviour can be represented by Equations A.1, A.2, and A.3.



$$\Sigma = w_0 b + \sum_{i=1}^n w_i x_i \quad (\text{A.1})$$

$$\hat{y} = \phi(\Sigma) \quad (\text{A.2})$$

Fig. A.1 Artificial neuron example. For a given neuron, the weighted sum of its input $\mathbf{x}$ (together with the bias input b) is passed to the threshold-based activation function ϕ in order to obtain the neuron output $\hat{y}$ (see Equations A.1, A.2, and A.3).

$$\hat{y} = \begin{cases} 1 & \text{if } \Sigma \geq \text{threshold} \\ 0 & \text{otherwise} \end{cases} \quad (\text{A.3})$$

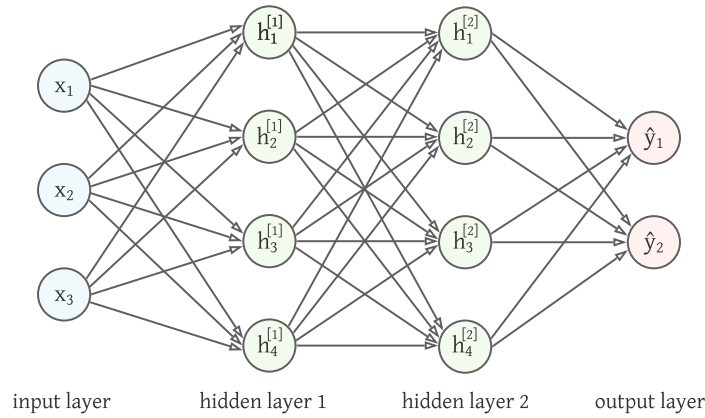


Fig. A.2 Example of a multi-layer perceptron: an input layer (a vector of length three), two hidden layers (also known as fully-connected or dense layers, consisting of four neurons each), and an output layer (composed of two neurons). All the neurons in one layer are connected to every neuron in the next layer.

Artificial neurons can be combined into networks of neurons and used to build multi-layer circuits (e.g., multi-layer perceptrons) that can be used to perform a number of functions and, more importantly, are capable of learning non-linear features from the input data [Grossberg, 1988; Hopfield, 1982]. An example of a multi-layer perceptron

(MLP), also called feedforward neural networks, is shown in Fig. A.2. To give an example: assume there is a mapping between some input $\mathbf{x}$ and some output y so that $y = f^*(\mathbf{x})$; the goal of an MLP would be to satisfy $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$ and result in the best function approximation by learning the best set of values for the network parameters $\boldsymbol{\theta}$ (also known as weights).

A.3.1 Forward and backward propagation

In this section, neural network training is described in the context of logistic regression. Logistic regression is a standard binary classification method, used to model the probability of a certain input to be either *true* or *false* [McCullagh and Nelder, 1989]. The neural network used is shown in Fig. A.3.

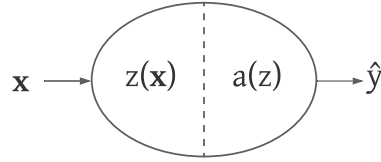


Fig. A.3 Simple neural network used for logistic regression. It consists of a single neuron. The workflow is the following: it accepts an input $\mathbf{x}$ that is passed to a linear equation z , followed by an activation function a ; it finally outputs the prediction $\hat{y}$.

where z is the following linear equation:

$$z(\mathbf{x}) = \mathbf{w}\mathbf{x} + b \quad (\text{A.4})$$

It is important to notice that $\mathbf{w}$ and b are the network parameters $\boldsymbol{\theta}$. The variable a is the sigmoid function σ , the activation function commonly chosen for both binary and multi-label classification problems:

$$a(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \quad (\text{A.5})$$

- If z is a large positive number, then $\sigma(z) \approx 1$.
- If z is a large negative number, then $\sigma(z) \approx 0$.
- If $z = 0$, then $\sigma(z) = 0.5$.

The goal of the algorithm is to make the prediction $\hat{y}$ as close as possible to the truth output y for each input example. To do so, the network has to be trained by tuning the network parameters $\boldsymbol{\theta}$. The initial values of network parameters $\boldsymbol{\theta}$ (in this example, $\mathbf{w}$ and b) are given by the initialisation method used (see Section A.3.2); if all weights

are initialised to zero, all the neurons become symmetric, meaning all the neurons learn the same function (in case of a neural network composed of more than one neuron). A neural network training can be divided into two steps: *forward propagation* and *backward propagation*.

Forward propagation

For each input example $\mathbf{x}^{(i)}$, the corresponding prediction $\hat{y}$ is calculated using the above equations altogether:

$$\hat{y}(\mathbf{x}; \boldsymbol{\theta}) = \sigma(\mathbf{w}\mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}\mathbf{x} + b)}} \quad (\text{A.6})$$

Once the prediction $\hat{y}$ has been calculated, one can measure the discrepancy between $\hat{y}$ and the true output y using a loss function (error function) L . For binary classification problems, the loss function used is the binary cross-entropy loss function:

$$L(y^{(i)}, \hat{y}^{(i)}) = -(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})) \quad (\text{A.7})$$

- If $y^{(i)} = 0$, then $L(y^{(i)}, \hat{y}^{(i)}) = -\log(1 - \hat{y}^{(i)})$, where $\hat{y}^{(i)}$ should be close to 0 in order to minimise L .
- If $y^{(i)} = 1$, then $L(y^{(i)}, \hat{y}^{(i)}) = -\log(\hat{y}^{(i)})$, where $\hat{y}^{(i)}$ should be close to 1 in order to minimise L .

The cost function J is the average of the loss function for the entire training set. The objective is to find the parameters $\mathbf{w}$ and b that minimise the overall cost function:

$$J(\boldsymbol{\theta}) = J(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^m L(y^{(i)}, \hat{y}^{(i)}) = -\frac{1}{m} \sum_{i=1}^m (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})) \quad (\text{A.8})$$

where m is the number of training examples.

Backward propagation

Once the cost J is calculated, the next step is to find the $\boldsymbol{\theta}$ parameters that minimise $J(\boldsymbol{\theta})$. Computing the partial derivatives of J with respect to the parameters $\boldsymbol{\theta}$ ($\mathbf{w}$ and b in the logistic regression example) can give us insights on how the J would behave if we increase or decrease the value of the $\mathbf{w}$ and b . The derivatives of J with respect to $\boldsymbol{\theta}$ are known as *gradients*. Once computed, the gradients are used to update the value of $\boldsymbol{\theta}$ as follows:

$$\theta \leftarrow \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}, \quad \forall \theta \in \theta \quad (\text{A.9})$$

Where α is the learning rate, the most critical hyper-parameter³ in ML models. The learning rate defines the size of the step taken while updating the network parameters θ . The larger the step, the faster the algorithm does its best to approach the global minimum, with the risk of “bouncing around” the minimum. On the other hand, the smaller the step, the slower the algorithm does its best to approach the global minimum while being likely to get stuck in a local minimum [Goodfellow et al., 2016; Smithing, 1999]. The backward propagation calculations are usually vectorised to be able to handle several input examples at the same time. In neural networks, the algorithm that iteratively computes the predictions (forward propagation), computes the derivatives (backward propagation), and updates the weights is known as *gradient descent*. Due to the large size of training datasets and in order to make the optimisation algorithm converge faster, *stochastic gradient descent* (SGD) is used instead of the traditional gradient descent. In SGD, the training is done iteratively, where only a small random portion (known as mini-batch) of the training dataset is used in each iteration. Neural networks tend to converge after computing several training epochs [Bottou et al., 2018].

A.3.2 Considerations

Activation functions

The example above shows how the sigmoid activation function is used in neural networks for logistic regression. For deeper networks, the sigmoid activation is the function commonly chosen as the activation function of the output layer for both binary and multi-label classification problems:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (\text{A.10})$$

For multi-class problems, the softmax activation function is used as the activation function of the output layer. Softmax [Bridle, 1990a,b] is a generalisation of the sigmoid activation function, and it forces all the values to sum up to 1:

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n_y} e^{z_j}}, \quad \forall i \in \{1, \dots, n_y\} \quad (\text{A.11})$$

³Hyper-parameters: parameters used to control the learning process, distinct from the network weights.

For the hidden layers, the hyperbolic tangent ($\tanh$) activation function tends to report better results than the sigmoid. It has a range between -1 and 1:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (\text{A.12})$$

However, the most common activation function for the hidden layers is the rectified-linear unit (ReLU) [Nair and Hinton, 2010]:

$$\text{ReLU}(z) = \max(0, z) \quad (\text{A.13})$$

It has been shown that supervised training of deep neural networks is much faster if the hidden layers are composed of ReLU activation functions [Glorot et al., 2011]. Other ReLU variations have been shown to report good performance for image classification tasks, among others [Xu et al., 2015]. An example of the above is the Leaky ReLU (also known as LReLU or LReL) activation function [Maas et al., 2013]:

$$\text{LReLU}(z) = \begin{cases} z & \text{if } z \geq 0 \\ \frac{z}{a} & \text{otherwise} \end{cases}, \quad a \in \mathbb{R}_{>1} \quad (\text{A.14})$$

Initialisation methods

Common initialisation methods are: *Xavier* initialisation [Glorot and Bengio, 2010] (better for $\tanh$ and sigmoid/softmax activation functions), where weights in the network are set from a uniform distribution with zero mean and a specific variance:

$$\text{Var}^2 = \frac{1}{N} \quad (\text{A.15})$$

and *He* initialisation [He et al., 2015b] (better for ReLU activation functions):

$$\text{Var}^2 = \frac{2}{N} \quad (\text{A.16})$$

where N is the number of input neurons to a particular layer.

Cost functions

The logistic regression example shows how the usage of the binary cross-entropy cost function for binary classification problems:

$$J = -\frac{1}{m} \sum_{i=1}^m (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})) \quad (\text{A.17})$$

Binary cross-entropy is just a special case of categorical cross-entropy, which is the cost function used for multi-class problems:

$$J = -\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^c y_j^{(i)} \log \hat{y}_j^{(i)} \quad (\text{A.18})$$

Finally, the cost function used for multi-task learning (multi-label problems) is the negative log-likelihood:

$$J = -\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^l (y_j^{(i)} \log(\hat{y}_j^{(i)}) + (1 - y_j^{(i)}) \log(1 - \hat{y}_j^{(i)})) \quad (\text{A.19})$$

where m is the number of training examples and l is the number of labels.

Optimisers

Variations of the standard SGD, such as AdaGrad [Duchi et al., 2011], ADADELTA [Zeiler, 2012], Adam optimiser [Kingma and Ba, 2014], or RMSprop [Tieleman and Hinton, 2012], among others [Ruder, 2017], have reported excellent convergence results [Sun, 2019]. One optimiser is not better than another by default, since the performance of an optimiser depends mostly on the setting. As an example, Tab. A.1 shows the performance of different optimisers on a public dataset.

Optimiser	Convergence time (hrs)	Accuracy	Loss
AdaGrad	2.625	0.404	2.048
ADADELTA	2.750	0.272	2.220
Adam	2.625	0.720	0.920
RMSprop	2.500	0.458	1.648
SGD	2.375	0.585	1.278

Table A.1 Training results for the Fashion-MNIST dataset using different gradient-descent optimisers. Table adapted from Ref. [Dogo et al., 2018].

Deep-learning frameworks

There are many deep-learning frameworks available, such as TensorFlow [Abadi et al., 2015, 2016], PyTorch [Paszke et al., 2019], Caffe [Jia et al., 2014], or Theano [Theano Development Team, 2016]. They offer simplified methods for designing and training deep neural networks on both CPU and GPU. Keras [Chollet et al., 2015] is a high-level API that runs on top of TensorFlow or Theano, ideal for fast development and prototyping. From TensorFlow v2.0, Keras is now available as a package within the framework.

A.3.3 Regularisation

Training neural networks is usually not an easy task. A neural network learns from some training data until it is ready to make predictions on unseen data. Figure A.4 shows three different ways a model can fit some training data. In the scenario in Fig. A.4a, the model is probably too simple to fit the training data, whereas the model in Fig. A.4c overfits the training data, meaning that it will struggle when making predictions on unseen data. The model in Fig. A.4b seems the most sensible of the three.

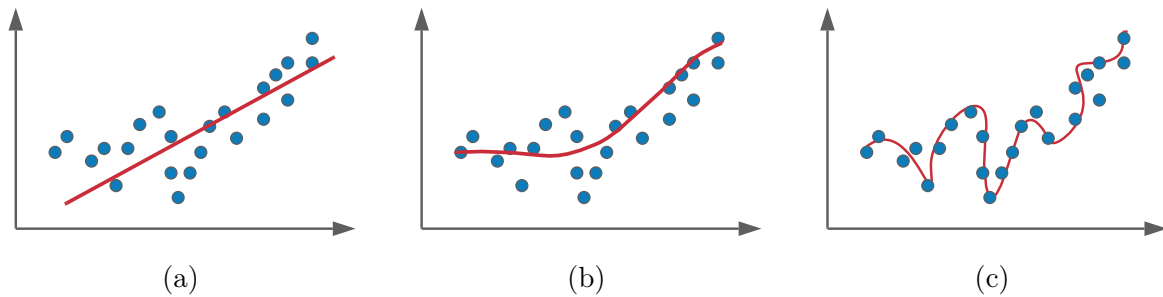


Fig. A.4 A comparison of three different ways a model can fit the training data. a) underfitting, where the model is too simple to accurately learn the behaviour of the training data; b) good compromise; c) overfitting (also known as overtraining), where the model is too complex to generalise properly on unseen data.

It is typically straightforward to solve problems due to underfitting. Common solutions are: increasing the training time, getting more training examples, adding more complexity to the model (i.e., increase the capacity of the model: more layers, more neurons per layer, etc.). However, overtraining could be a more serious issue. Typically, the datasets used for training supervised-learning models are divided into three groups:

- Training set: the model learns from this set only. It must be the largest set, and ideally, its data should come from a dataset independent of the validation and test sets.
- Validation set: it is used to evaluate the model (usually after every training epoch) and to fine-tune the model hyperparameters (e.g., the learning rate, or the model architecture itself). It is also known as the development set or just dev set. Ideally, its data should come from a dataset different to the one used for the training set.
- Test set: it is used to evaluate the model at the end of the training, only once the model is fully trained. Ideally, its data must come from a dataset different to the one used for the training set. Sometimes, the test set is not needed, indicating that only training and validation sets are available.

Figure A.5 shows a simplified example of the training and test errors compared to the complexity of the neural network used, showing that models tend to overtrain if they are too complex for the training data.

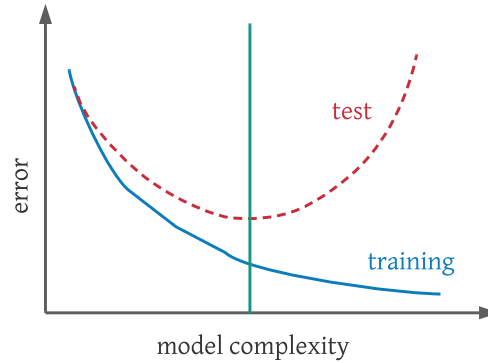


Fig. A.5 Training and test errors depending on the complexity of the model. For simpler models, the error is large. When we increase the complexity of the model, both the training and test errors decrease, meaning the model is learning from the training data and generalising on examples unseen during the training. However, if the model is too complex, it can overtrain. Overtraining means that the network is too powerful for the current problem, hence learning the training data “by heart” and not being able to generalise on the test set. The green vertical line represents the best model complexity.

One could think that simplifying the model should be sufficient to fix the overtraining issue. However, simplifying the network means reducing the model’s capacity, which is usually not ideal. If the training set is small, adding more training examples could also help to avoid overtraining, since maybe the network was learning all the training examples by heart. Increasing the size of the dataset is not always an option (it might be expensive to gather more data) or it can be not enough (the model still reports signs of overtraining); in such cases, *regularisation* is needed. Regularisation techniques make small modifications to the architecture, training phase, or even the input data in order to make the model generalise better (even on unseen data). Some common generalisation techniques are:

- L^1 and L^2 regularisation: they are known as weight decay and are standard regularisation methods, and they prevent overtraining by imposing a penalty on the model weights [Ng, 2004]. In order to calculate the new loss function L' , L^1 regularisation adds the sum of absolute values of the network parameters $\mathbf{w}$ to the original loss function L :

$$L' = L + \lambda \sum_{i=1}^N |w_i| \quad (\text{A.20})$$

while L^2 adds the sum of the squared values of the weights $\mathbf{w}$ to the loss:

$$L' = L + \lambda \sum_{i=1}^N w_i^2 \quad (\text{A.21})$$

where λ is the regularisation parameter and it should be > 0 .

- **Batch normalisation:** is a regularisation technique that standardises the inputs to a layer for each input batch of data, stabilising the learning process [Ioffe and Szegedy, 2015].
- **Dropout:** is a computationally inexpensive regularisation method that randomly ignores a number of non-output units from certain layers, introducing noise to the learning process and forcing the model to become robust to deal with the noise [Srivastava et al., 2014].
- **Early stopping:** training complex models (networks with enough capacity for learning the desired task) too long could cause the training error decrease steadily over time, but make the validation (and test) error rise at some point, similar to the behaviour shown in Fig. A.5 regarding model complexity. Early stopping [Prechelt, 1996] deals with this problem by monitoring the validation results periodically, and finishing the training as soon as it notices that the validation error does not decrease any more.
- **Data augmentation:** an easy way of increasing the size of the training dataset is to use data augmentation to create more input examples artificially [Shorten and Khoshgoftaar, 2019]. These new fake input examples are generated by applying some manipulations (e.g., mirroring, rotation) to the actual training examples.

A.3.4 Convolutional neural networks

Convolutional neural network (CNN) [LeCun et al., 1989, 1998] algorithms that operate on images have been very successful in a number of tasks [Bengio et al., 2013; Gu et al., 2018; Howard et al., 2017; LeCun et al., 2010]. The main feature of CNNs is that they apply a series of filters (using convolutions, hence the name of the CNN), usually followed by spatial pooling, applied in sequence to extract increasingly powerful and abstract features that allow the CNN to classify the images [Krizhevsky et al., 2012]. Each of the filters - also known as kernels - consists of a set of values that are learnt by the CNN through the training process. CNNs are typically deep neural networks that consist of many convolutional layers, with the output from one convolutional layer forming the input to the next. The last layers of a CNN are usually fully connected layers, where the output layer is followed by a sigmoid or softmax activation function. Current CNNs use

global average pooling (GAP) to replace fully connected layers in CNNs, which works in practice as a regularisation technique since with GAP the output layer of the model is fed directly with the average of each feature map from the last convolutional layer of the model [Lin et al., 2013].

Image classification

CNNs are fundamental for computer vision, especially since the introduction of the AlexNet, a CNN that outperformed previous techniques [Krizhevsky et al., 2012]. Some successful state-of-the-art CNNs are: VGG-16 and VGG-19 [Simonyan and Zisserman, 2014], SqueezeNet [Iandola et al., 2016], GoogleNet (also called Inception-v1, ILSVRC 2014 winner) [Szegedy et al., 2014], ResNet (ILSVRC 2015 winner) [He et al., 2015a, 2016], Inception-v2 and Inception-v3 [Szegedy et al., 2015], Inception-v4 and Inception-ResNet-v2 (unofficial new record on the 2015 ILSVRC) [Szegedy et al., 2016], ResNeXt (ILSVRC 2016 winner) [Xie et al., 2016], Xception [Chollet, 2016], Squeeze-and-Excitation Networks (ILSVRC 2017 winner) [Hu et al., 2017], NASNet [Zoph et al., 2017]. Figure A.6 shows how some of the CNNs above perform on the ImageNet dataset [Deng et al., 2009] (the dataset used for the ILSVRC challenge).

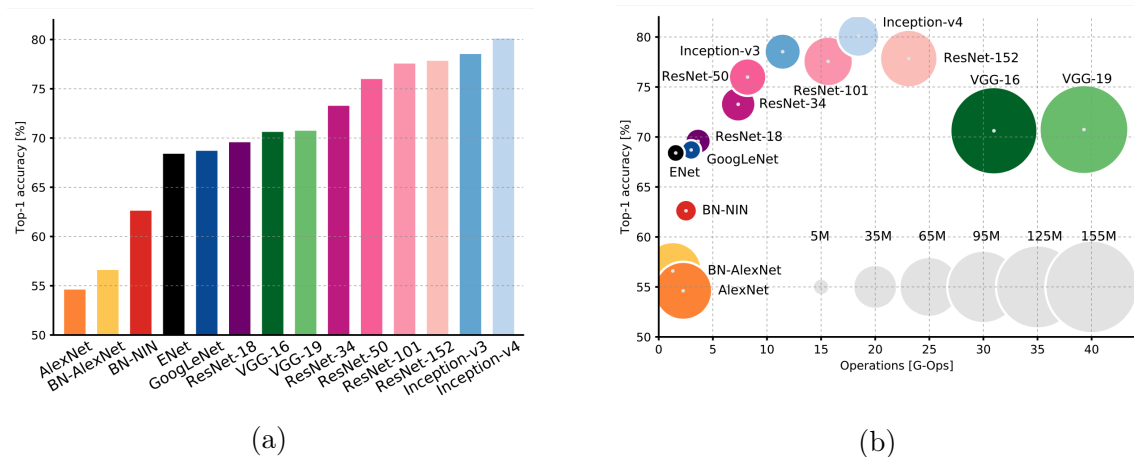


Fig. A.6 State-of-the-art CNN performance on ImageNet dataset comparison. a) top-1 validation accuracy; b) top-1 validation accuracy vs amount of operations required for a forward pass, where the radiuses of the circles are proportional to the number of parameters of the network. Source: Ref. [Canziani et al., 2016].

Semantic segmentation

CNNs are also popular for semantic segmentation (SS). The idea behind SS is to classify each pixel in an image in order to recognise what is in the image. In SS [Shelhamer et al.,

2017], a CNN outputs a map, usually with the same or proportional dimensions with respect to the input image (usually of arbitrary size), in which each pixel is categorised in a way that groups of neighbouring pixels with the same predicted class can form objects and shapes (see Fig. A.7). Since SS is a classification class, the number of labels is limited and predefined by the user. It is used in many fields, such as robot vision [Milioto et al., 2018] or autonomous driving [Teichmann et al., 2018].

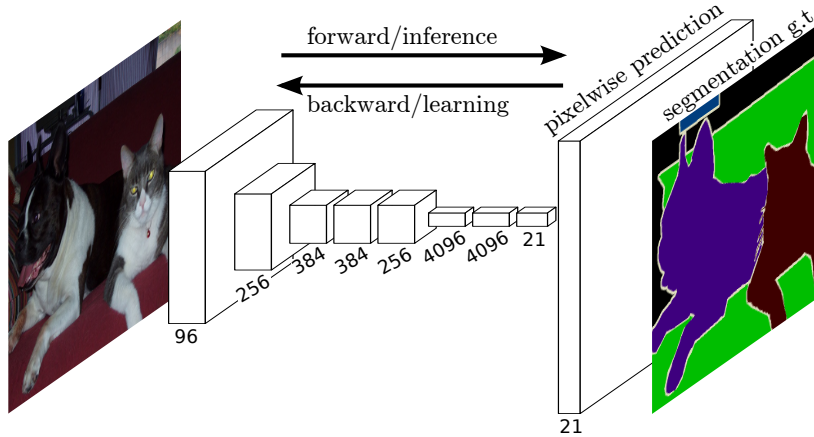


Fig. A.7 Semantic-segmentation example. The input image (left) is fed into the semantic-segmentation network. The output map (right) depicts the recognised objects: dog (purple), cat (dark red), sofa (green), window (blue), and background (black). Source: Ref. [Shelhamer et al., 2017].

Image-classification CNNs, such as GoogLeNet or ResNet, can be used as the core of SS networks [Lateef and Ruichek, 2019]. CNNs that perform SS tasks are known as fully convolutional networks (FCNs) [Shelhamer et al., 2017], since fully-connected layers are not present in their construction.

A.3.5 Graph neural networks

In computer science, a graph $\mathcal{G}$ is a data structure that represents a mathematical concept consisting of vertices $\mathcal{V}$ (also known as nodes) and edges $\mathcal{E}$:

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}). \quad (\text{A.22})$$

A graph can be directed, where each edge has a starting and an ending vertex that define a direction, or undirected, where the edge simply connects two nodes without inducing a sense of direction. In the case of the work presented in Chapter 4, we use an undirected graph, since we are only interested in the spatial connections between vertices. The deep-learning algorithm that operates on graphs is the graph neural

network (GNN) [Sperduti and Starita, 1997; Zhou et al., 2018]. GNNs are used in many different fields [Duvenaud et al., 2015; Shen et al., 2018] and can be applied for graph classification [Wang et al., 2018; Ying et al., 2018] or node classification [Hamilton et al., 2017; Kipf and Welling, 2016; Perozzi et al., 2014].

GraphSAGE

GraphSAGE [Hamilton et al., 2017] is a technique that leverages the features of graph nodes $\mathcal{V}$ - which can range from physical information to text attributes - to generate efficient representations on previously unseen samples by learning aggregator functions from training nodes. These aggregators can be simple functions (e.g., mean or maximum) or more complex ones, such as Long short-term memory (LSTM) cells [Hochreiter and Schmidhuber, 1997], and must be functions that take an arbitrary number of inputs without any given order. The model learns not only K aggregator functions that combine information from neighbouring nodes but also a set of weight matrices $\mathbf{W}^k, \forall k \in \{1, \dots, K\}$, which are used to propagate information through the K layers of the model and combines local information of the node with the aggregator information of its neighbours into an encoding vector (see Algorithm A.1).

Algorithm A.1: GraphSAGE embedding generation (i.e., forward propagation) algorithm. Source: Ref. [Hamilton et al., 2017].

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K ; weight matrices $\mathbf{W}^k, \forall k \in \{1, \dots, K\}$; non-linearity σ ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, \dots, K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

```

1  $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;
2 for  $k = 1 \dots K$  do
3   for  $v \in \mathcal{V}$  do
4      $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$ ;
5      $\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k))$ 
6   end
7    $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$ 
8 end
9  $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$ 
```

The number of aggregator functions is also used to define the depth of the model, meaning that a GraphSAGE model has a depth of K . In each layer of the aggregator information, a new representation of the node v is computed, denoted by $\mathbf{h}_v^k$ (with $\mathbf{h}_v^0$ being the initial node features $\mathbf{x}_v$). Once trained, it can produce the embedding of a new node given its input features and neighbourhood, in the form of the vector of the last layer $\mathbf{h}_v^K$; this embedding is then used as the input of a multilayer perceptron

(MLP) [Rosenblatt, 1958] that is responsible for predicting the label. Since GraphSAGE learns from node features, it allows developers to decide which information to use for each node. This means that the model predicts the label for each node based on the features of the target node as well as the features of its neighbours.

A.3.6 Generative adversarial networks

A generative adversarial network (GAN) [Goodfellow et al., 2014] is a deep-learning algorithm that aims to produce fake images indistinguishable from true images. To do so, GANs are trained following an adversarial process, where two models are trained simultaneously (see Fig. A.8). The goal is to learn a generator distribution $p_G(\mathbf{x})$ that matches the true data distribution $p_{data}(\mathbf{x})$ (i.e., the distribution of the training set). The model learns a generator network G that generates samples from the generator distribution p_G by transforming random noise variables $p_{noise}(\mathbf{z})$ into samples $G(\mathbf{z})$; To achieve this, D and G play the following two-player minimax game [Goodfellow et al., 2014] with value function $V_1(G, D)^4$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))] \quad (\text{A.23})$$

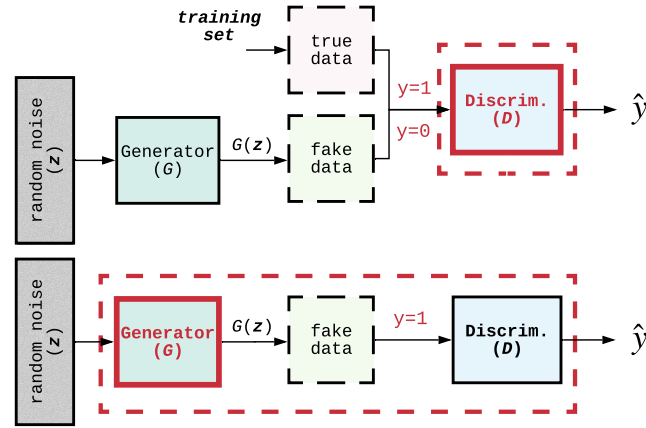


Fig. A.8 GAN training. The discriminator (Discrim.) learns to distinguish between true and fake images; the generator learns to make fake images from some random noise in such a way that the discriminator thinks they are true images. The red dashed lines enclose the neural network that is trained in each step and the inner red boxes correspond to the trainable modules within the neural network (thus, the weights of the other modules, if any, are frozen).

⁴“The expectation $\mathbb{E}$, or expected value, of some function $f(x)$ with respect to a probability distribution $p(x)$ is the average, or mean value, that f takes on when x is drawn from p .” [Goodfellow et al., 2016]

Current state-of-the-art GANs are able to produce high-quality fake images that no human can notice they are not real photos, as shown in Fig. A.9.



Fig. A.9 Examples of images generated with a GAN from some random noise. Source: Ref. [Brock et al., 2019].

A.4 Parallel and distributed training

Due to the Single Instruction, Multiple Data (SIMD) anatomy (from Flynn’s taxonomy) of deep-learning models - most of the training computation is based on operations over multiple input examples at the same time, i.e. matrix multiplications, processing units that allow vectorized operations are commonly the best choice to run such computationally-demanding trainings. Traditionally, such processing units were standard CPUs with vector instructions (e.g., Intel AVX2, AVX512F, and FMA) and GPUs. Nowadays, custom-designed ASICs (Application-Specific Integrated Circuits) offer a better performance-per-watt solution and are available as extra cores in CPU/GPU (e.g., NVIDIA Tensor Cores), or even in special AI accelerator devices (e.g., Google’s TPUs) [Jouppi, 2020]. There are mainly two different ways of distributing neural network computation across multiple workers:

A.4.1 Model parallelism

Different parts of the model run on different workers, processing the input data altogether. This method of parallelism is very specific since the model needs to have a parallel architecture (e.g., a model that is divided into different branches), so not every model can benefit from it.

A.4.2 Data parallelism

The entire model gets replicated on multiple workers (that run on a single or various nodes). In each iteration, each replica of the model processes a different batch of input data, and then they merge their results before performing the backpropagation (see Fig. A.10).

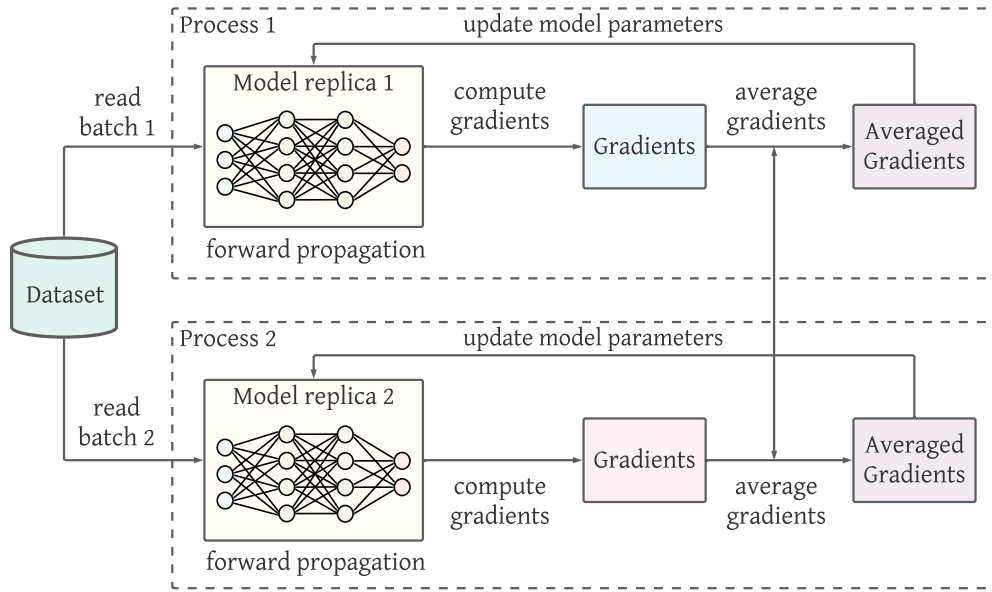


Fig. A.10 Data parallelism graphical example. The model is replicated on two different workers. Each model replica performs the forward propagation on an independent batch of data and computes the gradients. Then, the workers communicate to average their gradients so that the updates to the network parameters are consistent in each replica.

On top of the above, the distributed training can be *synchronous* or *asynchronous*. The workflow shown in Fig. A.10 could be interpreted as an example of synchronous distributed training, since there is a barrier until the gradients are averaged between the two workers. The main issue of synchronous training is that the processing units are idle until the gradients have synchronised. In asynchronous distributed training, each worker communicates with some *parameter servers* independently of the other workers; the parameter servers average the gradients received, send the updated parameters to all workers afterwards [Chilimbi et al., 2014; Dean et al., 2012]. Training in an asynchronous way would speed things up (workers can process the next batches of data during gradient synchronisation), with the disadvantage of losing training accuracy [Chen et al., 2017; Jin et al., 2016]. Since the access to the training data is random in SGD, random file accesses can easily saturate shared filesystems in distributed training environments [Zhang et al., 2019]. Deep-learning frameworks such as TensorFlow [Abadi et al., 2015, 2016]

or PyTorch [Paszke et al., 2019] include APIs to distribute the training across multiple nodes.

A.4.3 Horovod

Scaling the computation of neural networks can significantly reduce execution time. However, in practice, the following complications usually occur:

- Distributed models tend to be unable to take full advantage of the hardware resources available due to inter-GPU communication (averaging gradients). In fact, this can cause significant overhead in terms of execution time.
- Depending on the training API, the user often needs to make major changes to their source code to distribute the computation.
- The distribution of the training computation is usually focused on GPU, and not on CPU. Besides, all developers have multiple GPUs to run their deep-learning algorithms in a distributed way.

Horovod [Sergeev and Balso, 2018] it is an open-source distributed training framework for various deep-learning frameworks (i.e., TensorFlow, Keras, PyTorch, and MXNet) that handles the above issues. It solves the inter-worker communication by applying a custom ring reduction, it only requires a few extra lines of code from the user, and it allows to distribute the computation across multiple CPUs using MPI. Results in Ref. [Sergeev and Balso, 2018] show that Horovod drastically outperforms standard distributed TensorFlow in terms of images processed per second.

A.4.4 Deployment of deep-learning models on HPC environments

Deep-learning workloads are growing faster, and multiple organisations are using deep learning for solving complex problems, where training time is critical. Distributed and parallel processing is a necessary foundation, where the training time can be reduced by a factor of up to 121 [Vasudev, 2019] in some cases. Three main elements could be combined to handle distributed execution:

- A high-performance computing (HPC) cluster, with a batch queuing system to execute jobs.
- A message-passing library that implements the MPI standard such as Open MPI or MPICH. These implementations distribute the workload and take care of scattering and gathering the results across the nodes.

- A deep-learning library that supports multi-node jobs such as Horovod.

However, HPC systems have limitations. There are not usually on-demand systems, in a sense that the resources are shared among multiple users, and queue systems might need several days to start a particular job. Besides, the software stack is pre-installed for all users, and a tailored software stack is usually not supported (due to security and reliability restrictions). Finally, HPC systems have limited and expensive scalability.

To provide a much wider availability, some well-known cloud providers offer on-demand nodes [Amazon, 2020], even with AI accelerators such as GPUs or Google TPUs. Nodes can be used without waiting in a batch queue, and the pre-configured learning software stack is installed on particular nodes for deep-learning workloads [Amazon, 2020]. The software stack might include MPI alternatives such as Gloo (or mpi4py) for collective communications. Nowadays, cloud-based solutions have some challenges too. For instance, moving large amounts of data might be expensive in a public cloud. Moreover, working with sensitive data (e.g., medical data) should be avoided in public cloud systems. Deep-learning libraries are evolving very quickly, resulting in the need for frequent modifications to stay updated, causing public clouds to offer limited updated software stack combinations (e.g., undesired or outdated software versions).

Appendix B

Neutrino physics

Already introduced in Chapter 1, neutrinos are light fundamental particles, and are the second most abundant particles in the Universe, after photons. Neutrinos are leptons and they appear as three different generations in the Standard Model of particle physics (see Fig. B.1). They have no electric charge, a very tiny mass, and can interact only via the weak force. At the time of writing, the community have discovered three types of neutrinos, known as flavours: electron neutrino (ν_e), muon neutrino (ν_μ), and tau neutrino (ν_τ). Neutrinos can change from one flavour to another, meaning that a neutrino generated with a specific flavour can later be measured to have a different flavour; this phenomenon is known as *neutrino oscillations*.

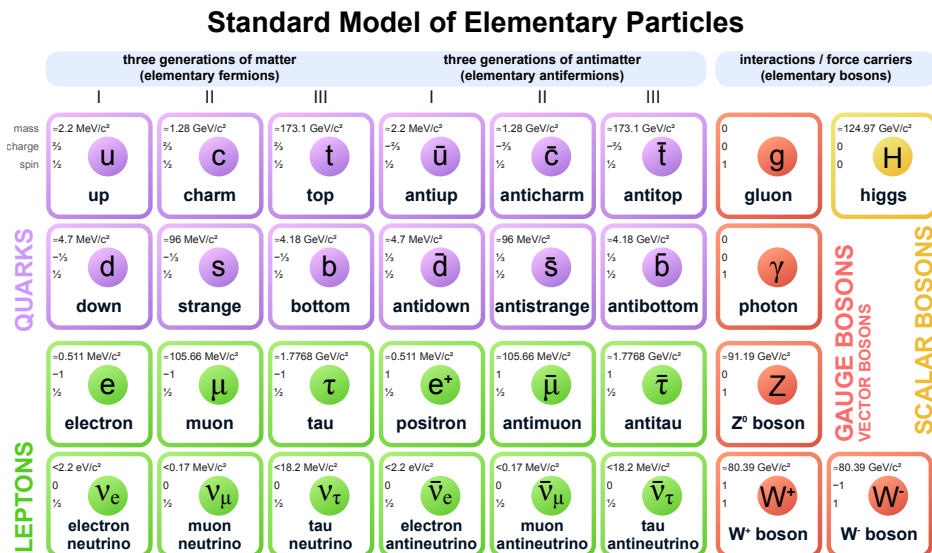


Fig. B.1 Overview of the Standard Model particles. Source: https://en.wikipedia.org/wiki/Particle_physics.

Neutrino interactions with matter can be classified as charge current (CC) or neutral current (NC) processes. In neutrino oscillation experiments, CC interactions typically form the signal. In contrast, NC interactions are considered background since it is not possible to identify the neutrino flavour in NC interactions, so they can not be used for studying the flavour oscillations. Among the different types of CC interactions, we can find: quasi-elastic (CC QE), resonance (CC Res), and deep inelastic (CC DIS) interactions.

B.1 Neutrino oscillations

Neutrino oscillations are a quantum mechanical phenomenon in which a neutrino changes its flavour as it travels. The fact that neutrinos oscillate demonstrates that they have mass, but at the same time, it means that the mass eigenstates can not be identical to the weak flavour eigenstates. For the three neutrinos, ν_e , ν_μ , ν_τ , the unitary mixing matrix U relates the flavour eigenstates, $|\nu_\alpha\rangle$, to the mass eigenstates, $|\nu_i\rangle$ [Kayser, 1981]:

$$|\nu_\alpha\rangle = \sum_{i=1}^3 U_{\alpha i} |\nu_i\rangle; \quad \begin{pmatrix} \nu_e \\ \nu_\mu \\ \nu_\tau \end{pmatrix} = \begin{pmatrix} U_{e1} & U_{e2} & U_{e3} \\ U_{\mu 1} & U_{\mu 2} & U_{\mu 3} \\ U_{\tau 1} & U_{\tau 2} & U_{\tau 3} \end{pmatrix} \begin{pmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{pmatrix} \quad (\text{B.1})$$

where $U_{\alpha i}$ is known as the Pontecorvo–Maki–Nakagawa–Sakata (PMNS) matrix [Maki et al., 1962; Pontecorvo, 1960], analogous to the Cabibbo–Kobayashi–Maskawa (CKM) matrix known from the quark sector [Chau and Keung, 1984]. The PMNS matrix is commonly expressed as the product of three matrices containing the three mixing angles θ_{12} , θ_{23} , θ_{13} , and one CP -violating phase δ_{CP} :

$$U = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_{23} & s_{23} \\ 0 & -s_{23} & c_{23} \end{pmatrix} \begin{pmatrix} c_{13} & 0 & s_{13}e^{-i\delta_{CP}} \\ 0 & 1 & 0 \\ -s_{13}e^{i\delta_{CP}} & 0 & c_{13} \end{pmatrix} \begin{pmatrix} c_{12} & s_{12} & 0 \\ -s_{12} & c_{12} & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (\text{B.2})$$

where $c_{ij} = \cos \theta_{ij}$ and $s_{ij} = \sin \theta_{ij}$. The left and right matrices describe the atmospheric ($\nu_\mu \rightarrow \nu_\tau$ oscillations) and solar ($\nu_e \rightarrow \nu_\mu$ and $\nu_e \rightarrow \nu_\tau$ oscillations) sectors, respectively, while the central matrix controls the oscillations of $\nu_\mu \rightarrow \nu_e$ at the atmospheric mass scale. U can therefore be written as:

$$U = \begin{pmatrix} c_{12}c_{13} & s_{12}c_{13} & s_{13}e^{-i\delta_{CP}} \\ -s_{12}c_{23} - c_{12}s_{23}s_{13}e^{-i\delta_{CP}} & c_{12}c_{23} - s_{12}s_{13}s_{23}e^{-i\delta_{CP}} & c_{13}s_{23} \\ s_{12}s_{23} - c_{12}s_{13}c_{23}e^{-i\delta_{CP}} & -c_{12}s_{23} - s_{12}s_{13}c_{23}e^{-i\delta_{CP}} & c_{13}c_{23} \end{pmatrix} \quad (\text{B.3})$$

The probability for a neutrino with flavour α and energy E to be measured as a neutrino of flavour β after travelling in vacuum a distance L is given by:

$$P(\nu_\alpha \rightarrow \nu_\beta) = \delta_{\alpha\beta} - 4 \sum_{i>j} \Re(U^{\beta i} U_{\alpha i} U_{\beta j} U^{\alpha j}) \sin^2 \left(\Delta m_{ij}^2 \frac{L}{4E} \right) + 2 \sum_{i>j} \Im(U^{\beta i} U_{\alpha i} U_{\beta j} U^{\alpha j}) \sin \left(\Delta m_{ij}^2 \frac{L}{2E} \right). \quad (\text{B.4})$$

where $\Re$ and $\Im$ are the real and imaginary parts, respectively, and $\Delta m_{ij}^2 = m_i^2 - m_j^2$ indicates the mass-squared splittings of two mass eigenstates i and j . When $\alpha = \beta$, the equation simplifies (the imaginary term goes to zero) and it is called the survival probability:

$$P(\nu_\alpha \rightarrow \nu_\alpha) = 1 - 4 \sum_{i>j} |U_{\alpha i}|^2 |U^{\alpha j}|^2 \sin^2 \left(\Delta m_{ij}^2 \frac{L}{4E} \right) \quad (\text{B.5})$$

Open questions

Many of the parameters describing neutrino oscillations have been measured, but some open questions remain:

- Do neutrinos and antineutrinos oscillate in the same way? A measurement of $\sin \delta_{CP} \neq 0$ would indicate that they do not, and thus constitute a discovery of CP -violation.
- Is the third mass state, ν_3 , the heaviest or lightest of the mass states? These orderings are referred to as the *normal hierarchy* and *inverted hierarchy*, respectively.
- There is an ambiguity between values of θ_{23} in the first ($<45^\circ$) or second octant ($>45^\circ$). Which one is correct?

The next generation of neutrino oscillation experiments aim to answer these questions.

B.2 Neutrino oscillation experiments

Over the last twenty years neutrino oscillations [Maki et al., 1962; Pontecorvo, 1968] have become well-established [Abe et al., 2020; Acero et al., 2019; Adamson et al., 2014; Aharmim et al., 2005; Ahn et al., 2006; An et al., 2012; Araki et al., 2005; Fukuda et al., 1998a] and the field is moving into the precision measurement era. The PMNS [Maki et al., 1962; Pontecorvo, 1968] neutrino oscillation formalism describes the observed data with six fundamental parameters. These are the three mixing angles and CP -violating phase, δ_{CP} , from the PMNS matrix and two mass-squared splittings (differences between the squared masses of the neutrino mass states). If $\sin(\delta_{CP})$ is non-zero then the vacuum oscillation probabilities of neutrinos and antineutrinos will be different.

Long-baseline neutrino oscillation experiments use two detectors to characterise a beam of (anti)neutrinos: a near detector, located a few hundred metres away from the target that measures the original beam composition, and a far detector, located several hundred kilometres away, that allows for the determination of the beam composition after neutrino flavour oscillations. The energy of these beam neutrinos ranges from a few hundred MeV up to several GeV. Charged particles can be produced in neutrino interactions, and the energy that they deposit as they traverse the detector can be used to reconstruct the events. In general, the larger the energy transferred from the neutrino to the nucleus, the larger the number of particles and particle types produced in the final state. Oscillation probabilities are inferred from comparison of the observed neutrino spectra at the near and far detectors which are used to constrain values of the neutrino oscillation parameters.

Modelling nuclear interactions in the target nuclei is highly complex, particularly for high-energy transfers where the hadronic component of the interaction is more important. As a result, current long-baseline neutrino oscillation experiments mostly analyse interactions with low particle multiplicity. This situation, however, is expected to change in the coming years. On the one hand, the statistical and systematic uncertainties of current experiments have decreased significantly over recent years such that neutrino-nucleus modelling is becoming a dominant source of uncertainty [Abe et al., 2020; Acero et al., 2018]. On the other hand, future experiments like DUNE [Abi et al., 2020a] will use a broad-band energy neutrino beam, expecting a significant fraction of the neutrino interactions to have a high energy transfer to the nucleus.

As a result, in recent years, the neutrino physics community has turned its attention to measuring neutrino-nucleus interaction cross-sections for different ranges of energies and target materials [Tanabashi et al., 2018] as a way to constrain the oscillation uncertainties while providing new measurements to further develop the interaction models. In parallel, a new generation of neutrino detectors are under development that aim to resolve and

reliably identify short particle tracks even in very complex interactions. To achieve this, two main detector technologies stand out: one is based on liquid-argon time projection chambers (LArTPCs) [Rubbia, 1977] and the other is based on finely segmented plastic scintillators with three readout views [Blondel et al., 2018] that will form part of the near detectors for T2K [Abe et al., 2019] and, possibly, DUNE [Yang, 2019].

B.3 The DUNE experiment

DUNE [Abi et al., 2020a] is a next-generation neutrino oscillation experiment with a primary scientific goal of making precise measurements of the parameters governing long-baseline neutrino oscillation. A particular priority is the observation of CP -violation in the neutrino sector. In DUNE, a muon neutrino (ν_μ)- or muon antineutrino ($\bar{\nu}_\mu$)-dominated beam will be produced by the Long-Baseline Neutrino Facility (LBNF) beamline and characterised by a near detector (ND) at Fermilab before the neutrinos travel 1285 km to the Sanford Underground Research Facility (SURF), as illustrated in Fig. B.2. The neutrino and antineutrino beams are produced using opposite current in the magnetic focusing horns, hence “forward horn current” (FHC) and “reverse horn current” (RHC) refer to the neutrino and antineutrino beam modes, respectively. The far detector (FD) will consist of four 10 kt (fiducial) LArTPC detectors.

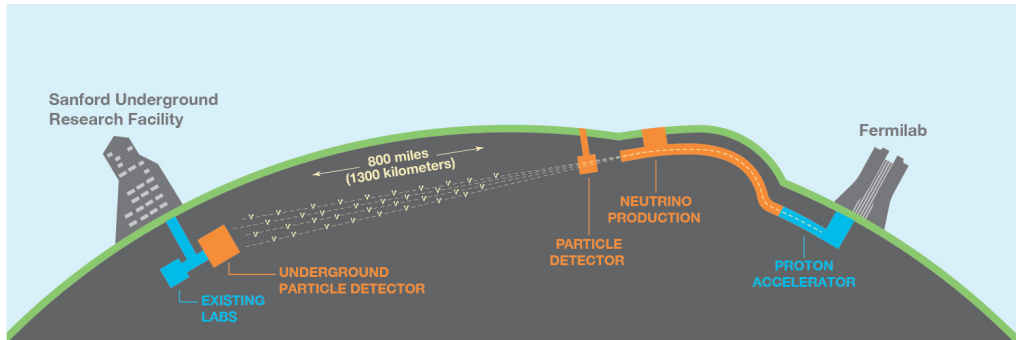


Fig. B.2 DUNE will consist of two neutrino detectors. The near detector (ND), at Fermilab (Illinois), will record particle interactions near the neutrino beam source. The far detector (FD) will be located ~ 1300 km from the ND and 1.5 km underground, at the Sanford Underground Research Facility (SURF, in South Dakota). Source: <https://www.dunescience.org>.

B.3.1 DUNE far detector

Neutrinos are detected via their interaction products i.e. observation of the leptons and hadrons that are produced when the neutrinos interact in the detector. In the single-phase

LArTPC design that will be used for the first DUNE FD module, three wire readout planes collect the ionisation charge that is generated when charged particles traverse the liquid-argon volume. The ionisation charge drifts in a constant electric field to the readout planes and the drift time provides a dimension of position information, hence the name “time projection chamber”. The position of the charge observed in each of the three planes is combined with the drift time to create three views of each neutrino interaction. The wires that form the planes are separated by approximately 5 mm giving the FD a fine-grained sampling of the neutrino interaction products. The electronic signals from the wires are sampled at a rate of 2 MHz, giving a similar effective spatial resolution in the time direction. Two of the wire planes are induction planes, biased to be transparent to the drifting electrons, such that they induce net-zero fluctuation in the wire current as they pass the wire plane. The third view is called the collection plane as it actually collects the drifting electrons. The four DUNE FD modules may not all have identical designs, but they will all produce similar images of the neutrino interactions, so the performance of the single-phase design is used in the work presented in Chapter 3. Other potential designs must have at least the same sampling capabilities as the single-phase design, if not better, to be considered.

B.3.2 CP -violation measurement

DUNE is sensitive to four neutrino oscillation parameters, namely Δm_{31}^2 , θ_{23} , θ_{13} and δ_{CP} , which can be measured using four data samples: two for neutrinos and two for antineutrinos. The FD data used in the oscillation analysis measure the “disappearance” channels (i.e. $\nu_\mu \rightarrow \nu_\mu$ and $\bar{\nu}_\mu \rightarrow \bar{\nu}_\mu$), which are primarily sensitive to $|\Delta m_{31}^2|$ and $\sin^2 2\theta_{23}$, and the “appearance” channels (i.e. $\nu_\mu \rightarrow \nu_e$ and $\bar{\nu}_\mu \rightarrow \bar{\nu}_e$), which are sensitive to all four parameters, including the sign of Δm_{31}^2 . In all of these samples, interactions where the neutrinos scatter via charged-current (CC) exchange off the nuclei in the far detector are selected. In a CC interaction, the final state includes a charged lepton with the same flavour as the incoming neutrino and one or more hadrons, depending on the details of the interaction. Therefore, a critical aspect of event selection is the ability to identify the flavour of the final-state lepton, and thus classify the signal (i.e. CC ν_μ , CC $\bar{\nu}_\mu$, CC ν_e and CC $\bar{\nu}_e$) interactions and have a powerful rejection of background events. At the energies relevant to the DUNE oscillation analysis, a final-state muon produces a long, straight track in the detector, while a final-state electron produces an electromagnetic (EM) shower. Examples of signal CC ν_e and CC ν_μ interactions are shown in Figs. 3.1 and 3.2a, respectively.

The main background to the CC ν_μ and CC $\bar{\nu}_\mu$ event selections are neutral current (NC) interactions with charged pions ($\pi^\pm$) in the final state that can mimic the $\mu^\pm$, an

example of which is shown in Fig. 3.2b. Neutral current interactions with a final-state π^0 meson, such as the one shown in Fig. 3.2c, where the photons from π^0 decay may mimic the EM shower from an electron, form the primary reducible background to the CC ν_e and CC $\bar{\nu}_e$ event selections. A small fraction of electron neutrinos are intrinsic to the beam (and thus are not the result of neutrino oscillations). These events form a background for the oscillation analysis as they are indistinguishable from CC ν_e appearance events. Once the four samples have been selected and the neutrino energy has been reconstructed, a fit is performed to the reconstructed neutrino energy distributions in the four samples to extract the neutrino oscillation parameters θ_{13} , θ_{23} , Δm_{31}^2 , and δ_{CP} . This fit accounts for the effects of systematic uncertainties, including the constraints on those uncertainties from fits to ND data. Figure B.3 shows the appearance samples and how they are expected to vary with the true value of δ_{CP} , for a data collection period of 3.5 years staged running in both FHC and RHC beam modes. The staging plan assumes two FD modules will be ready at the start of the beam data taking, and modules three and four will become operational after one year and two years, respectively. Full details of the DUNE staging plan and the oscillation analysis, including the assumed oscillation parameters, are provided in Ref. [Abi et al., 2020d].

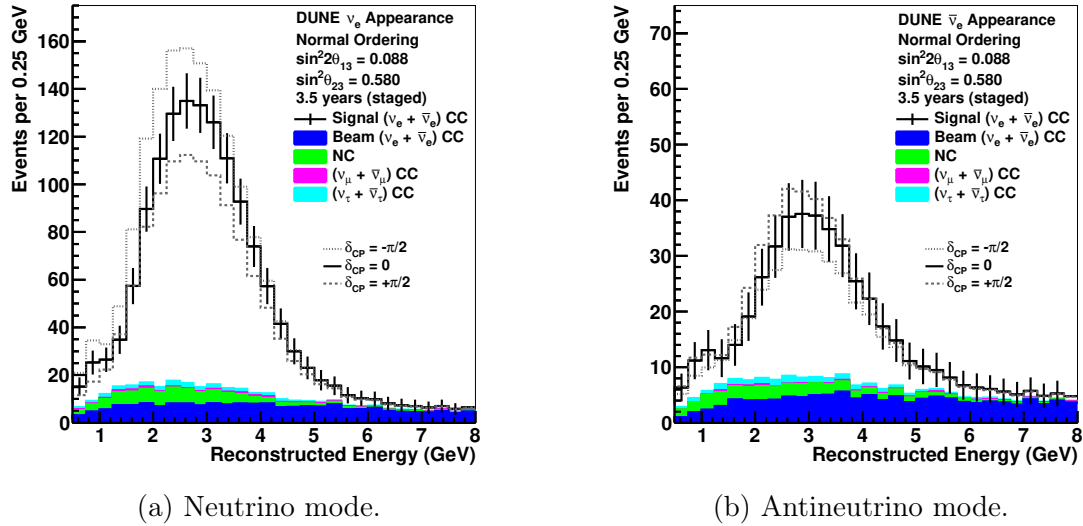


Fig. B.3 Reconstructed energy distribution of ν_e and $\bar{\nu}_e$ CC-like events selected by the convolutional neural network algorithm (CVN) assuming 3.5 years (staged) running in the neutrino-beam mode (a) and antineutrino-beam mode (b), for a total of seven years (staged) exposure. The plots assume normal mass ordering and include curves for $\delta_{CP} = -\pi/2$, 0, and $\pi/2$. Background from ν_μ -CC, ν_τ -CC, intrinsic ν_e -CC, and NC interactions are shown as stacked, filled histograms. Figure reproduced from Ref. [Abi et al., 2020d].

B.4 The T2K experiment

The Tokai to Kamioka (T2K) experiment [Abe et al., 2011a] is a long-baseline neutrino oscillation experiment in Japan that started its operation in 2009, with the primary goal of discovering ν_e appearance from a ν_μ beam. In T2K, the (muon) neutrino beam source is located in Tokai next to the near detectors (NDs), while the far detector, the Super-Kamiokande, is installed 295 km distant, in Kamioka; this setup is shown in Fig. B.4.

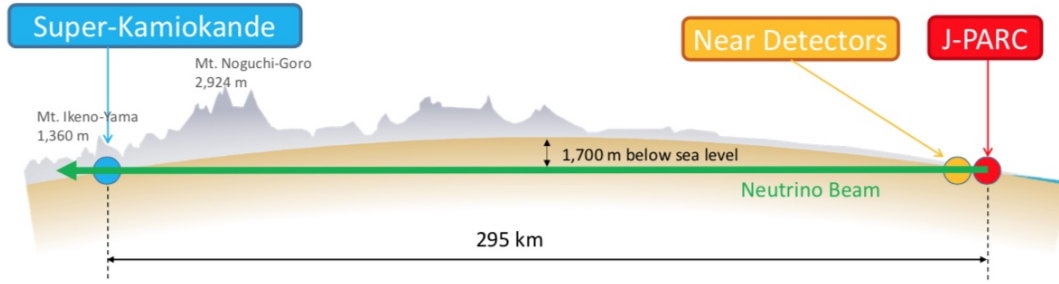


Fig. B.4 The T2K experiment has an intense muon neutrino beam at the J-PARC facility (Tokai), close to the near detectors. The far detector, the Super-Kamiokande, is ~ 300 km far away from the beam source, located under Mount Ikeno (Kamioka). Source: <https://www.t2k-experiment.org/t2k/>.

T2K was the first experiment to find $\nu_\mu \rightarrow \nu_e$ appearance [Abe et al., 2011b]; it has also given the first hint of a significant matter-antimatter asymmetry in neutrino oscillations [Abe et al., 2015, 2020; Escudero, 2016]. The experiment has two NDs, INGRID and ND280, both 280 metres from the target. For the latter, the plan is to upgrade it to improve its performance by modifying the upstream part of the detector, adding, among others, a new highly granular scintillator detector (SuperFGD).

B.4.1 SuperFGD

The Super Fine-Grained Detector (SuperFGD) [Abe et al., 2019] is a finely segmented scintillator detector which will be used for upgrading the ND280 detector of the T2K experiment. A finely segmented scintillator detector consists of a 3D matrix of plastic scintillator cubes. The scintillation light produced by charged particles traversing the cubes is read out by three orthogonal wavelength-shifting (WLS) fibers that transport the scintillation light out of the detector where silicon photomultipliers (SiPMs) convert it into a certain number of photoelectrons (p.e.), as illustrated in Figs. B.5 and B.6.

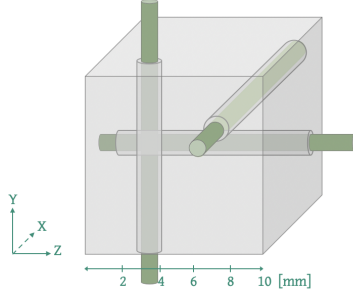


Fig. B.5 Geometry of a single SuperFGD element. Each cube (grey) is intersected by 3 WLS fibres (green). The whole SuperFGD will be array of $56 \times 184 \times 192$ of these elements ($H \times L \times W$).

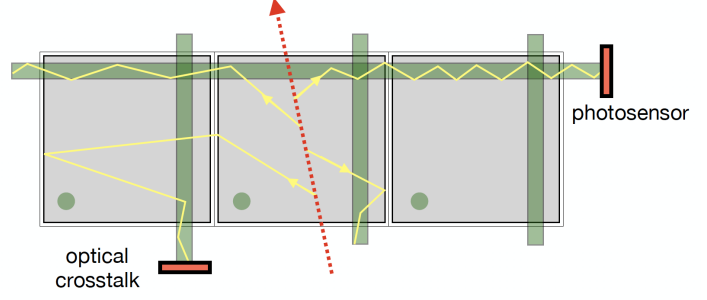


Fig. B.6 Sketch of the signal generation, fibre transport and signal detection processes highlighting the production of optical crosstalk signals. The cubes are depicted in grey, the WLS fibres in green, the dashed red line is a charged track and photons are illustrated in yellow.

The detector will have 2 million plastic scintillator cubes, each $1 \times 1 \times 1 \text{ cm}^3$ in size, and provides three orthogonal 2D projections of particle tracks produced by a neutrino interaction, as depicted in Fig. 4.1a (from Chapter 4).

To reconstruct neutrino interactions in three dimensions, the light yield measurements in the three 2D views are matched together, as shown in Fig. 4.1b (also from Chapter 4). The 3D objects, corresponding to the cubes where the energy deposition is reconstructed, are referred to as *voxels*. In addition to the cubes where a particle has passed and deposited energy, light-leakage between neighbouring cubes can create additional *crosstalk* signals [Blondel et al., 2020; Mineev et al., 2019], as depicted in Fig. B.6. Moreover, ambiguities in the matching process can produce *ghost* voxels, shown in Fig. B.7.

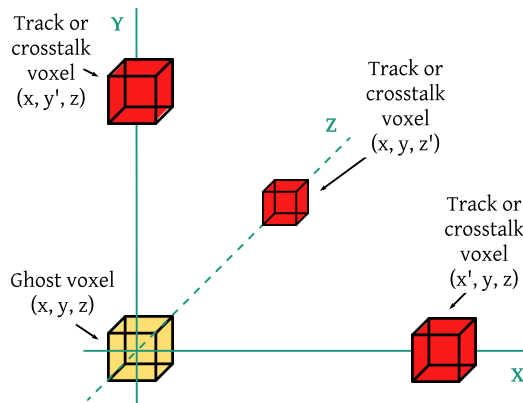


Fig. B.7 Example of a ghost voxel appearance. When matching the coordinates of the fibres that recorded an energy deposition, voxels may appear where no true signal exists, becoming non-physical ghost voxels.

Appendix C

Complementary information to Chapter 4

C.1 Input variables

The list of variables used as features for the graph nodes is given below. Each node is placed at XYZ coordinates matching the centre of a cube, however, these centre coordinates are not node variables by themselves since the detector response is isotropic. The numbers in front of each variable match those in Fig. C.1.

- 0-2: peXY , peXZ , peYZ
Number of photons detected in the XY, XZ or YZ-fiber intersecting the cube under consideration corrected by the expected attenuation.
- 3-5: mXY , mXZ , mYZ
Number of active voxels intersected by the fiber associated to peXY , peXZ or peYZ .
- 6: pewav
Average number of detected photons peXY , peXZ , peYZ , each weighted by the fiber multiplicity mXY , mXZ , mYZ .

$$\text{pewav} = \frac{\frac{\text{peXY}}{\text{mXY}} + \frac{\text{peXZ}}{\text{mXZ}} + \frac{\text{peYZ}}{\text{mYZ}}}{3}$$

- 7-9: pullX , pullY , pullZ
Relative difference between the light measured in two different 2D planes.

$$\text{pullX} = \frac{\text{peXY} - \text{peXZ}}{\text{peXY} + \text{peXZ}} \quad \text{pullY} = \frac{\text{peXY} - \text{peYZ}}{\text{peXY} + \text{peYZ}} \quad \text{pullZ} = \frac{\text{peXZ} - \text{peYZ}}{\text{peXZ} + \text{peYZ}}$$

- 10: `residual`

Similarity of the light yield measured in the three 2D planes, measured as the squared distance from each `peXY`, `peXZ`, `peYZ` to the average, weighted by the squared average.

$$\mu = \frac{\text{peXY} + \text{peXZ} + \text{peYZ}}{3}$$

$$\text{residual} = \frac{(\text{peXY} - \mu)^2 + (\text{peXZ} - \mu)^2 + (\text{peYZ} - \mu)^2}{\mu^2}$$

- 11: `pullXYZ`

Similarity of the light yield measured in the three 2D planes, measured as a combination of 2D pulls (a_1, a_2, a_3) weighted by `pewav`.

$$a_1 = \frac{\frac{\text{peXY}}{\text{mXY}} - \frac{\text{peXZ}}{\text{mXZ}}}{\frac{\text{peXY}}{\text{mXY}} + \frac{\text{peXZ}}{\text{mXZ}}} \quad a_2 = \frac{\frac{\text{peXY}}{\text{mXY}} - \frac{\text{peYZ}}{\text{mYZ}}}{\frac{\text{peXY}}{\text{mXY}} + \frac{\text{peYZ}}{\text{mYZ}}} \quad a_3 = \frac{\frac{\text{peXZ}}{\text{mXZ}} - \frac{\text{peYZ}}{\text{mYZ}}}{\frac{\text{peXZ}}{\text{mXZ}} + \frac{\text{peYZ}}{\text{mYZ}}}$$

$$\text{pullXYZ} = \frac{a_1 a_2 + a_1 a_3 + a_2 a_3}{\text{pewav}}$$

- 12: `ratioMQ`

Ratio between the average voxel multiplicity in the three fibers and `pewav`.

$$\text{ratioMQ} = \frac{\frac{\text{mXY} + \text{mXZ} + \text{mYZ}}{3}}{\text{pewav}}$$

- 13-14: `R1`, `R3`

Number of active neighbour voxels in a sphere of certain radius: `R1`=1 cm, `R2`=2 cm, and `R3`=5 cm. `R2` was not used as a variable due to the high correlation with `R1`, but is used to compute `RR`.

- 15-20: `x+`, `x-`, `y+`, `y-`, `z+`, `z-`

Boolean variables representing the existence of immediate neighbours in each of the 6 surrounding cubes

- 21: `orthogonal_neighbour`

It is 1 if any of `x+`, `x-`, `y+`, `y-`, `z+`, `z-` is 1.

- 22: `RR`

Ratio between the number of close and far voxels. The $\epsilon = 10^{-7}$ prevents numerical problems when `R3`=0.

$$RR = \frac{R2}{R3 + \epsilon}$$

- 23: `ratioDQ`

Ratio between the average voxel distance `aveDist` around the voxel and the weighted average light yield `pewav`.

$$\text{ratioDQ} = \frac{\text{aveDist}}{\text{pewav}}$$

- 24: `aveDist`

Average distance from the voxel center C to all fired voxel centers (C_i) within a sphere of radius 2.5 cm.

$$\text{aveDist} = \frac{1}{N} \sum_i^N \text{EuclidianDist}(C, C_i)$$

Figure C.1 shows the correlations of these input variables for the GENIE and P-Bomb data samples. Differences between the two matrices arise from the different topologies of interactions produced by the two generator methods.

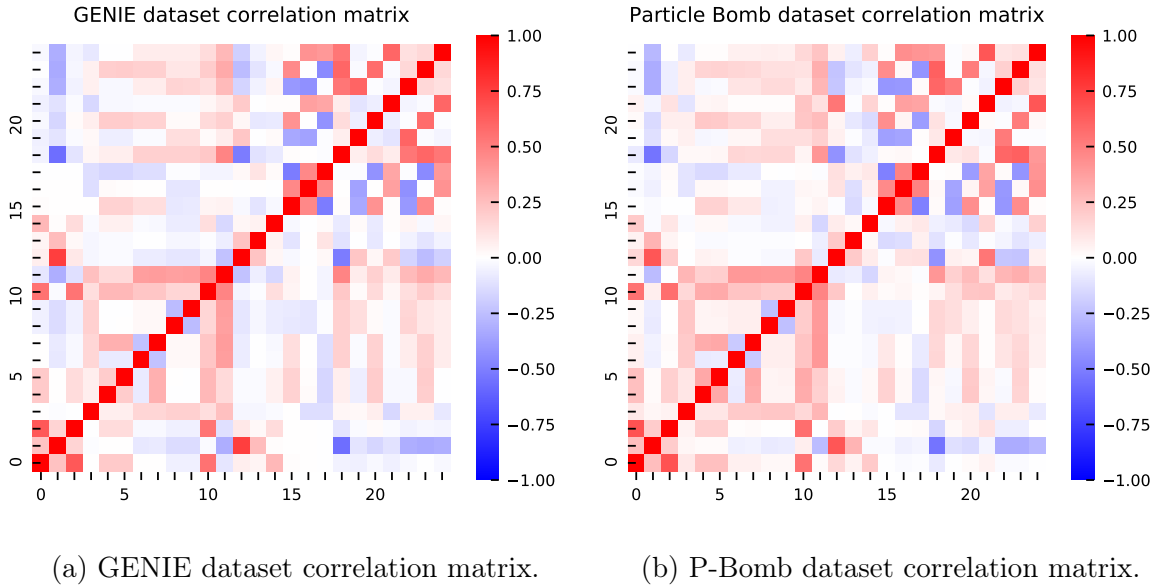


Fig. C.1 Correlation matrices for the input variables of the GENIE (a) and P-Bomb (b) datasets used. The list of 25 variables defined in Appendix C gives the mapping between the numbers on the axes and the variable names.

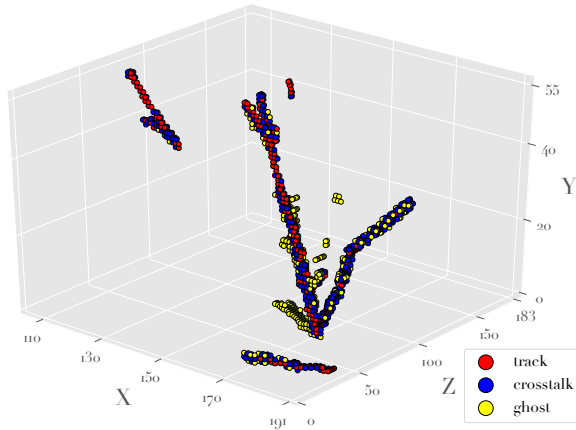
A number of these variables are calculated from the same underlying properties of the energy deposits. In theory, an infinitely deep GNN trained on an infinite amount of

training data would be able to extract all of the information required for classification from the few underlying properties. In practice, we use a larger number of derived variables to guide the GNN to allow it to more easily extract information from the data and to converge quickly in the training process. Global position was intentionally not used as a variable to prevent the GNN from learning very fine details from the choice of training sample.

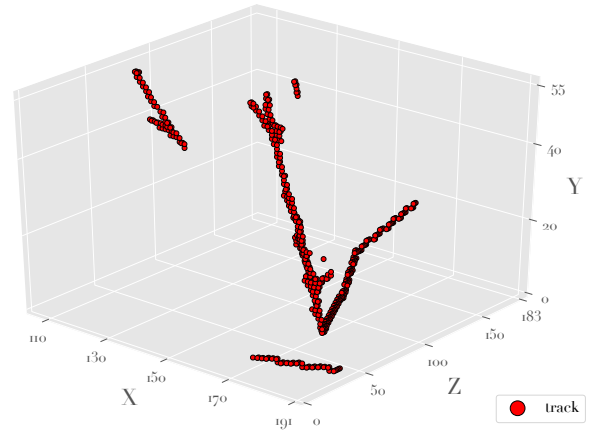
C.2 Event gallery

This section contains a number of visualisations to show the classification performance of the GNN for a number of neutrino interactions with different complexity and topology. Displays are shown for different events in Figs. C.2 - C.7: all voxels with their true classification, only the true track voxels, the classified track voxels using the charge cut method, and the classified track voxels using the GNN. The interactions shown here are examples of interactions containing many ghost voxels in order to showcase the GNN performance.

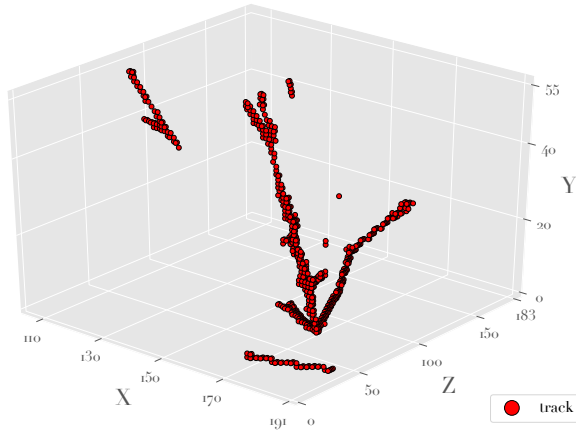
The track voxel classification ability of the charge cut and GNN methods can be seen by comparing subfigures (c) and (d) with (b), respectively, for each interaction. The GNN is able to reject ghost voxels very well, as shown in Figs. C.3, C.5, C.6 and C.7 where ghost tracks remain using the charge cut method. In general, the performance improvement from the GNN increases with the complexity of the interactions. For simple interactions with only a single muon in the final state both methods perform similarly.



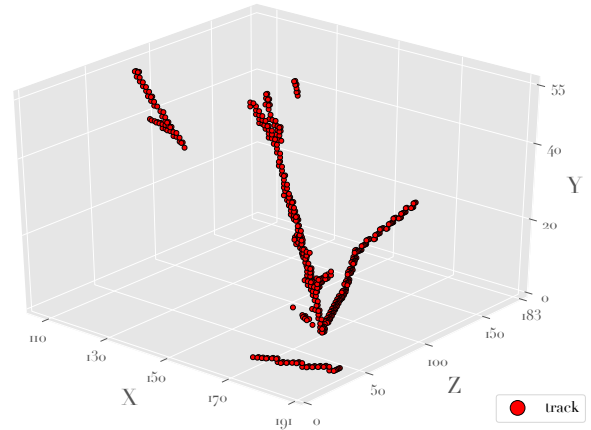
(a) The 3D voxels labelled as track (red), crosstalk (blue) and ghost (yellow) according to the truth information from the simulation are shown.



(b) Only the 3D voxels labelled as track according to the truth information from the simulation are shown.

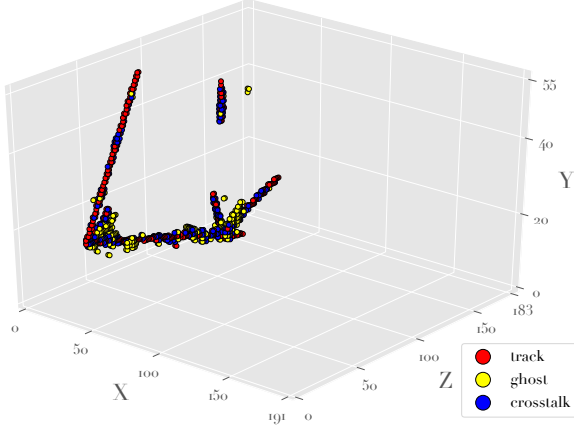


(c) The 3D voxels labelled as track according to the charge cut classification are shown.

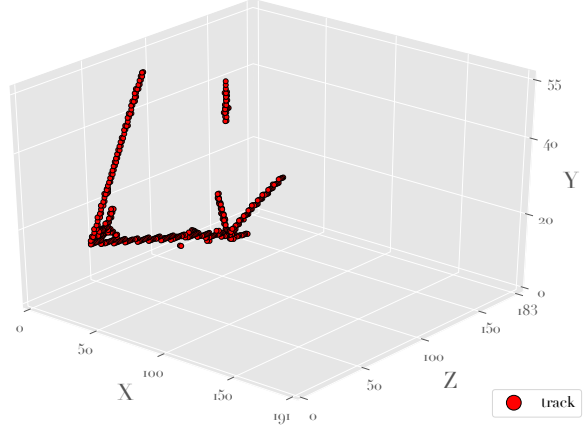


(d) The 3D voxels labelled as track according to the GNN classification are shown.

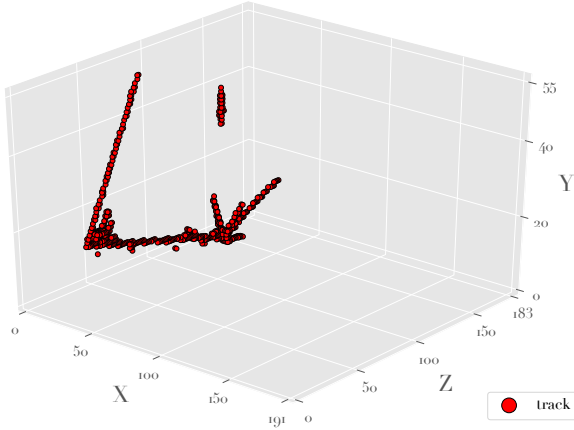
Fig. C.2 3D visualisation of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views. The GNN cut is able to almost entirely reject the fake track travelling on the XZ plane and stopping near to the vertex at $X \sim 160$ cm and $Z \sim 70$ cm, while the charge cut cannot. The energy of the incoming neutrino is 4.754 GeV. The axes are in cm.



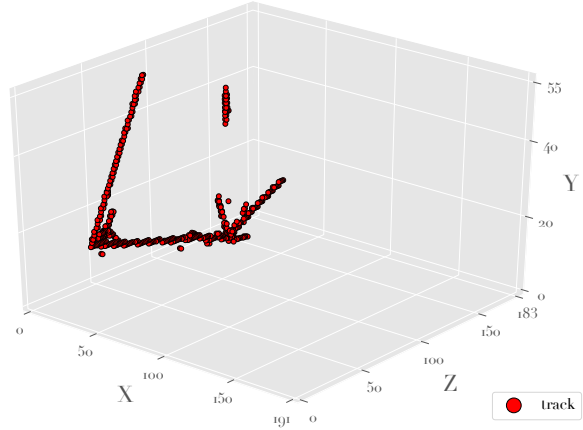
(a) The 3D voxels labelled as track (red), crosstalk (blue) and ghost (yellow) according to the truth information from the simulation are shown.



(b) Only the 3D voxels labelled as track according to the truth information from the simulation are shown.

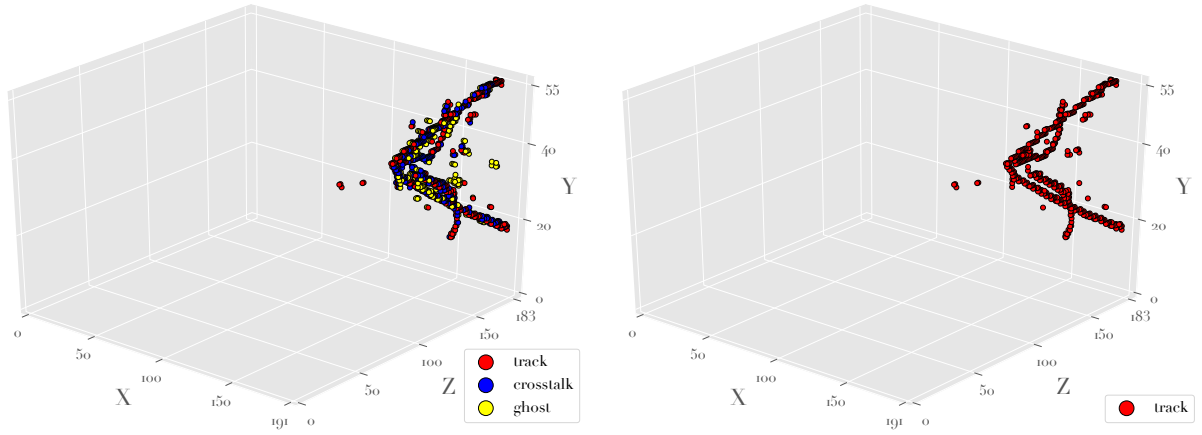


(c) The 3D voxels labelled as track according to the charge cut classification are shown.



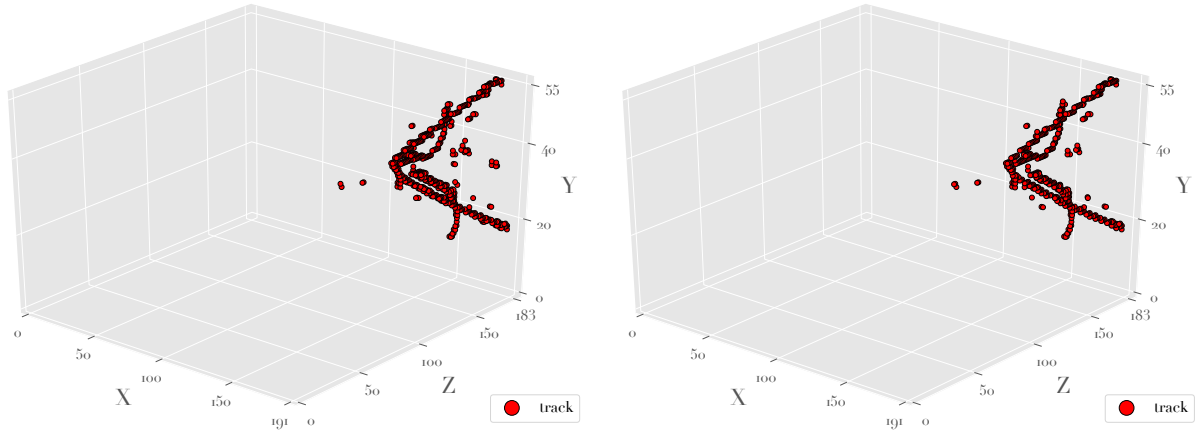
(d) The 3D voxels labelled as track according to the GNN classification are shown.

Fig. C.3 3D visualisation of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views. The charge cut is not able to reject two fake tracks, one coming from a vertex at $X < 50$ cm $Z < 50$ cm travelling on the XZ plane and stopping near to the vertex at $X \sim 160$ cm and $Z \sim 70$ cm. Moreover, the charge cut leaves a bump of ghost voxels around the vertex that could mimic the interaction of a few low-energy protons, an effect that could bias the reconstruction of the neutrino energy. The energy of the incoming neutrino is 760 MeV. The axes are in cm.



(a) The 3D voxels labelled as track (red), crosstalk (blue) and ghost (yellow) according to the truth information from the simulation are shown.

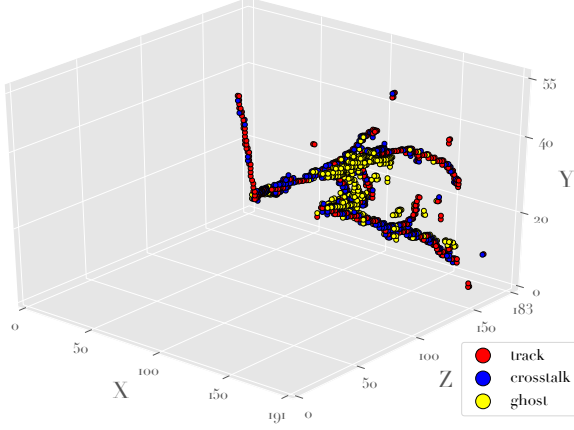
(b) Only the 3D voxels labelled as track according to the truth information from the simulation are shown.



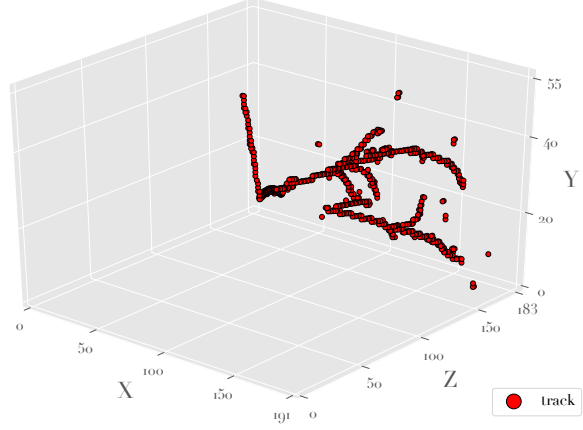
(c) The 3D voxels labelled as track according to the charge cut classification are shown.

(d) The 3D voxels labelled as track according to the GNN classification are shown.

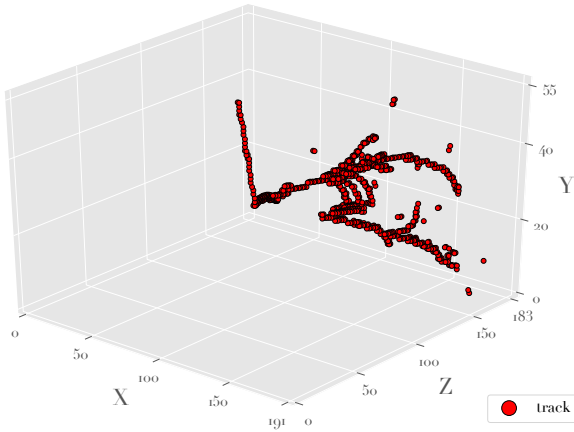
Fig. C.4 3D visualisation of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views. In this even the performance of GNN and the charge cut is quite similar because the ghost voxels are mainly given by the overlap of crosstalk hits in the 2D readout views. The energy of the incoming neutrino is 5.076 GeV. The axes are in cm.



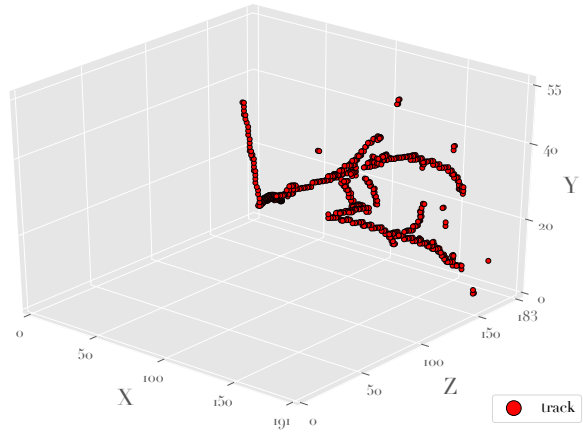
(a) The 3D voxels labelled as track (red), crosstalk (blue) and ghost (yellow) according to the truth information from the simulation are shown.



(b) Only the 3D voxels labelled as track according to the truth information from the simulation are shown.

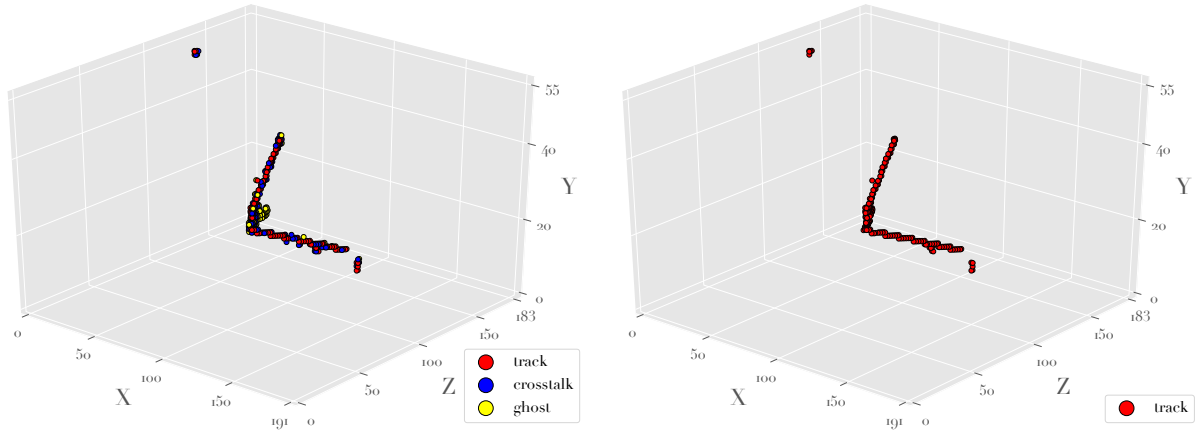


(c) The 3D voxels labelled as track according to the charge cut classification are shown.



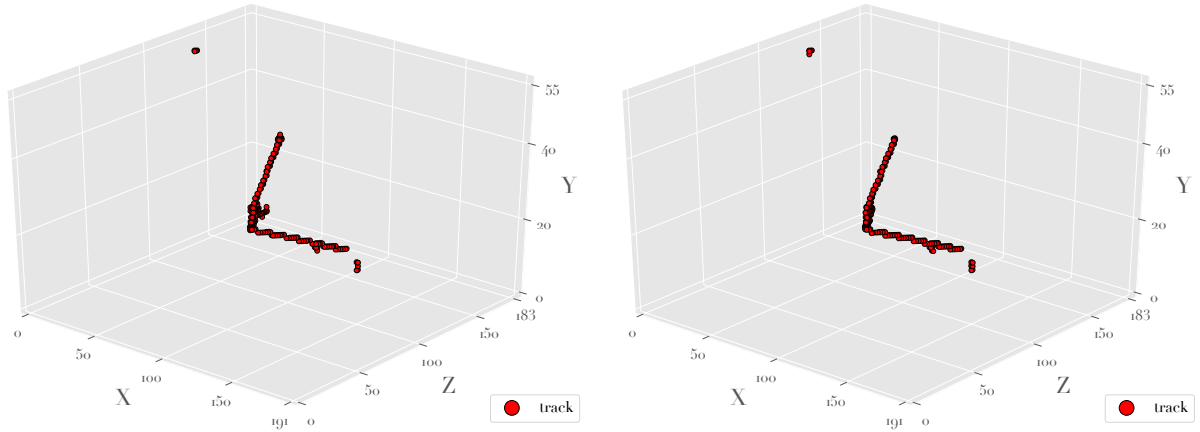
(d) The 3D voxels labelled as track according to the GNN classification are shown.

Fig. C.5 3D visualisation of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views. This neutrino event has a quite high multiplicity and tracks are quite close each other. This produce relatively big clusters of ghost voxels that produce at least two fake tracks even after the charge cut. Instead GNN allows to classify ghosts more precisely and correctly visualise the correct number of tracks. Moreover, the charge cut makes true tracks more fat making their separation harder and, potentially, less precise the particle momentum reconstruction. The energy of the incoming neutrino is 1.064 GeV. The axes are in cm.



(a) The 3D voxels labelled as track (red), crosstalk (blue) and ghost (yellow) according to the truth information from the simulation are shown.

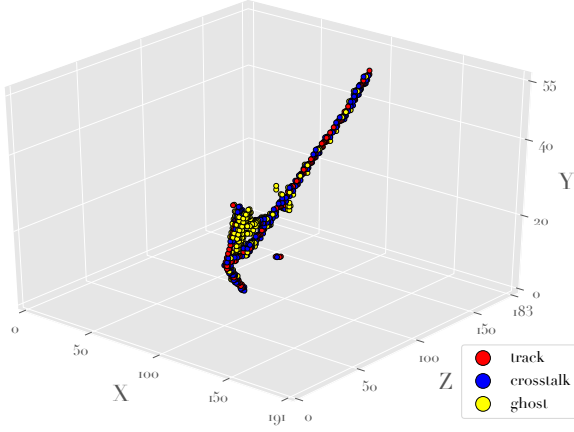
(b) Only the 3D voxels labelled as track according to the truth information from the simulation are shown.



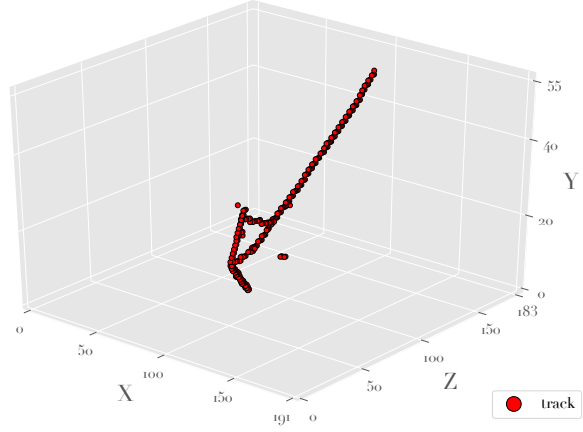
(c) The 3D voxels labelled as track according to the charge cut classification are shown.

(d) The 3D voxels labelled as track according to the GNN classification are shown.

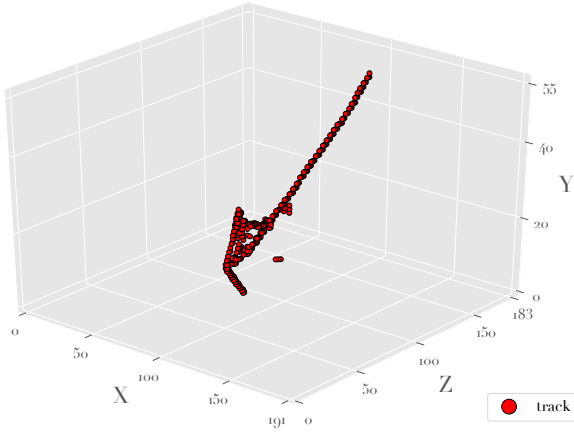
Fig. C.6 3D visualisation of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views. Although this is a relatively simple neutrino event, the charge cut is not able to reject a fake track stopping near the neutrino interaction vertex while GNN can provide a much cleaner reconstruction. The energy of the incoming neutrino is 1.132 GeV. The axes are in cm.



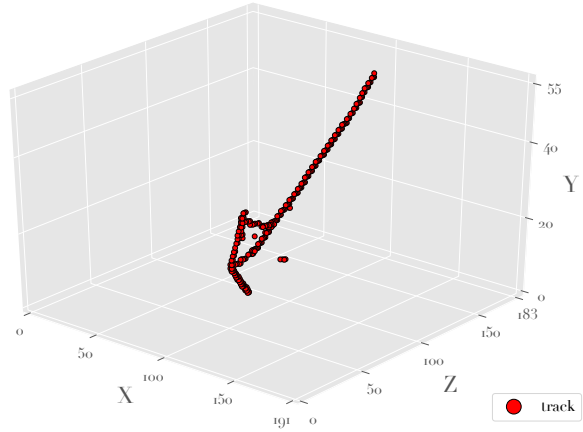
(a) The 3D voxels labelled as track (red), crosstalk (blue) and ghost (yellow) according to the truth information from the simulation are shown.



(b) Only the 3D voxels labelled as track according to the truth information from the simulation are shown.



(c) The 3D voxels labelled as track according to the charge cut classification are shown.



(d) The 3D voxels labelled as track according to the GNN classification are shown.

Fig. C.7 3D visualisation of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views. In the neutrino event GNN can easily reject the relatively big cluster of ghost voxels that would make difficult a proper reconstruction of the number of tracks and corresponding energy, in particular near to the interaction vertex. The energy of the incoming neutrino is 1.897 GeV. The axes are in cm.

Appendix D

Complementary information to Chapter 6

D.1 DaloFlow user guide

The very first time you deploy DaloFlow, you would need to execute the following commands:

1. Clone the GitHub repository:

```
git clone https://github.com/saulam/daloflow.git
cd daloflow
chmod +x ./daloflow.sh
./daloflow.sh postclone
```

2. Install pre-requisites:

```
./daloflow.sh prerequisites
```

3. Build the container image:

```
./daloflow.sh build
```

D.1.1 Several containers on a single node

This option allows developers to run their jobs on a single computer, as shown in Fig. D.1.

A DaloFlow working session has three phases:

1. Start the desired number of containers using Docker Compose:

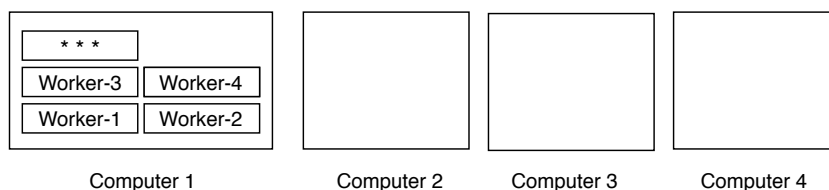


Fig. D.1 Several workers on a single computer.

```
./daloflow.sh start 2
```

A file named *machine_horovod* will be created, including one line per container.

2. Execute the deep-learning job:

```
./daloflow.sh mpirun 2 \
"python3_tensorflow2_keras_mnist.py"
```

3. To end the working session, just execute:

```
./daloflow.sh stop
```

D.1.2 Several containers on several nodes

This option allows developers to execute their jobs on several computers, as shown in Fig. D.2.

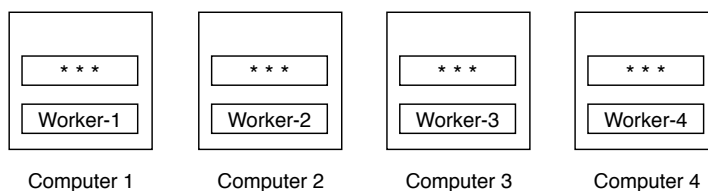


Fig. D.2 Several workers on several computers.

1. Start the desired number of distributed containers using Docker Swarm:

```
./daloflow.sh swarm-start 4
```

A file named *machine_horovod* will be created, including one line per container.

2. Execute the deep-learning job:

```
./daloflow.sh mpirun 4 \
"python3_tensorflow2_keras_mnist.py"
```

3. To end the working session, just execute:

```
./daloflow.sh swarm-stop
```

D.2 Horovod deployment comparison

We can check the simplicity of DaloFlow by executing a similar example using only Horovod:

1. Open four terminals and connect each one to the corresponding host.
2. Execute on the secondary workers:

```
host2$ nvidia-docker run -it --network=host -v /mnt/share/ssh:/
      root/.ssh horovod:latest \
      bash -c "/usr/sbin/sshd -p 12345; _sleep_infinity"
host3$ nvidia-docker run -it --network=host -v /mnt/share/ssh:/
      root/.ssh horovod:latest \
      bash -c "/usr/sbin/sshd -p 12345; _sleep_infinity"
host4$ nvidia-docker run -it --network=host -v /mnt/share/ssh:/
      root/.ssh horovod:latest \
      bash -c "/usr/sbin/sshd -p 12345; _sleep_infinity"
```

3. Execute on the primary worker:

```
host1$ nvidia-docker run -it --network=host -v /mnt/share/ssh:/
      root/.ssh horovod:latest \
horovodrun -np 16 -H host1:4,host2:4,host3:4,host4:4 -p 12345
      python tensorflow2_keras_mnist.py
```


Index

- CP*-violation, 2, 33, 142
- Activation function, 123
- Artificial neural network, 119
- Backward propagation, 122
- Batch, 117
- Classification problem, 118
- Computational efficiency, 4, 19, 79
- Computer vision, 9, 129
- Convolutional neural network, 9, 12, 21, 128
- Cost function, 27, 50, 122, 124
- Crosstalk, 45
- Deep-learning framework, 52, 89, 125
- Distributed training, 17, 79, 133
- Epoch, 117
- Event classification, 3, 11, 24
- Far detector, 2, 21, 140
- Fast simulation, 9, 73, 74
- Forward propagation, 122
- Generative adversarial network, 10, 64, 132
- Ghost voxel, 45
- Graph neural network, 15, 47, 130
- Hyper-parameters, 123
- Initialisation method, 124
- Iteration, 117
- Model parameters, 64
- Near detector, 2, 61, 140
- Network parameters, 25, 121
- Network weights, *see* Network parameters
- Neutrino flavour, 1, 25, 137
- Neutrino oscillation experiments, 2, 140
- Neutrino oscillations, 1, 137
- Optimiser, 27, 52, 69, 125
- Particle track, 3, 45
- Reconstruction, 3, 45, 140, 145
- Sample, 117
- Signal efficiency, 3, 12, 32, 52
- Signal purity, 33, 52
- Stochastic gradient descent, 123
- Training example, 117
- Voxel, 45