Høgskulen på Vestlandet

BACHELOROPPGAVE

Deep Learning Approach for Binary Classification of Microscopic Black Holes and Sphalerons

Optimization by employing custom loss Function

Dyp lærings metode for binær klassifisering av mikroskopiske svarte hull og sfaleroner

optimalisering ved bruk av tilpasset tapfunksjon

Daniel Kristiansen Gunleiksrud og Sayna Ganjei

Dataingeniør og Informasjonsteknologi
Fakultet for ingeniør- og naturvitskap (FIN)/Institutt for datateknologi, elektroteknologi og realfag/Dataingeniør og Informasjonsteknologi
Carsten Helgesen

Innleveringsdato (22 Mai 2023)

Rapportens tittel: Klassifisering av sphaleroner og micro svarte hull Classification of Sphalerons and Micro Black Holes	*Dato: 22.05.2023*
Forfatter(e): *Daniel Kristiansen Gunleiksrud og Sayna Ganjei*	*Antall sider u/vedlegg: 85 sider* *Antall sider vedlegg: 116 sider*
Studieretning: *Dataingeniør og Informasjonsteknologi*	*Antall disketter/CD-er:* 0
Kontaktperson ved studieretning: *Carsten Helgesen*	*Gradering:* Ingen
Merknader:	

Oppdragsgiver: ATLAS-temaet ved Høgskulen på vestlandet i regi av CERN	*Oppdragsgivers referanse:* Ingen
Oppdragsgivers kontaktperson: Trygve Buanes Steffen Mæland	*Telefon:* 55 58 70 87/480 58 430 55 58 77 94

Sammendrag:

Dette prosjektet presenterer en dyp læringsmetode for klassifisering av mikroskopiske svarte hull og sfaleroner ved hjelp av konvolusjonelle nevrale nettverk. Hovedmålet med dette prosjektet er å utvikle en dyp læringsmodell som nøyaktig og effektivt kan forutsi klassifiseringen av simulerte kollisjonsdata ved hjelp av veiledet læring.

Stikkord:

Dyplæring	ATLAS, CERN	Klassifisering

Abstract

This study presents a deep learning approach for classification of microscopic black holes and sphalerons using convolutional neural networks. The main objective of this study is to develop a deep learning model that can accurately and efficiently predict the classification of simulated collision data using supervised learning techniques. The proposed approach aims to overcome the challenges associated with traditional classification methods, which may be time-consuming and require significant expertise in particle physics and advanced mathematics. By leveraging the power of deep learning, this study demonstrates the potential to improve the accuracy and efficiency of microscopic black hole and sphaleron classification.

Acknowledgements

We would like to express our gratitude to the ATLAS group at Western Norway University of Applied Sciences (HVL), for providing us with the opportunity to work on such an exciting project. Their support and guidance has been invaluable in the production of this paper, by generously sharing their expertise and knowledge with us throughout the project.

We would like to extend special thanks to our external advisors, Associate Professors Trygve Buanes and Steffen Mæland, whose invaluable insights and feedback have helped us to shape and refine our work.

Furthermore, we also want to acknowledge the contribution of Aurora Singstad Grefsrud, who developed the 2D histogram data used in this project.

Finally, we would like to thank Carsten Helgesen, our intern advisor, who has provided valuable assistance with refining the structure and writing methods of this paper. His guidance has been significant in helping us to write a consistent paper.

Contents

Glossary

Physics

Antimatter - Matter composed of particles with opposite charge of what ”normal” matter has.

ATLAS detector - A particle detector used in the Large Hadron Collider (LHC) at CERN to study high-energy particle collisions. The ATLAS detector is designed to detect a wide range of particles and phenomena resulting from these collisions, including the Higgs boson.

ATLAS experiment - A research collaboration aimed at discovering new particles and phenomena to improve our understanding of the fundamental nature of the universe. The ATLAS experiment uses the ATLAS detector at the LHC to collect and analyze data from high-energy particle collisions.

Big Bang - A model describing the origin of our universe. A big ”explosion” in which all matter originated from.

CERN - A European research organization that operates the world’s largest particle physics laboratory. Located in Switzerland, CERN conducts research primarily on high-energy particle physics, including the study of subatomic particles and their interactions.

Dark Matter - Unknown type of matter which does not interact strongly with normal matter. Dark matter has gravity like ”normal” matter. 85% of matter in our universe consists of dark matter.

Decay - A spontaneous process where a unstable particle transforms into multiple other particles.

Elemantary particles - Smallest known building blocks of the universe.

Electromagnetic calorimeter - A detector that measures the energy of electrons and photons as they interact with matter. The electromagnetic calorimeter is one of several subsystems used in the LHC detector to study high-energy particle collisions.

Electrons - Elementary particle that has a negative charge and no strong interaction. Often orbiting an atom.

η(Eta) - A coordinate used to measure the angle of a particle relative to the particle beam in a detector.

Exotic particles - Unusual or rare subatomic particles that have properties that are not well understood or predicted by existing theories. Examples include particles with multiple quarks or anti-quarks, or particles that violate conservation laws.

Fundamental forces - The basic forces that govern the behavior of subatomic particles, the strong nuclear force, weak nuclear force, electromagnetic force, and gravitational force.

Gravitational attraction - The force of attraction between two objects with mass, which is proportional to the product of their masses and inversely proportional to the square of the distance between them.

Hadron Calorimeter - A detector that measures the energy of hadrons, which are particles that are made up of quarks and are subject to the strong nuclear force.

Hadronization - The process by which quarks and gluons produced in high-energy particle collisions combine to form stable hadrons, such as protons and neutrons.

Hadrons - Subatomic particles made up of quarks that are subject to the strong nuclear force, such as protons and neutrons.

Higgs boson - A subatomic particle that is thought to give mass to other particles through interactions with the Higgs field. Its discovery was announced by CERN in 2012.

Jet - A collimated stream of particles produced in high-energy collisions, such as those that occur at the LHC.

Large Hadron Collider (LHC) - The world's largest and most powerful particle accelerator, located at CERN in Switzerland. The LHC is used to study the fundamental properties of subatomic particles by colliding them at high speeds.

Matter - The substance that makes up all physical objects, including subatomic particles.

Microscopic black hole - A hypothetical type of black hole that is much smaller than typical black holes and is thought to be produced by high-energy particle collisions.

Monte carlo simulation - Is a mathematical technique used to simulated events with uncertain outcomes. Used to generate the data used for the construction of 2D Histograms.

Particle - A small, subatomic unit that cannot be further divided without losing its unique properties. Examples include protons, electrons, and neutrinos.

Particle collision - A process in which two or more subatomic particles interact with each other, resulting in the formation of new particles and the release of energy.

Photons - Elementary particles that make up light and other forms of electromagnetic radiation. They have no mass and travel at the speed of light.

φ(Phi) - An angle used to describe the direction of particles in LHC detectors in the plane orthogonal to the beam axis.

Proton beam - A high-energy beam of protons that is accelerated and directed toward a target in order to produce collisions and study subatomic particles.

Proton-proton collision - A type of high-energy particle collision that occurs when two beams of protons are directed toward each other and collide, producing a variety of new particles.

Silicon tracker - The first component of the LHC detector that measures the direction of particles produced in collisions.

Sphaleron - A hypothetical phenomenon in physics that is thought to be related to the breaking of certain symmetries in subatomic particles.

TeV - A unit of energy used in particle physics, equivalent to one trillion electron volts.

Vacuum state - The lowest possible energy state of a system, which may still contain some energy due to fluctuations in the vacuum. In particle physics, the vacuum state is thought to be filled with virtual particles that constantly pop in and out of existence.

Velocity - The rate at which an object changes its position with respect to time, typically measured in units of meters per second or kilometers per hour.

Machine learning

2D Histogram - A graphical representation of data that shows the frequency of values across two dimensions.

Activation function - An activation function is a function that determines the value that a node should send to the next node.

Artificial neural network - A machine learning algorithm that uses artificial neurons to learn data from data to make decisions or predictions.

Artificial Neuron - An artificial neuron, also called a perceptron, is a computational unit that takes inputs and produces an output based on a set of weights and bias. It is the fundamental building block of an artificial neural network.

Average pooling - A pooling operation in which each output element is the average of the corresponding input elements within a pooling region. It is commonly used in convolutional neural networks to reduce the spatial dimensions of feature maps.

Backpropagation - Involves calculating the gradient of the loss function with respect to the network's parameters and using it to update the weights of the network in reverse order.

Batch - Refers to a smaller section of training or test data that are being passed through the machine learning model. Choosing a batch size too large may cause the gpu to run out of memory.

Bias - A parameter value in a function that is a constant value added to the output of a node. This value is used to adjust the output of each node in the network.

Bin box - Related to histograms. The bin box refers to a interval of number of which all elements gets places in the same bar plot.

Bottleneck residual block - A type of residual block in a deep neural network architecture that reduces the number of feature maps before applying a convolutional layer. This is done to decrease the computational cost and improve performance.

Channel - Refers to the third dimension of an image(RGB layer) or feature map in a convolutional neural network. Each channel represents a different feature or filter that the network has learned.

CNN - Convolutional Neural Network (CNN) is a neural network architecture that divides an image into small parts and iterates through them, making it possible to find patterns independently of position.

Concatenate - A way to combine items. Often refers to combining matrises along a given axis.

Convolutional Block - A building block in a convolutional neural network architecture that typically consists of several convolutional layers with nonlinear activation functions, followed by a pooling layer.

Convolutional Layer - A layer in a convolutional neural network that performs convolution on the input image or feature map with a set of learnable filters, producing a set of output features.

Convolutional Operation - The mathematical operation performed by a convolutional layer in a neural network. It involves sliding a kernel over the input image or feature map and computing the dot product between the kernel and the local region of the input.

Custom-loss-x - The Custom loss function developed by modifying the softmax function. The x indicates the root index used.

Deep learning - A subfield of machine learning that involves training artificial neural networks with multiple layers.

Dimensionality - The number of dimensions in a dataset or a feature space. In the context of deep learning, dimensionality refers to the number of parameters or weights in a neural network.

Dropout - Dropout determines the probability that a connection between two nodes will temporarily disappear. This is used to prevent overfitting.

Epoch - A complete pass through the entire training dataset during the training of a neural network.

Error Rate - The proportion of incorrect predictions made by a model on a given dataset. It is commonly used as a performance metric for classification tasks.

Exploding Problem - The phenomenon that occurs when the gradient values in a neural network become very large during training, causing the weights to update by a large amount and destabilizing the training process. This can occur when the learning rate is too high or when the network architecture is not well-suited to the task.

Fast.ai - High-level Python library for machine learning.

Feature engineering - The process of creating new variables from existing data.

Feature map - A 2D array obtained from applying a filter to an input image, representing a specific feature that the filter is designed to detect.

Filter - A small matrix of numerical values used for operations such as convolution and pooling in a convolutional neural network.

Fully connected layer - A type of neural network layer where each neuron is connected to every neuron in the previous and next layers.

Global minimum - The lowest possible value of a function, and represents the optimal solution to an optimization problem.

Gradient - A vector that represents the slope of a function at a given point, and is used in optimization algorithms to find the minimum or maximum of the function.

Hidden layer - Layers that are between input layer and output layers. The layers where machine learning occurs.

Hyperparameter - Model-specific parameter that does not change during training. Examples of this could be model architecture, loss function, learning rate, or number of epochs.

Identity block - A residual block in a convolutional neural network that preserves the input's spatial dimensions.

ImageNet - Image database containing 1000 classes. Designed to be used for training AI algorithms.

Image classification - The task of assigning a label to an input image, based on the content of the image.

Input layer - The first layer of a neural network, which receives the input data.

Iteration - A single step in the optimization process of a machine learning model.

Kernel - Another name for a filter in a convolutional neural network.

Learning rate - A variable that determines the size of the steps taken during gradient descent.

Library - A collection of pre-written code in a specific programming language, designed to be used by other programs.

Learning rate - A variable that determines the size of the steps taken during gradient descent, and affects the speed and accuracy of the training process.

Loss function - Measures the difference between the predicted output of a machine learning model and the target output, and is used to train the model by adjusting its parameters to minimize the loss.

Loss value - The numerical value of the loss function for a specific set of input and output data.

Max pooling - A type of pooling operation in a convolutional neural network, where the maximum value within a specific region is selected as the output.

Model parameter - Model parameter is the weight and bias between the different node connections.

Optimizer - An algorithm used to minimize the loss function during training. Performs the gradient decent to change the model parameters to reduce loss. Examples of optimizers are Stochastic Gradient Descent (SGD) or Adam optimizer which is an extension of SGD.

Output layer - The final layer of a neural network, which produces the model's predictions.

Overfitting - When the model performs better on data it has seen before (training dataset) than on data it has not seen.

Padding - A hyperparameter in a convolutional neural network that determines the amount of pixels that are added to the edge around a image. The pixel value is most commonly zeros.

Parameter values - Refers to the numerical values assigned to variables in a function, which can affect the behavior or outcome of the model.

Pixel - The smallest unit of an image, represented as a single point in a 2D array.

PyTorch - Low-level Python library for machine learning.

Receptive field - The receptive field of a neuron in a convolutional neural network (CNN) is the region of the input image that the neuron is looking at. It refers to the size of the area in the input image that contributes to the activation of a given neuron.

ReLu - ReLu (Rectified Linear Unit) is a commonly used activation function in neural networks. It is defined as f(x) = max(0,x), which means that it outputs the input if it is positive, and zero otherwise.

ResNet - ResNet (Residual Network) is a type of neural network architecture that was designed to address the vanishing gradient problem in deep neural networks. It uses skip connections and residual blocks to enable the gradient to flow more easily through the network, allowing for deeper architectures.

Residual Block - A residual block is a building block used in ResNet architectures. It consists of two or more convolutional layers, and uses a skip connection to add the input to the output of the block.

Residual parts - A component used to build a ResNet. The component consists of 2 or more CNN layers. The input and output of this component are added together so that during back-propagation, the gradient can ”jump” over steps.

Root index - Root index is the symbol used to indicate what root is being applied. For square root the root index is 2, most time not even written.

Scalar product - The scalar product, also known as the dot product or inner product, is a mathematical operation that takes two vectors and returns a single number. It is defined as the sum of the products of the corresponding elements of the two vectors.

Scheduler - PyTorch implements ways to adjust the learning rate based on the number of epochs. Using a scheduler is a good way to prevent the model from becoming worse. Gamma is a hyperparameter used to determine how much the learning rate should decrease.

Sigmoid - Sigmoid is an activation function commonly used in neural networks. It is defined as f(x) = 1 / (1 + exp(-x)), and outputs values between 0 and 1. It is often used in the output layer of binary classification problems.

Skip connection - A skip connection, also known as a shortcut connection or identity mapping, is a way of connecting the input of a layer to the output of a layer that is not directly adjacent. It is used to address the vanishing gradient problem in deep neural networks.

Softmax - Softmax is an activation function commonly used in the output layer of a neural network for multiclass classification problems. It takes a vector of values and outputs a vector of probabilities that add up to 1.

Standard deviation - Standard deviation is a mathematical function that on average tells how far each value lies from the mean.

Stochastic Gradient Descent - Stochastic Gradient Descent

Stride - A hyperparameter in a convolutional neural network that determines the step size of the filter kernel as it slides along the input volume. Stride value is usually set to 1 by default, which means the filter slides 1 pixel at a time.

Summing function - A summing function is a mathematical operation that takes two or more numbers as input and returns their sum.

The vanishing gradient problem - Refers to the problem of the gradient becoming smaller and smaller as it propagates through many layers of a deep neural network, which can make it difficult to train the network.

Transfer Learning - Transfer learning is a technique in machine learning where a model trained on one task is used as a starting point for training a model on a different task. It is commonly used in deep learning to leverage pre-trained models on large datasets to improve performance on smaller datasets.

Traditional fully connected network - A traditional fully connected network, also known as a feedforward neural network, is a type of neural network architecture where the input is passed through a series of layers of fully connected neurons. Each neuron in a layer is connected to every neuron in the previous layer, and the output of each neuron is passed through an activation function.

Training process - The training process in a neural network involves feeding input data through the network, computing a loss function that measures how well the model is performing, and using an optimization algorithm to update the weights of the model to minimize the loss function.

Underfitting - Underfitting occurs when a model is not complex enough to capture the patterns in the data, and as a result, has poor performance on both the training and test data.

Weight - A parameter value in a function that represents the strength of a connection between two nodes. This value can be a negative or positive. During the training process, the weights are updated to minimize the loss function and improve the accuracy of the model. Proper initialization of weights is crucial for the performance of a neural network.

Acronyms

ATLAS A Toroidal LHC ApparatuS

CERN European Council for Nuclear Research

LHC Large Hadron Collider

API Application programming interface

CNN Convolutional neural network

MBH Microscopic black hole

ResNet Residual Network

SPH Sphalerons

1 Introduction

In order to provide a comprehensive understanding of the thesis, this chapter serves as an introduction to the background and context of the project. The chapter introduces the idea behind the thesis, as well as introducing the project owners that have established the problem statement. Moreover, this chapter presents the structure of the thesis, with a brief introduction to every chapter.

1.1 Context

The Large Hadron Collider (LHC), situated at the Organization for Nuclear Research (CERN) outside of Geneva, Switzerland, is currently the worlds largest and most powerful particle accelerator. Along with its state-of-the-art technology and groundbreaking research capabilities, it serves a crucial asset to scientific research (Rossi, 2016). With a circumference of 27 kilometers, the LHC is capable of accelerating particles to nearly the speed of light (Jones, 2023).

The LHC is a scientific tool that helps us explore the secrets of the universe and expand our understanding of the fundamental building blocks of matter and the laws of physics. The discovery of the Higgs boson, a particle that gives mass to other particles (Jones, 2023), is among the LHC's most significant achievements.

On July 5th 2022, the announcement was made regarding the commencement of Run 3 for the LHC. During Run 3, the LHC will be colliding proton beams at a world-breaking-record of 13.6 trillion electronvolts (TeV) – 6.8 TeV per beam, for nearly four years (CERN, 2022). With higher beam energy and intensity during Run 3, the ATLAS experiment will have the opportunity to push the boundaries of its physics research to new heights (ATLAS-Collaboration, 2022).

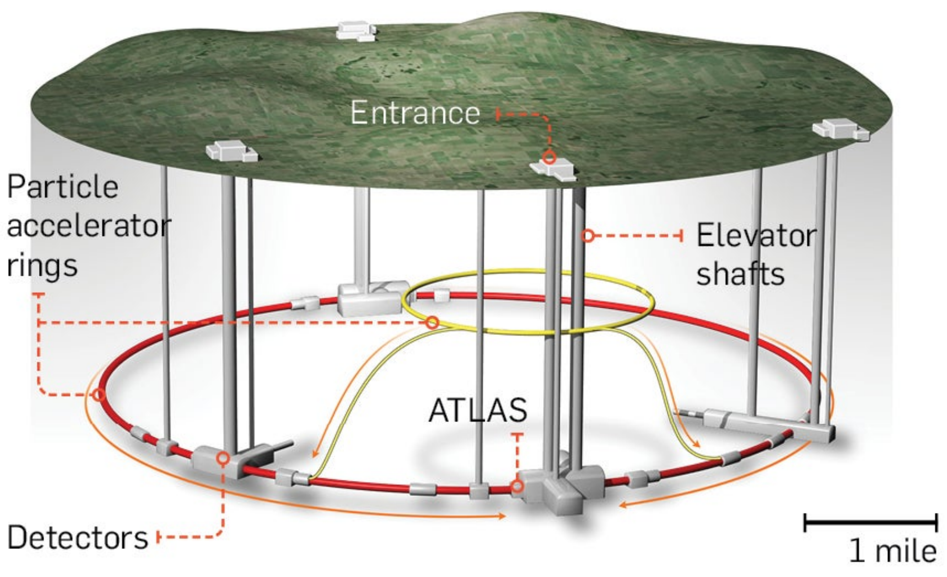


Figure 1.1: LHC - Schematic layout of the Large Hadron Collider (LHC) underground, as well as the location of the ATLAS detector (Popsci, 2015).

Among the four detectors at LHC, the ATLAS detector (A Toroidal LHC ApparatuS) is the world's largest general-purpose particle detector. The ATLAS detector has the shape of a cylinder, measuring 44 meters long and 25 meters in diameter (Li1, 2017). The detector consists of six different detecting subsystems (ATLAS-Experiment, n.d), that measures particles that are produced in proton-proton collisions.

When accelerating two beams of protons in the LHC, proton-proton collisions are intended to occur in the detectors. High-energy particle collisions generates a new state with a short lifetime. This state decays into a scattering of new particles. The scattering forms jetstreams that the ATLAS detector aim to detect, and the ATLAS experiment aim to analyse and research (May, 2022).

Deep learning provides a solution to the ever more complex analysis required for extracting valuable information from collision data. Hence, the implementation of deep learning approaches has become increasingly in favour for enhancing the performance of tasks such as jet classification (ATLAS-Collaboration, 2022).

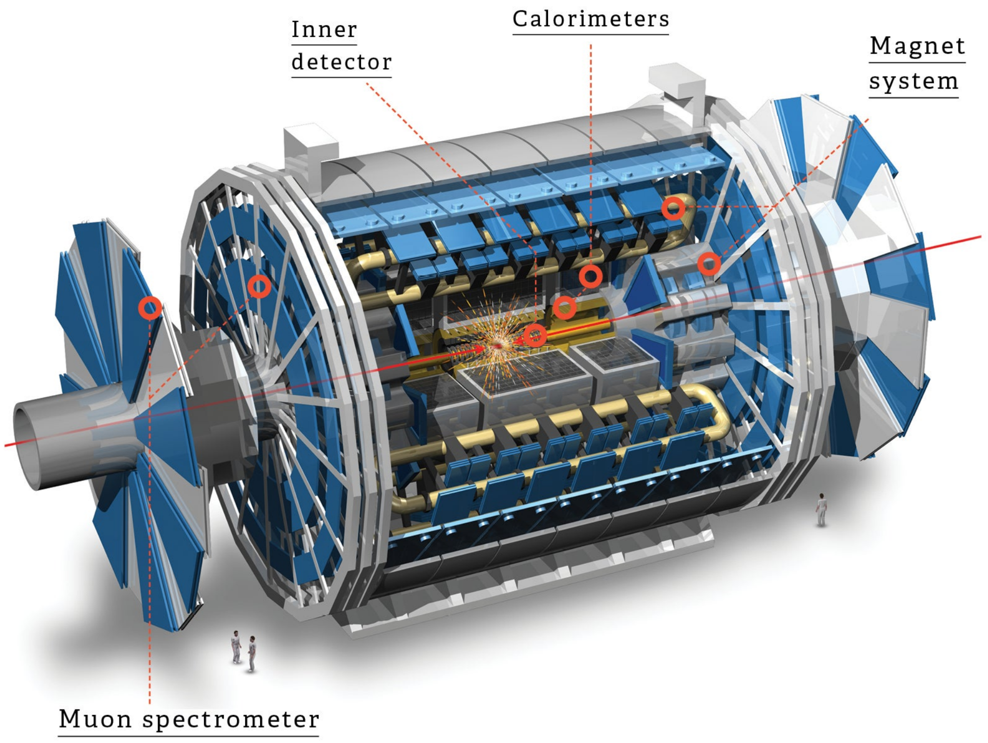


Figure 1.2: ATLAS detector - Layout of the ATLAS detector. Size comparison of the detector with humans to scale (Popsci, 2015).

1.2 Project Owner

The ATLAS group is a research group at Western Norway University of Applied Sciences (HVL) in collaboration with CERN. Their goal is to expand scientific knowledge about one of the most significant and complex questions in modern physics: *What is Dark Matter?* (HVL-ATLAS-Group, n.d)

The group consists of four associate professors, one postdoctoral researcher, two PhD students, and multiple master students. The project group has received guidance from two external advisors, Associate Professors Trygve Buanes and Steffen Mæland, throughout the project.

The project group has worked closely with three representatives from the ATLAS group, each of whom has expertise relevant to guiding the project. Trygve Buanes has expertise in particle physics data analysis, Steffen Mæland in applications of machine learning, and Aurora Singstad Grefsrud in the domain and is the developer behind the 2D histogram data used in the project (HVL-ATLAS-Group, n.d).

1.3 Structure

Chapter 1 - Introduction: The chapter refers to the project's topic, as well as introducing the project owners and the structure of the report.

Chapter 2 – Theoretical Background: The chapter explains theoretical concepts within physics and deep learning, as well as explains the use of data, and why it is in the form of a 2D histogram.

Chapter 3 – Project Description: The chapter refers to what the project is about by referring to practical background and previous work. Motivation, goals, and possible solutions are explained, as well as the literature and resources used to achieve the results, and any limitations regarding the project.

Chapter 4 – Project Setup: The chapter refers to the proposed solutions and discusses which solutions to use for the final result. It also refers to the selection of technological tools and methodologies used during the development process.

Chapter 5 – Detailed Solution: The chapter discusses the project's final solution and the approach taken to achieve it, and providing an in-depth description of the solution.

Chapter 6 – Results: This chapter examine the solution's outcomes through a close inspection of data and graphs, while analyzing and discussing both successful aspects and areas that didn't perform as expected.

Chapter 7 – Conclusion: This chapter wraps up the overall project development, addressing the problem statement and summarizing the final results.

Chapter 8 – Suggestion for further research: This chapter discusses potential work for further research aiming to improve and optimize the results.

2 Theoretical Background

This chapter introduces current theory of particle collisions that occur at the ATLAS detector, as well as the production of microscopic black holes and sphalerons. Furthermore, the chapter provides a detailed explanation of how a artificial neural network is trained, and the dataset that will be utilized to train the deep learning model.

2.1 Particle Physics

2.1.1 Particle Collision in ATLAS Detector

The LHC enables the acceleration of two beams of protons to nearly the speed of light. These beams collide with each other within a controlled environment, such as the ATLAS detector, Figure 2.1 shows an official depiction of particle collision in the ATLAS detector. The collisions aim to recreate the conditions that existed in the early universe, shortly after the Big Bang (Cern., n.d*b*). Physicists are able to identify the particles that emerge from a collision by studying their momentum, energy, and other properties (Cern., n.d*a*).

Microscopic black holes and sphalerons are not particles in the conventional sense, but rather short-lived phenomena that may occur during a particle collision. The ATLAS experiment aims to identify these phenomena which are theorized to happen during high-energy collisions (The-CMS-Collaboration, 2018).

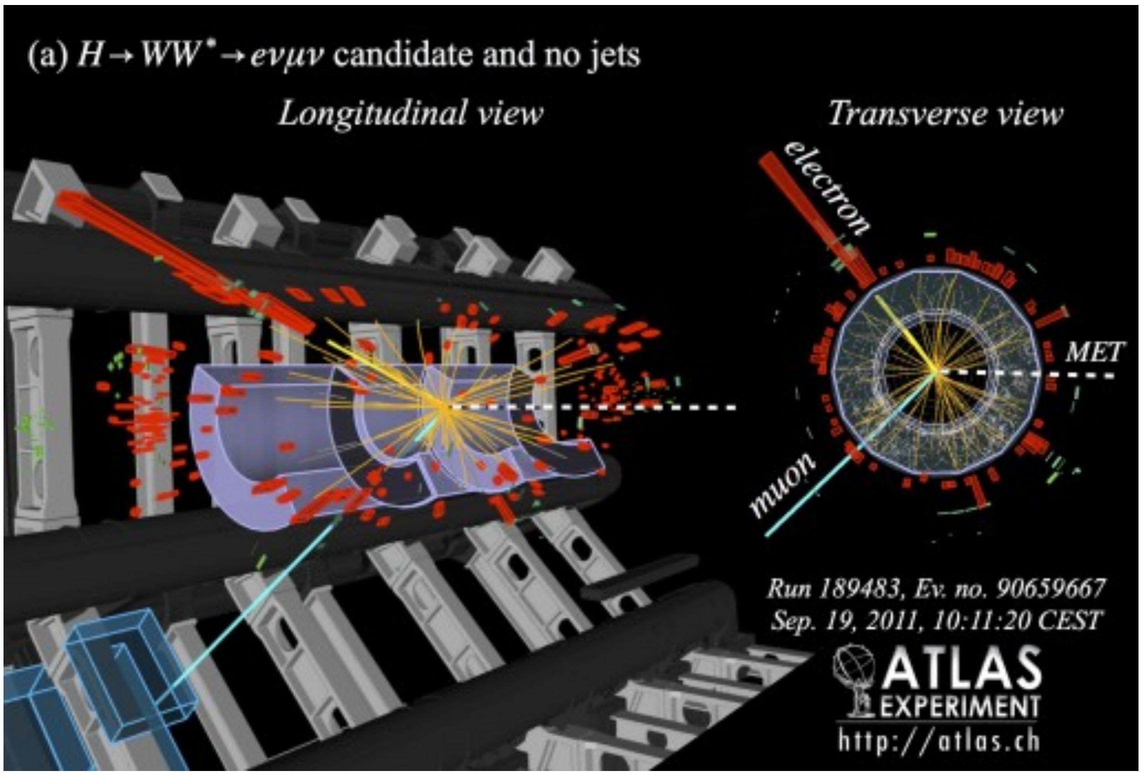

(a) $H \rightarrow WW^* \rightarrow e\nu\mu\nu$ candidate and no jets
Longitudinal view
Transverse view
electron
MET
muon
Run 189483, Ev. no. 90659667
Sep. 19, 2011, 10:11:20 CEST
ATLAS EXPERIMENT
http://atlas.ch

Figure 2.1: Particle Collision - Visualization of Particle Collision in ATLAS detector (CERN, 2023).

2.1.2 Jet detection

When high-energy proton-proton collisions occur at the LHC, a large fraction of the resulting particles tend to cluster together into groups, known as jets. The ATLAS detector is designed to measure the energy and properties of particles produced in these collisions, which in turn allows for the detection and study of jets. The detector holds a series of calorimeters that are able to detect the energy of particles as they interact with the detector material. When particles deposit energy in the calorimeters, they create clusters of energy deposits that are referred to as topologically-related energy deposits. Jets are observed as groups of topologically-related energy deposits in the ATLAS calorimeters. (OpenData-ATLAS, n.d)

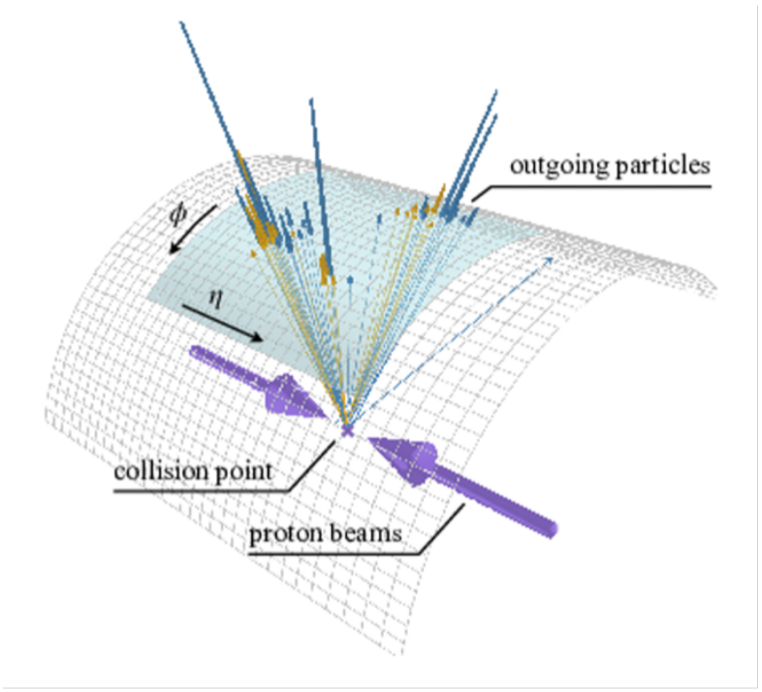


Figure 2.2: Jet Formation 1 - Illustration of jet formations after a proton-proton collision in the ATLAS detector (Qu et al., 2022).

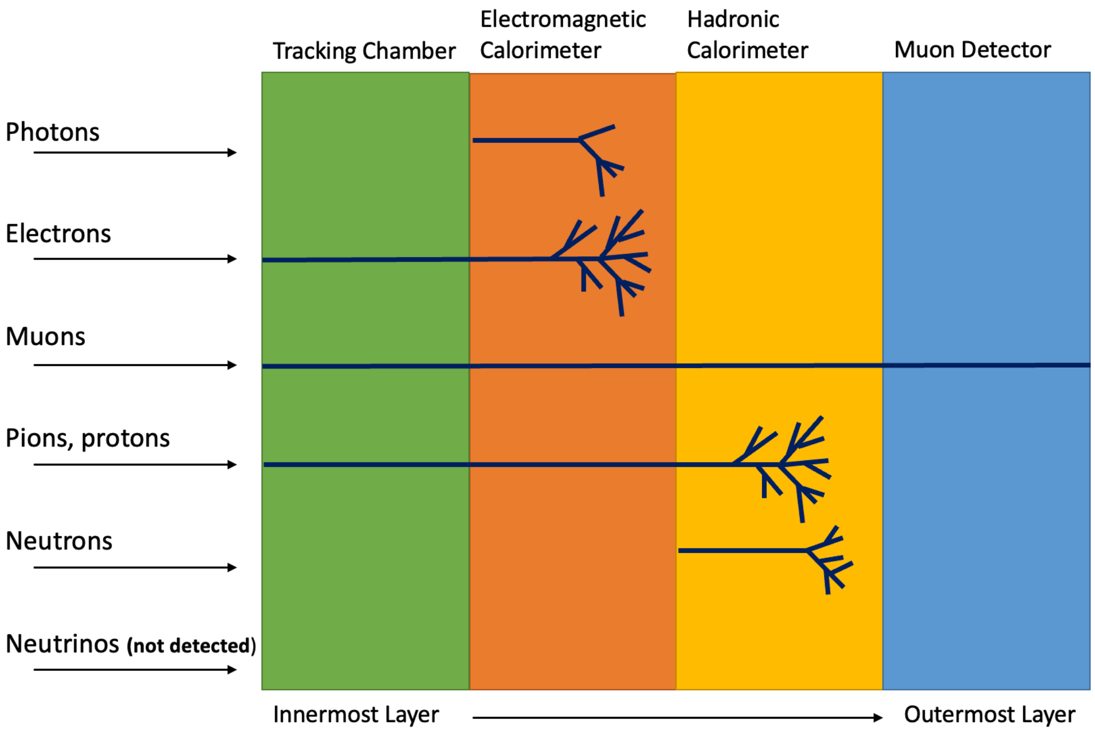


Figure 2.3: Detectors in ATLAS - Figure illustrating how different particles react with the detector layers. (Figure inspired by Amram and Etzion (2023)

2.1.3 Hadronization

Hadronization is a process that occurs shortly after a collision, following the decay of a particle. In this process, new elementary particles are formed around the original particle, resulting in a spread of hadrons that move together. The hadronization process is over long before the particles reach the detector. Hence, analytically calculating hadronization is very complex (Webber, 1999). For this reason, Monte Carlo simulated data is used for research. Monte Carlo simulation will be presented in more detail in Section 2.5.1. Hadronization is significant for researchers to assess the state that occurred after a collision and before a jet was formed.

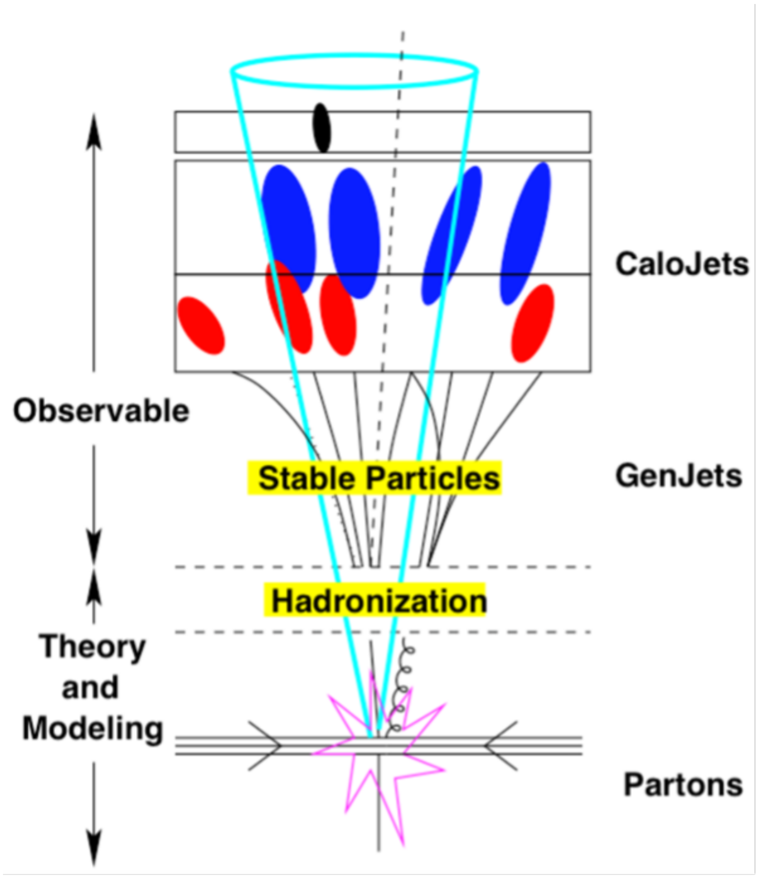


Figure 2.4: Jet Formation 2 - The figure illustrates the processes after a proton-proton collision, and how the jets are formed. The particles are observable right after the hadronization process (Sumida, 2011).

2.1.4 Microscopic Black Holes

In 1971, Stephen Hawking predicted that microscopic black holes could have formed after the Big Bang (Pinochet, 2019). This theory has not been proven. With the help of the LHC, CERN aims to produce microscopic black holes during a proton-proton collision. Production requires a energy density that is sufficient so that no repulsive forces can balance the gravitational attraction. If microscopic black holes are produced in the LHC, they will have a very short lifetime due to Hawking radiation, a phenomenon where black holes emit particles and lose mass, eventually decaying. This process results in a distinct spread of subatomic particles that form jet streams. If the existence of microscopic black holes is confirmed, scientists will likely be able to confirm the theory of the existence of multiple unseen dimensions in the universe. Furthermore, scientists will be closer to an answer as to why gravity appears weak in comparison to other fundamental forces (The-CMS-Collaboration, 2018).

2.1.5 Sphaleron

A sphaleron is a hypothetical phenomenon that may have occurred during the beginning of the universe, just after the Big Bang. LHC recreates the conditions that existed in the early universe, as mentioned in Section 2.1.1. For this reason, sphalerons may possibly arise in high-energy proton-proton collisions in the ATLAS detector.

A Sphaleron is an energy transition that occurs between two different vacuum states. A vacuum state is a state where energy is at its lowest (Xu, 2022). Simply put, a sphaleron is a process that can convert elementary particles into other types of elementary particles. Sphalerons are associated with saddle points as shown in Figure 2.5, and can be interpreted as the top of a transition where energy is at its highest (Chaudhuri and Khlopov, 2021).

The existence of sphalerons has not been confirmed, but they play a significant role in the study of particle physics and the understanding of the universe by answering the question of why there is more matter than antimatter in the universe.

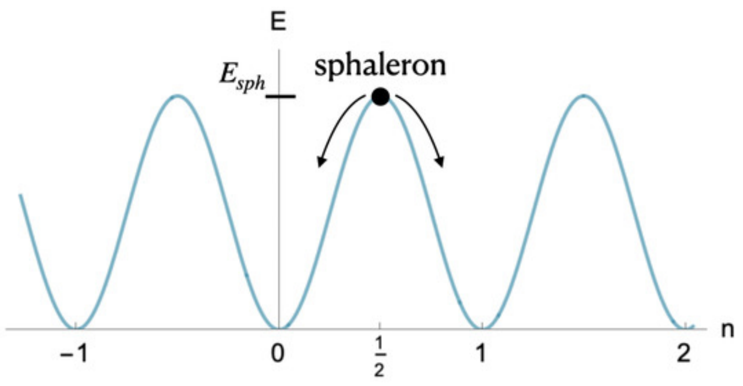


Figure 2.5: Sphaleron - The figure shows the saddlepoint of the the sphaleron, which exists between two vacua of equal energy (Gannouji, 2022).

Microscopic black holes and sphalerons are anticipated to show similar signatures within the detector, making their classification a non-trivial task.

2.2 Machine Learning

Machine learning refers to a broad set of techniques that enable computer systems to learn from data and improve their performance on specific tasks without being explicitly programmed. In essence, machine learning enables systems to learn from experience and adapt to new condition, making it a powerful tool for solving complex problems across a variety of domains.

2.2.1 Machine Learning Techniques

Machine learning covers a variety of techniques that are suited to different types of tasks. The most common techniques are supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, models are trained on labeled data to make predictions on new, unlabeled data. Unsupervised learning, on the other hand, involves finding patterns and relationships in the data without the use of labeled data. Reinforcement learning is a type of learning where an agent interacts with an environment to learn how to perform a task. The Figure 2.6 provided below shows what type of tasks the different techniques can do. The figure provides an overview of the different machine learning techniques and the types of tasks each one can accomplish (Janiesch et al., 2021).

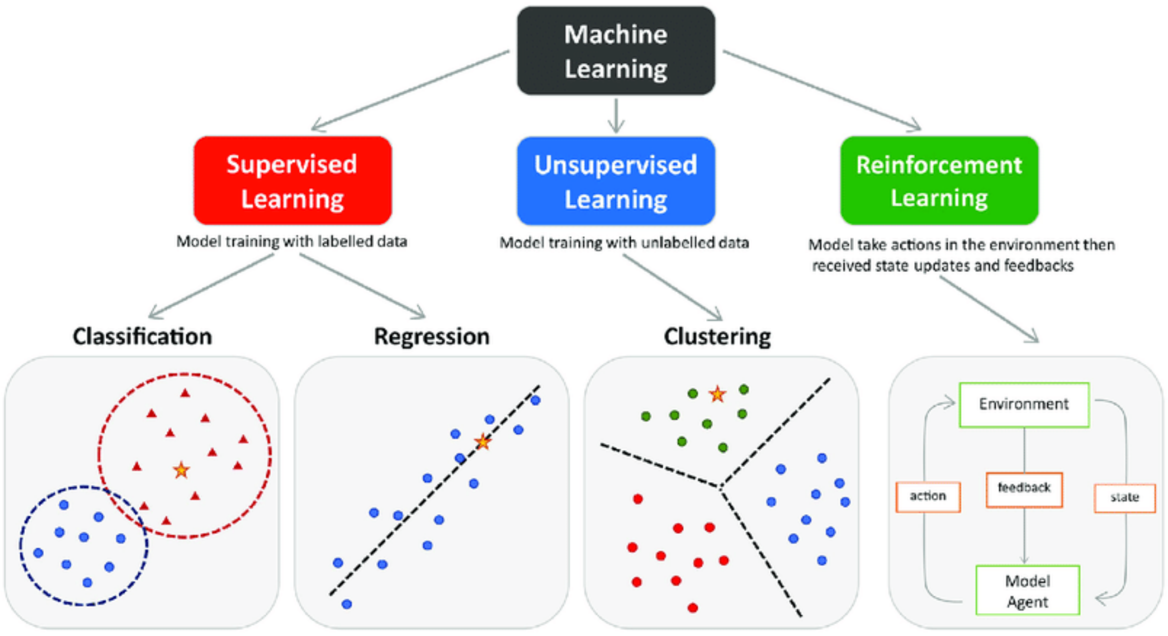


Figure 2.6: Machine Learning Techniques - Coloured dots and triangles represent the training data. Yellow stars represent the new data which can be predicted by the trained mode (Peng et al., 2021).

This project utilizes supervised learning as the data provided is labeled and the task involves classification between sphalerons and microscopic black holes. In addition, transfer learning is also employed in this project, which allows a model trained on one task to be adapted to perform a related task. Further details on supervised learning and transfer learning and are provided in the following sections, Section 2.2.2 and Section 2.2.3.

2.2.2 Supervised learning

Image classification is typically approached as a supervised learning problem, meaning that it aims to categorize images into different classes by assigning accurate labels. Supervised learning is a method used to train a model on labeled data, extract the features, and learn what is typical for that specific image or object in the image. When testing the model on unlabeled data, it is expected that the model has previously learned the patterns and can correctly make predictions (Cunningham et al., 2008).

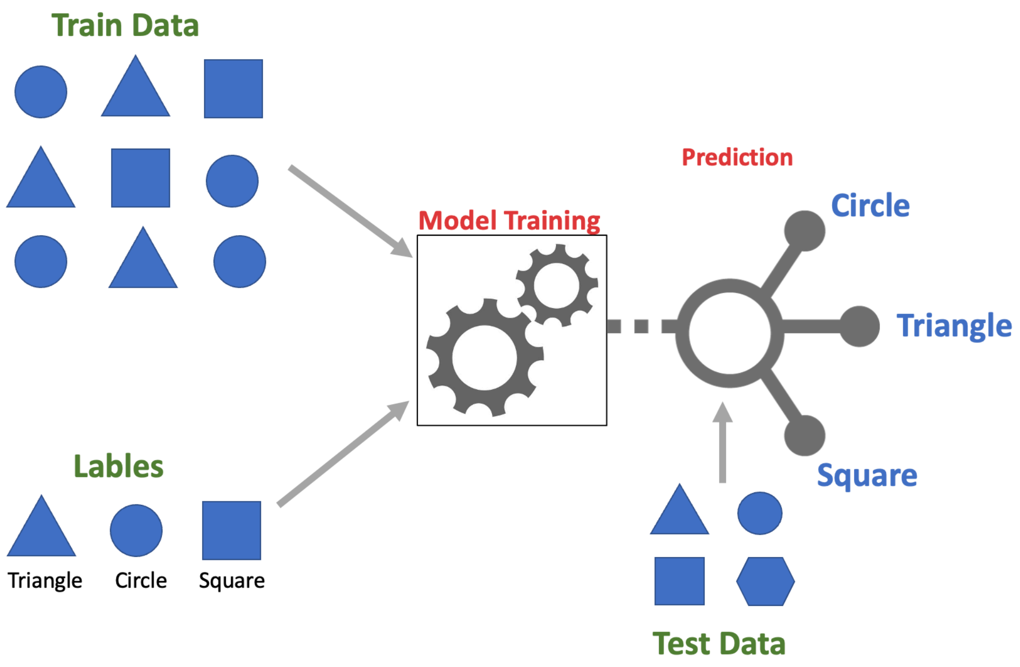


Figure 2.7: Supervised Learning - The figure shows the process of training a model using labeled data and making predictions. It's important to note that the test data is inputted to the model solely for prediction purposes to check the accuracy of the model, and is not to be used for training.

2.2.3 Transfer learning

Transfer learning involves taking a model that has already been trained on a particular task and repurposing it for another related task. The idea behind transfer learning is that many tasks have shared features and patterns in their data, and these can be learned by a model trained on a related task. By using a pre-trained model as a starting point, a significant amount of time can be saved, as well as computational resources that would have been required to train a new model from scratch (Kandel and Castelli, 2020).

When initializing a model all the weights and connections are given random values. By giving the model values that are generalized for any task, training would be faster as all the patterns are already found by the model. This is often done by training a model on a much larger dataset with multiple classes, freezing all previous layers so they do not change under backwards propagation and adding a fully connected layer with desired output layer. An example is using ImageNet 4.1.4.

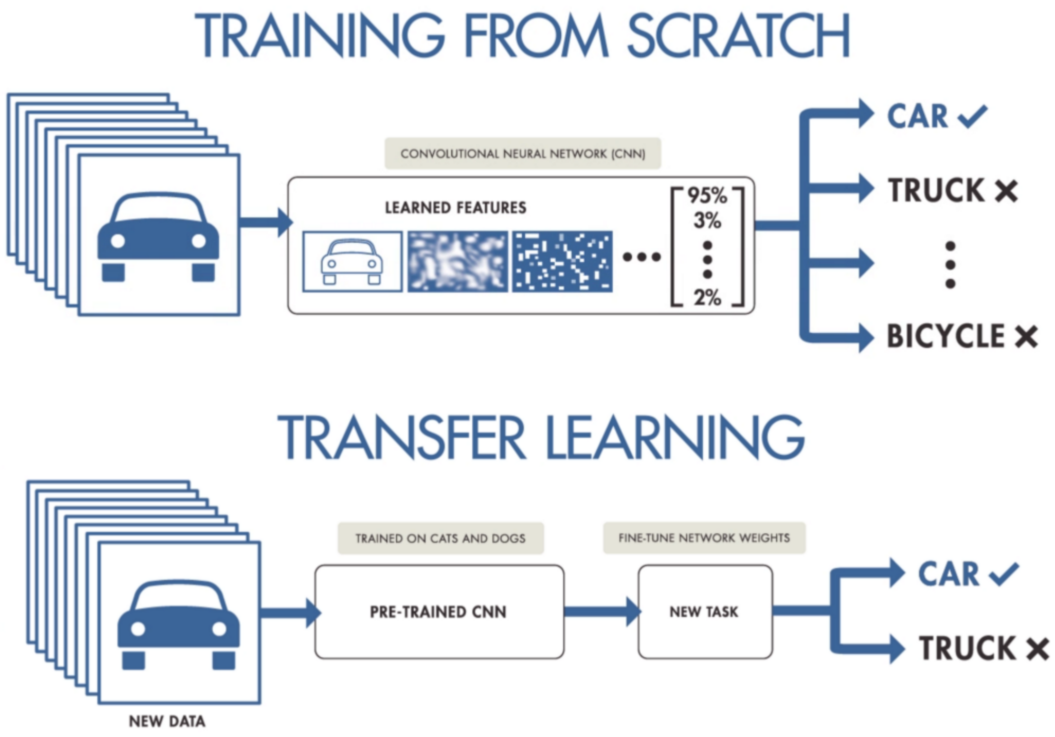


Figure 2.8: Transfer Learning - The figure shows the difference between training a model from scratch and utilizing a transfer learning techniques by making predictions on a pre-trained model (MathWorks, 2021*a*).

2.2.4 Conceptual Distinction

In order to provide a comprehension of the broad domain of Machine Learning, a hierarchical relationship among the terms - Machine Learning algorithms, Artificial Neural Network, and Deep Neural Network - is illustrated through a Figure 2.9 below.

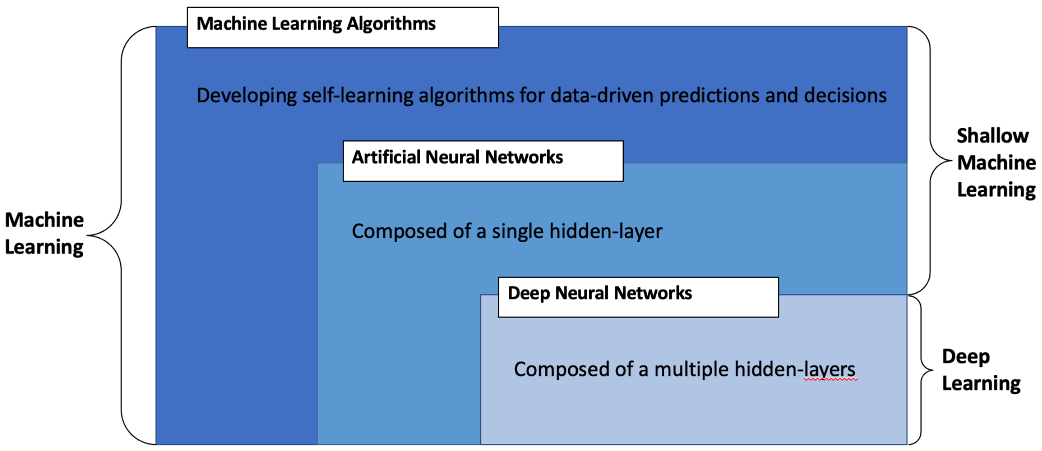

Machine Learning Algorithms
Developing self-learning algorithms for data-driven predictions and decisions
Artificial Neural Networks
Composed of a single hidden-layer
Deep Neural Networks
Composed of a multiple hidden-layers
Machine Learning
Shallow Machine Learning
Deep Learning

Figure 2.9: Venn Diagram - The Venn diagram representing the machine learning domain shows that deep learning and neural networks are subsets of machine learning. The diagram also indicates the complexity of these areas within the larger field of machine learning (Janiesch et al., 2021)

As illustrated in the Figure 2.9, Artificial Neural Networks (ANNs) and Deep Neural Networks (DNNs) are subset of Machine Learning (ML), yet they are more specialized types of architectures within this field. Traditional machine learning algorithms, such as linear regression and decision trees, have a separate feature extraction step before the model starts the training process. On the contrary, ANNs perform the feature extraction step implicitly during the model training process in the hidden layer of the architecture. In this layer, the model learns to extract features from the input data that are relevant for the task at hand.

However, both the machine learning algorithm approach and the ANN approach are referred to as ”shallow machine learning,” which indicates their simple design and limited ability to learn complex features. Therefore, they may not be sufficient for more challenging tasks.DNNs solve this problem by adding multiple layers in the hidden layer, allowing for more complex feature extraction. The feature extraction is hierarchical, where lower layers learn to extract simple, low-level features, and higher layers learn to combine and abstract these lower-level features into more complex and high-level representations. The following Section 2.3 and Section 2.4, provides a deeper understanding of how these architectures operate.

2.3 Artificial Neural Network

As previously discussed, a neural network is a subset of machine learning, and the term 'artificial neural network' refers to a network composed of an input layer, hidden layer(s), and an output layer. While a basic neural network consists of a single hidden layer, numerous other architectures delve beyond this simple structure, adding complexity and depth to the hidden layer. In this section, the fundamental parts that apply to all neural networks will be discussed.

2.3.1 Structure of Artificial Neural Networks

Artificial neural networks has a complex structure. The model is inspired by the human brain, particularly the neurons found within it, hence the name "neural network" In computer science, "neurons" are known as "nodes." The neurons are organized into three layers: input layer, hidden layer, and the output layer. Connections between the neurons create a network that enables information to move through the system. (Howard and Gugger, 2020)

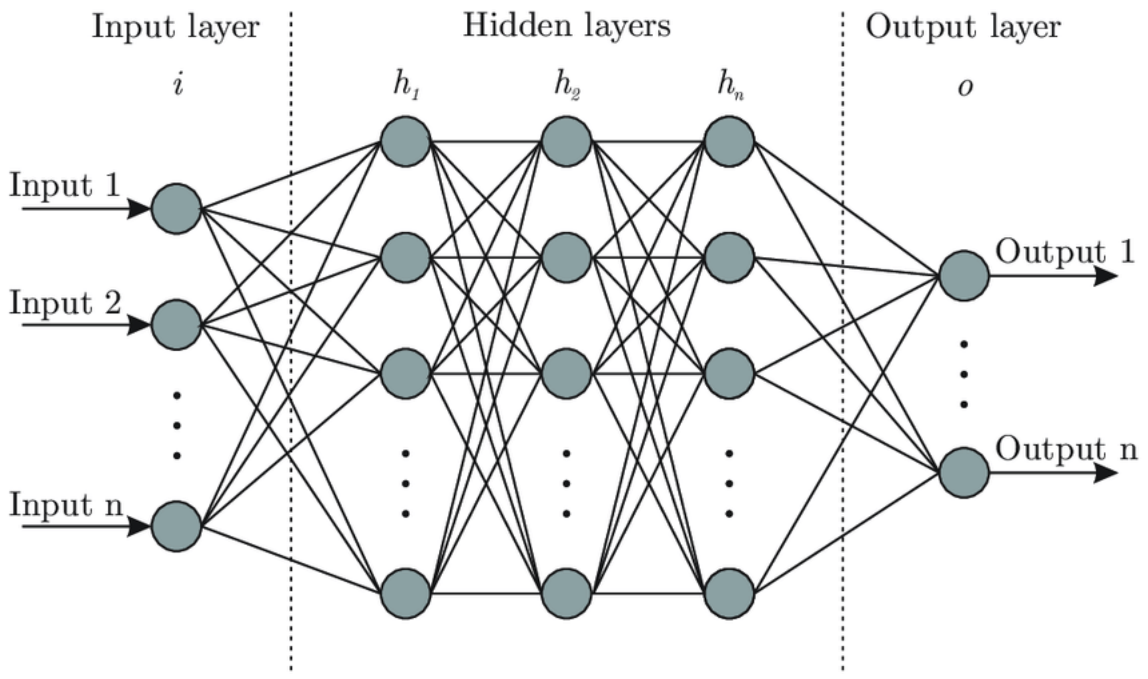


Figure 2.10: Fully Connected Neural Network (FCN) - A simple five layered fully connected neural network, comprised of a input layer, thee hidden layers and an output layer (Sidiya and Li, 2020).

A neural network consists of a single input layer and output layer, and one or more hidden layers (Howard and Gugger, 2020). As depicted in Figure 2.11, a neuron within a neural network receives a set of input values $(x_1, x_2, \ldots, x_m)$ either directly from the dataset if it is within the input layer, or from a previous neuron if it is within the hidden- or output layer.

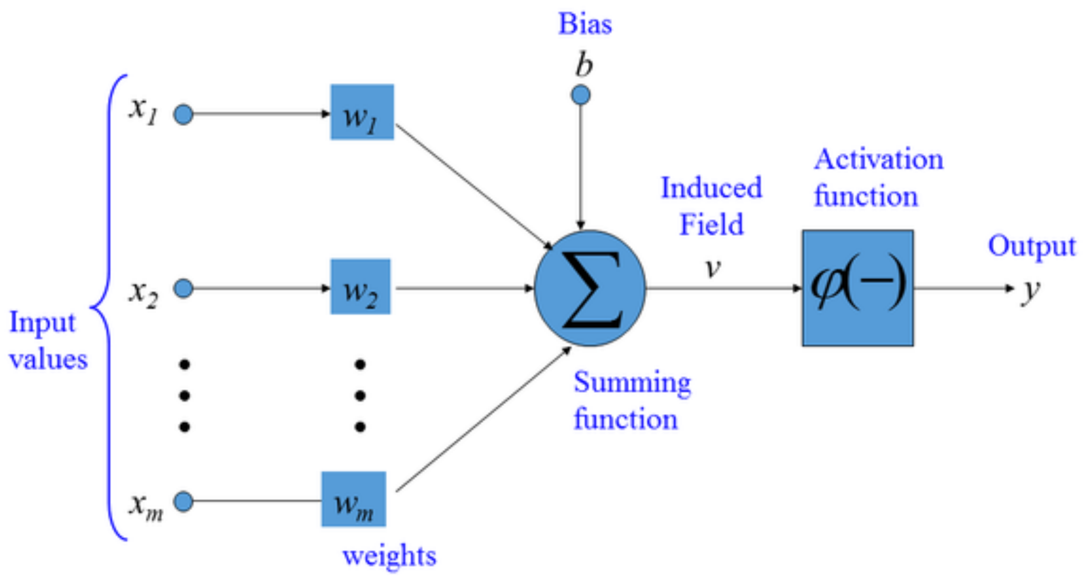


Figure 2.11: An Artificial Neuron - The figure depicts the structure of an artificial neuron that processes input data, and forwards output data. Composed of a set of weights assigned to each connection, a bias term that is added to the weighted sum of inputs, and an activation function that maps the output (Kaya and Baştemur Kaya, 2021).

The value of the model parameters is determined during the training process. The parameter values are referred to as weights and biases in a connection. The weight $(w_1, w_2, \ldots, w_m)$ represents the strength of a connection between two nodes and is usually a value between -1 and 1. Bias (b) is a constant value added to the summing function. This value is used to adjust the output of each node in the network. The activation function is essential in avoiding a linear output as it allows the network to make non-linear decisions. This is crucial in preventing the network from limiting itself to linear patterns and ensures that it is capable of capturing all the various types of patterns that may arise within the dataset (O'Shea and Nash, 2015).

Common activation functions for classification:

$$Relu(z) = max(0, z)$$

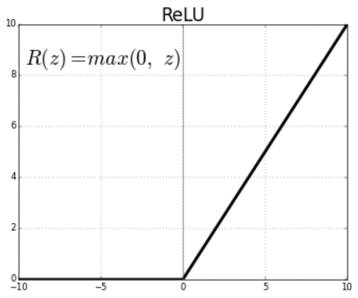


Figure 2.12: ReLU applies to hidden layer for better computation performance (Vieira and Paixao, 2019)

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

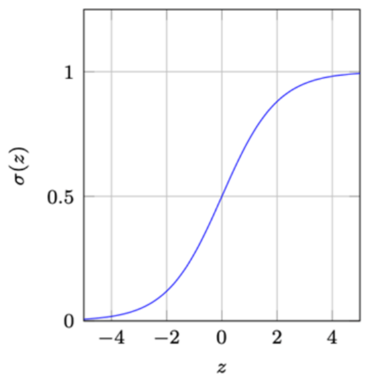


Figure 2.13: Sigmoid applies to binary classification methods (Vieira and Paixao, 2019).

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K$$

Softmax applies to multiclass problems, and is an extension of sigmoid. Softmax used for a single variable input would be equal to sigmoid function on Figure 2.13. The function parameter z_i is a vector containing the model output for a batch. The output of the function is a vector containg the model estimates.

2.3.2 Hyperparameter Optimization

The goal of training an artificial neural network is to find optimal model parameter values that minimize the loss value, so the network can give accurate predictions. In contrast to model parameters, hyperparameters in a neural network are determined before the training process and remain unchanged. Hyperparameters are adjusted by changing their value or by using different functions to achieve a model with optimal performance. It is the hyperparameter values that determine the value of the model parameters (Howard and Gugger, 2020).

There are several hyperparameters that can be adjusted and tested to improve the performance of training the neural network. In this section, different hyperparameters will be reviewed to gain a good understanding of their function and role in the network.

Epoch In one epoch, the neural network is trained on the entire training data set. The data is used exactly once. An epoch is completed when the data has been moved forward, and Backpropagation 2.3.4 is performed to adjust the weights.

Batch In an epoch, the training data is divided into batches to avoid problems that may arise due to limited storage space on a computer . When all batches have trained the neural network, an epoch is completed.

Optimizer Hyperparameters can also be functions. There are different optimization functions to choose from. The goal of an optimization function is to adjust the weights and learning rate to achieve a good combination of values that contribute to reducing the loss value.

Learning Rate The speed at which the neural network learns is defined by a learning rate value. This hyperparameter is a parameter in an optimization function. It controls how much the optimization function can adjust the weight with respect to the gradient. The smaller the value, the slower it will move towards the global minimum or local minimum. Too large of a learning rate can result in the model parameters updating too quickly, causing the loss function to diverge. It can also result in overfitting to the training data, causing the model to perform poorly on new, unseen data.

In order to optimize hyperparameters, various algorithms can be employed. Among the most common hyperparameter optimization techniques are Grid Search, Random Search, and Genetic Algorithm. For this particular project, Grid Search is utilized. Grid Search makes a complete search over a given subset of the hyperparameters space of the training algorithm, Figure 2.14 shows an example of how it would look like. A range of possible parameters are manually set, and the algorithm makes a complete search over them. Essentially, the grid search algorithm utilizes a brute-force approach, which can result in extended execution times (Liashchynskyi and Liashchynskyi, 2019).

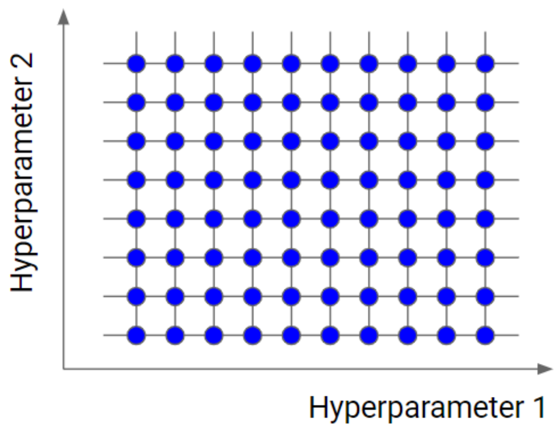


Figure 2.14: Grid Search - Figure shows what combinations of two hyperparameters would be used in grid search (Teacher, 2021).

2.3.3 Loss Function

In the final layer of the network, the predicted output is evaluated to determine its difference with the target output. This evaluation is accomplished by employing a loss function, which calculates the difference between the predicted output and the target output. The purpose of the loss function is to express how far off the computed output is from the target value. The goal is to achieve the lowest possible loss value, indicating a close match between the predicted output and the target outputs. By employing backpropagation, the model parameters are adjusted to minimize the loss.

The standard choice of loss function for a classification task, is the combination of softmax with cross entropy (Ruby et al., 2020). In binary classification, where C = 2, the Figure 2.15 below illustrates the pipeline from the output value to the loss value:

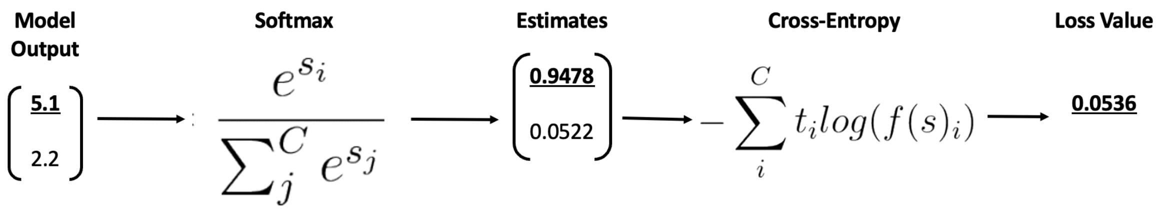


Figure 2.15: Softmax and Cross-Entropy - The figure is an example of a binary classification pipeline from the model output to the loss value.

In Figure 2.15, the model generates output values for both MBH (highlighted in bold) and SPH. The highest value among them indicates the predicted label, which in this case is MBH. However, both of these values are then passed through the softmax function. Only the MBH value is fed into the cross-entropy function, as the model knows this based on the target label. The cross-entropy function calculates the difference between the predicted value (MBH) and the target value, resulting in a loss value. The loss value is propagated backward through the network until all the weights are updated.

For binary classification problems the cross entropy uses the sigmoid or softmax activation function to map input values to a range between 0 and 1. The loss increases when the predicted estimation for the true label decreases. If the loss increases, it indicates that the current image classification model is performing poorly and can be characterized as a poor classification model. On the other hand a loss value near zero indicates that the model has done an almost perfect prediction.

As described in Section 4.4.4, cross-entropy will be used as a comparison metric when evaluating different loss functions for the thesis. Additionally, a custom loss function was developed in this thesis, which is discussed further in Section 5.2.1. This custom function takes inspiration from zero-one loss and will be used to compare its performance against cross-entropy loss function in the evaluation process.

2.3.4 Training the Model

In the first epoch (epoch 0), the neural network is initialized with random values, unless the model is pre-trained, then it is initialized with the weights from the pre-trained model, which is a common practice in Transfer Learning 2.2.3. The input data is passed through the network in a process called forward propagation. During forward propagation, each layer of the network applies a set of mathematical transformations to the input data. This process involves multiplying the input by a set of weights, adding a bias term, and applying an activation function. The output of one layer serves as the input for the next layer, and the process is repeated until the final layer produces the network's output. As these initial weights are random, the predicted output is also essentially random.

Furthermore, the loss function calculates the difference between the predicted value and the correct value. The goal of the training process is to minimise the loss between these two values, to achieve better and more accurate predictions.

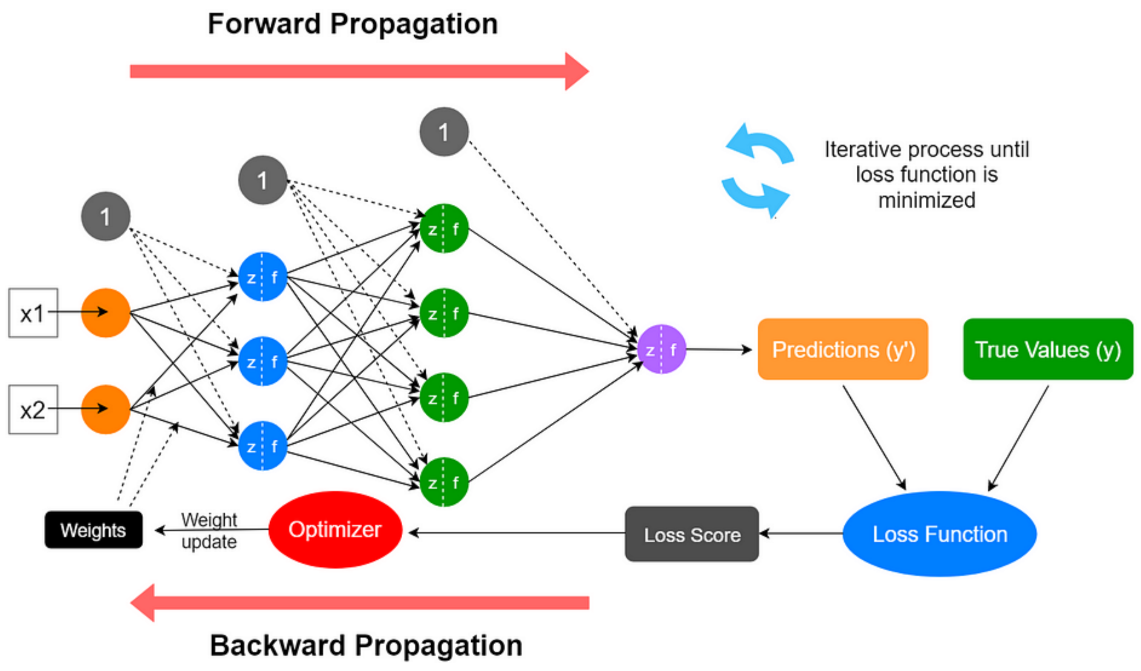


Figure 2.16: FP and BP - The figure illustartes the process of forward propagation and back-propagation, which involves forwarding input data through a neural network to produce an output, computing the loss, and then backpropagating gradients through the network to update its parameters. This process is repeated for each epoch until the loss is minimized (Pramoditha, 2021).

To minimize the loss, an algorithm called backpropagation is used. Backpropagation is able to calculate the gradient of the weights in the network with respect to the loss function during training. During this process, the optimizer is the significant hyperparameter that updates the weight and learning rate with respect to the gradient. The gradient represents the direction and magnitude of the change required to minimize the loss. Once the gradients are computed, an optimizer algorithm, such as Stochastic Gradient Descent (SGD) or Adam optimizer, updates the weights of the network. The learning rate determines the step size of the weight updates.

Backpropagation is performed in each batch, which is a subset of the dataset. The process of forward propagation, calculating the loss, and performing backpropagation is repeated for each batch until the entire dataset has been processed. The choice of the number of epochs and batch size depends on the size of the dataset, the complexity of the model, and the desired level of accuracy. These hyperparameters must be tested and adjusted during the training process to achieve optimal performance. (Howard and Gugger, 2020)

2.3.5 Overfitting and underfitting

During the process of training a model on a dataset through multiple iterations, a noticeable ”bad fit” in performance can occur, due to either overfitting or underfitting. A ”bad fit” means that the model does not accurately represent the relationship between the input data and the desired output. Overfitting occurs when a model becomes too specialized in learning the patterns and details of the training dataset, causing it to lose generalization ability and perform poorly on new, unseen data. Underfitting occurs when a model is too simplistic and fails to capture the complexity of the training dataset, leading to poor performance on both the training and test data (Biswal, 2023).

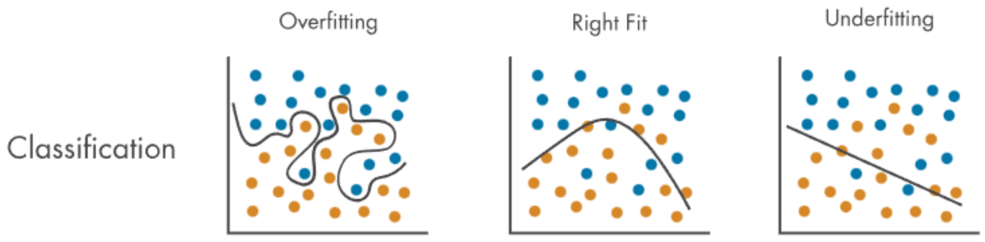


Figure 2.17: Overfitting and underfitting - Figure above shows how 3 different models could predict. An extreme overfitting, a good prediction, and an extreme underfitting (MathWorks, 2021*b*).

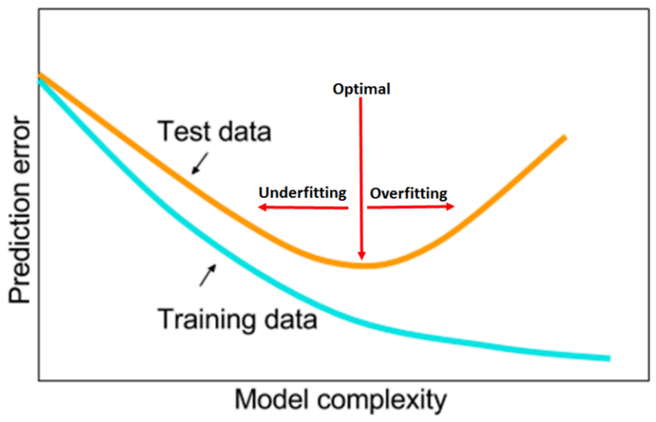


Figure 2.18: Observing overfitting - When test data has a lower performance on accuracy than train data, it is a common observation of the model overfitting (Smith, 2018).

The goal of training a machine learning model is to increase the level of generalization, meaning the ability to adapt to new unseen data. To increase generalization; overfitting and underfitting can to be reduced. Tuning Hyperparameters 2.3.2 are generally the method used to prevent overfitting, as well as regularization methods. It is important to note that even if a model performs poorly due to a bad fit, it can still have the ability to generalize well.

2.4 Deep Neural Network

In the previous section, the fundamental components of a neural network were discussed. However, this section will be explore more complex architectures that utilize multiple layers of neurons to extract more intricate features from data. The architecture of a neural network with multiple hidden layers, which is commonly referred to as a Deep Neural Network. There are several types of deep neural networks, but for this project, but this project will be centered on Convolutional Neural Networks (CNNs) and Residual Networks (ResNets).

2.4.1 CNN architecture

Convolutional Neural Network (CNN) is a subtype of a neural network that is commonly used for image classification tasks. Unlike a traditional fully connected neural network, CNN contains at least one convolution layer. The convolutional layers are the main building blocks of CNN. The layer applies a set of filters to the input image, enabling the network to detect specific features and patterns in an image independent of location. Convolutional layers are often combined with pooling layers, and fully connected layers. These additional layers help to further refine the output of the network and improve its accuracy. (O'Shea and Nash, 2015)

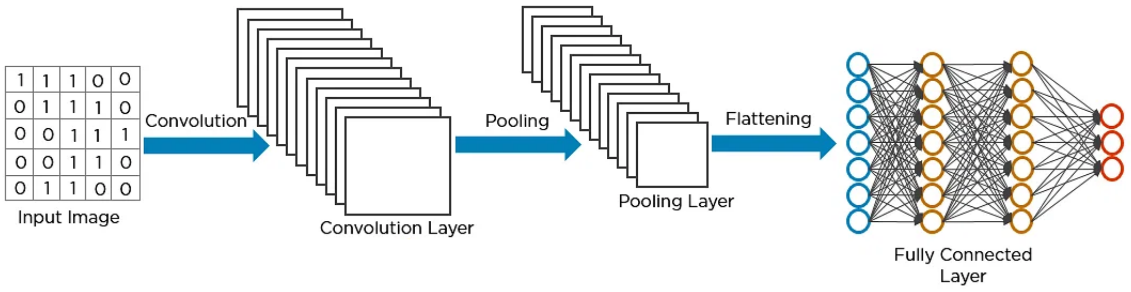


Figure 2.19: Convolutional Neural Network (CNN) - A simple CNN model comprised of a single convolution layer and pooling layer, followed by a flattening of outputs that is feed into a fully connected layer (InterviewBit, 2021).

Convolutional Neural Networks (CNNs) are generally considered to be more suitable for image classification tasks than regular neural networks. Due to the dimensionality of the input data and the number of parameters required to process full images accurately, regular neural networks do not scale well for image classification tasks. Regular neural networks treat each input feature independently, which can lead to the loss of spatial information. In contrast, the complexity of the convolutional layers, and the pooling layer of the CNN ensures the preservation of spatial information (Pan et al., 2018).

To process an image, each pixel is passed into the network. For a image size of 50x50x3 (50 wide, 50 high, 3 color channels), 50x50x3 = 7 500 input neurons must be provided. The modelparameters would multiply quickly, and would lead to overfitting. Convolutional operations solves this problem by reducing the dimentionality of the image (O'Shea and Nash, 2015).

2.4.2 Convolution Layer

Unlike an artificial neuron in a fully-connected layer, a neuron in a convolutional layer is not connected to the entire input. Instead, it only processes a section of the input data, known as the receptive field. This allows the convolutional layer to identify specific features in the input image. The use of receptive fields is a fundamental concept in CNNs and is what sets them apart from other neural network architectures (Dumoulin and Visin, 2018).

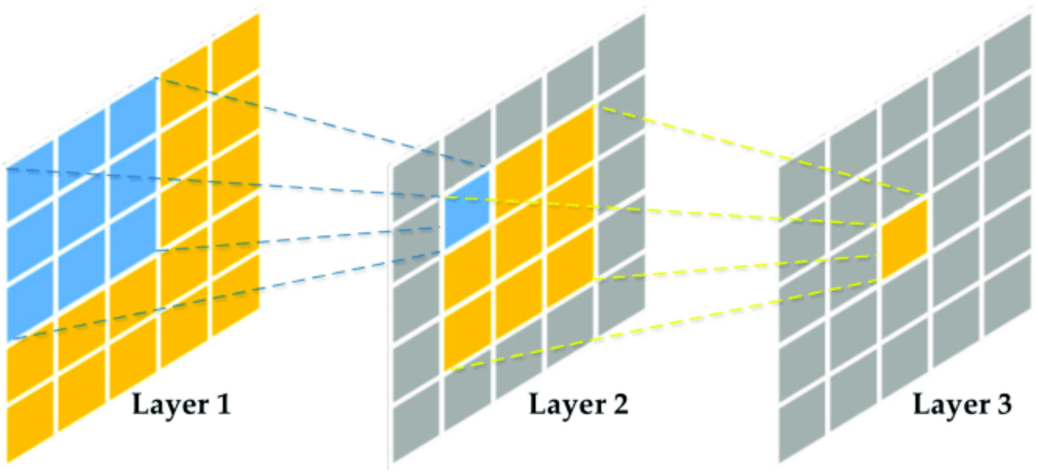


Figure 2.20: Receptive Field - Schematic diagram of the receptive field in CNNs (Xu et al., 2021).

A convolution operation is performed when a kernel K_1 is applied to the input image (I). The kernel is also referred to as the "filter". During convolutional operations, the kernel is gradually shifted one pixel at a time, ensuring that the details of the input image are preserved. The output of this operation is a single feature map. $F_1 = I * K_1$, (* = convolutional operation) (Karpathy et al., 2016).

To compute the individual values of the feature map matrix, the scalar product of the kernel with the corresponding region of the input image is taken, As depicted in Figure 2.21. For instance, if the input image is of size 4x4x3 and the kernel is of size 2x2, then the scalar product of the kernel with each 2x2 region of the input image is calculated. (Karpathy et al., 2016) This operation is repeated until the kernel has traversed the entire input image, resulting in a complete feature map filled with values (Liu et al., 2015).

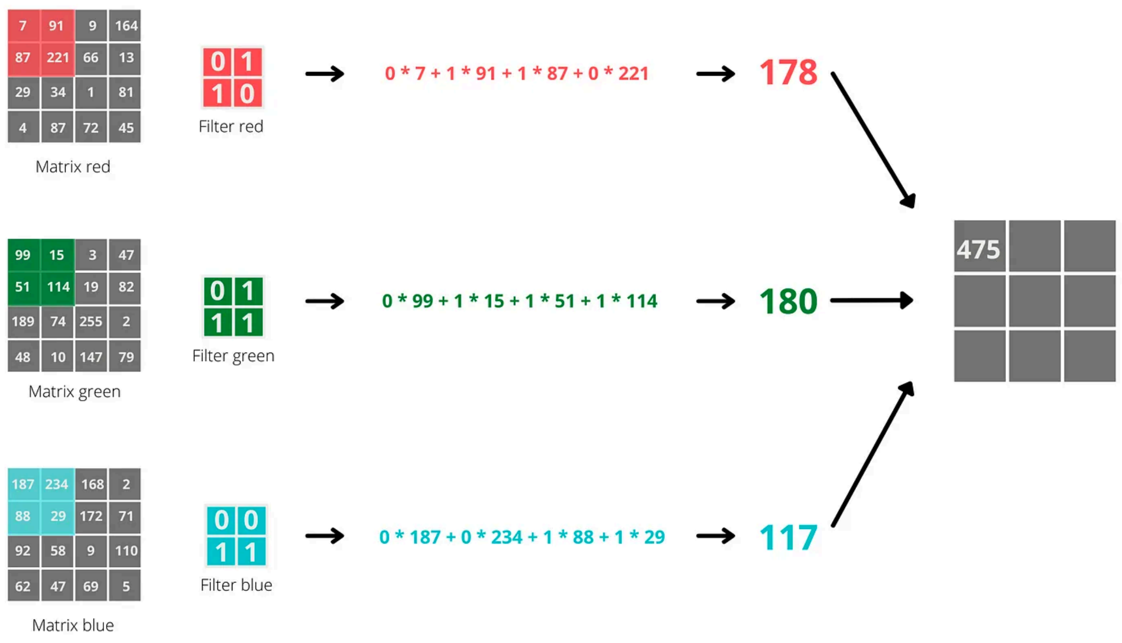


Figure 2.21: Convolution - Each layer undergoes matrix multiplication with a 2x2 kernel, and the resulting values are summed (178+180+117=475). Upon the filter/kernel traversing the entire input, a complete feature map is produced Camp (2021).

Applying the convolution operation multiple times with n different kernels will result in n feature maps. These feature maps are usually concatenated to form a block of feature maps. The total number of feature maps produced is equal to n, which corresponds to the number of kernels used to apply convolution on the input image (Karpathy et al., 2016).

2.4.3 Pooling Layer

In the pooling layer, the feature maps obtained from the convolutional layer are taken as input. The primary goal of this layer is to further reduce the dimensionality of the feature maps and detect the most significant features in the image (O'Shea and Nash, 2015).

As an example, consider the resulting feature map from the convolutional layer in Section 2.4.2, which was a 3x3 matrix. In the pooling operation, a 2x2 matrix will be generated as the output. That being the case, the feature map will be divided into all possible 2x2 submatrices.

Figure 2.22 shows the difference between max pooling and average pooling. For max pooling, the maximum value in each submatrix is selected as the output (627). In contrary, for average pooling, the average of the four values in each submatrix is calculated (546).

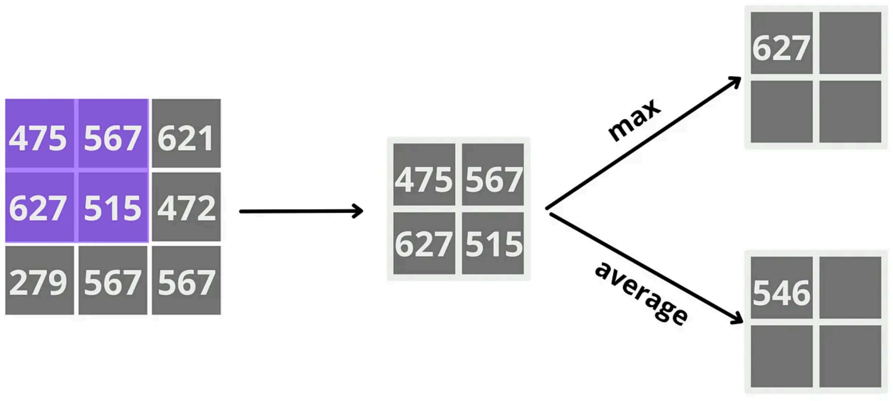


Figure 2.22: Pooling layer - Illustration of the pooling layer with either a max-pooling operation or an average-pooling operation (Camp, 2021).

Additionally, the pooling layer also filters out noise from the input image. This involves removing elements of the image that are irrelevant to the classification task, thereby improving the overall performance of the network (O'Shea and Nash, 2015).

2.4.4 Fully Connected Layer

The fully connected layer resembles the layers of a regular neural network. After the convolutional and pooling operations, the dimensionality of the input is reduced, resulting in fewer neurons and requiring fewer resources during training (O'Shea and Nash, 2015). The training is equal to the description in Section 2.3.4.

2.4.5 ResNet

The residual neural network (ResNet) architecture is based on the convolutional neural network architecture, with a focus on improving performance and efficiency by increasing the number of layers. The motivation behind this approach is that as networks become deeper, they can suffer by an issue known as the ”vanishing gradient problem.” To address this challenge, the ResNet architecture was developed, which effectively resolves the problem through the incorporation of residual connections, also referred to as skip connections. In Figure 2.24 there is shown an expample of the same ResNet model with and without residual connections. In the following section, a more comprehensive explanation will be provided.

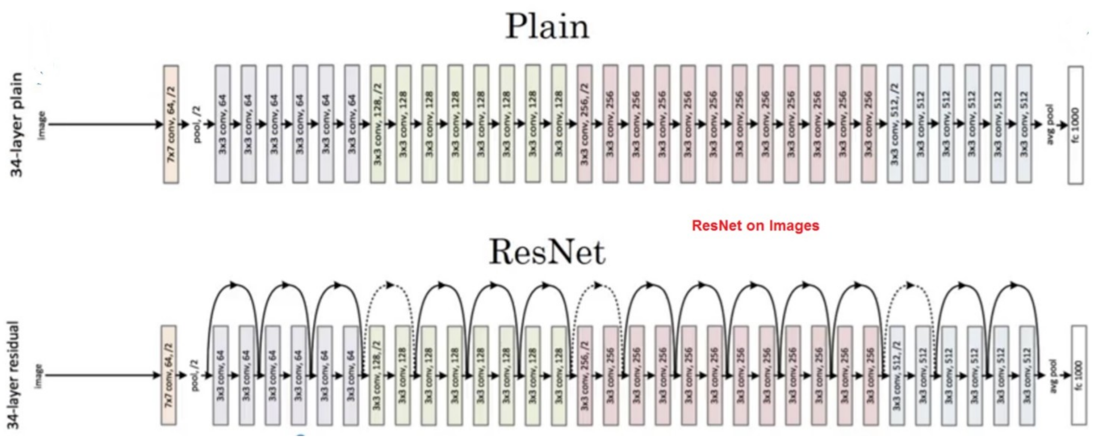


Figure 2.23: ResNet Architecture - The figure shows the architectural distinctions between a plain neural network and a ResNet architecture with residual connections (arrows).(IndoML, 2018)

Typically, increasing the depth of a neural network should result in a decrease in the error rate during training and testing as the network learns more complex features. However, due to the vanishing gradient decent problem, the error rate start to increase again. This problem also causes overfitting of the model, hence increases the error (Zhang et al., 2015).

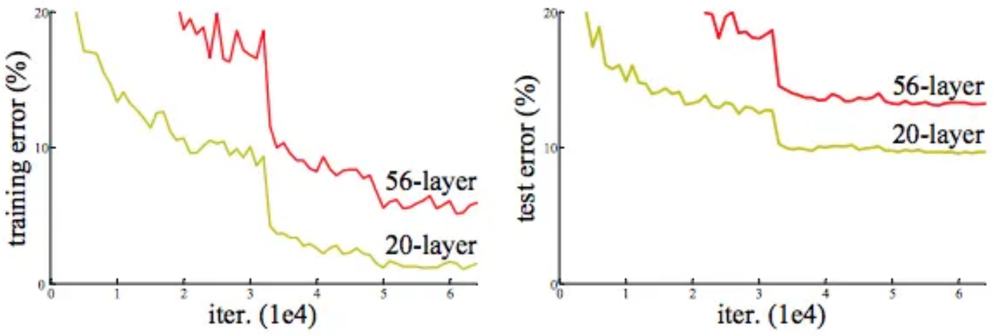


Figure 2.24: Plain Networks - Training error (left) and test error (right) with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. (Zhang et al., 2015)

Vanishing gradient problem To update the weights in the neural network during training, backpropagation must be performed. This involves calculating the gradient by taking the derivative of the loss with respect to the weights.

When using sigmoid in the output layer, the derivative ranges from 0 to 0.25. This means that the output values of the gradients will be small. This will result in only a small improvement in the weights of the network. Ultimately when the gradient vanishes, the weight will not be updated, hence the network cannot learn, resulting in decrease of performance (Huber et al., 2019).

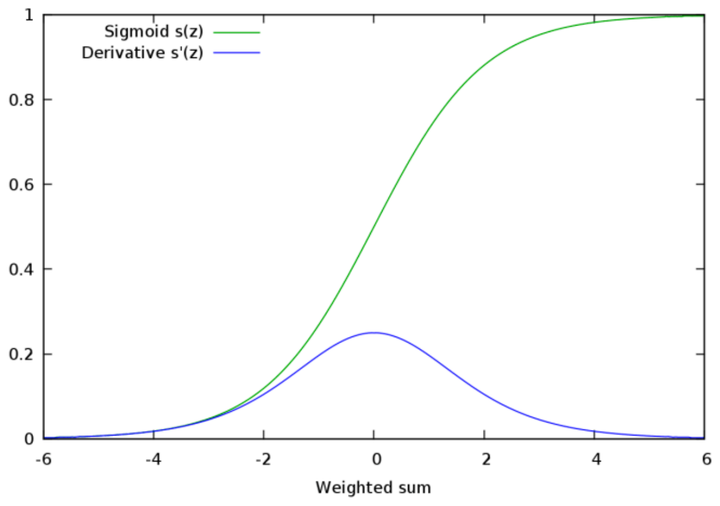


Figure 2.25: Derivative of Sigmoid - The figure depicts that the sigmoid function and its derivative have different ranges. The sigmoid function ranges from 0 to 1, whereas its derivative spans from 0 to 0.25. (Bahr, 2012)

Residual Block ResNet solves the vanishing gradient problem by using skip connections between every two layers, as well as direct connections along all the layers. One building block of such a connection is called a ”residual block.” These residual blocks are stacked on top of each other to maintain efficient learning of parameters. By using skip connections, the gradient can flow directly from the output to the input of the residual block, and bypass some layers in between, allowing for better gradient propagation. (Zagoruyko and Komodakis, 2017).

The residual block uses the parameters from the previous activation function, which allows the network to learn more efficiently by reducing the number of computations required for each layer. This helps to overcome the vanishing gradient problem by allowing the network to maintain a stronger gradient signal throughout the training process (Zagoruyko and Komodakis, 2017).

There are two main types of residual blocks: the identity block and the convolutional block. Determinating which one to use depends primarily on whether the input and output dimensions are the same or different (Dai et al., 2022).

Identity block The identity block is utilized when the input activation has the same dimensions as the output activation, and is the standard block used in ResNet. An example of an identity block is shown in Figure 2.26. (Dai et al., 2022).

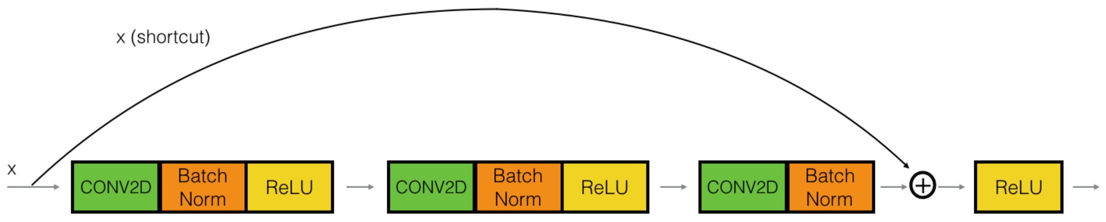


Figure 2.26: Identity Block - The figure shows that the upper path (right) is the ”shortcut path”, and the lower path is the ”main path” (left) (Balciunas, n.d.).

Convolutional block When the input and output dimensions don't match up, The convolutional block is used. In addition to handling spatial dimensions, convolutional blocks can also be used to handle changes in the number of channels or filters in the input and output. An example of a convolutional block is shown in Figure 2.27(Dai et al., 2022).

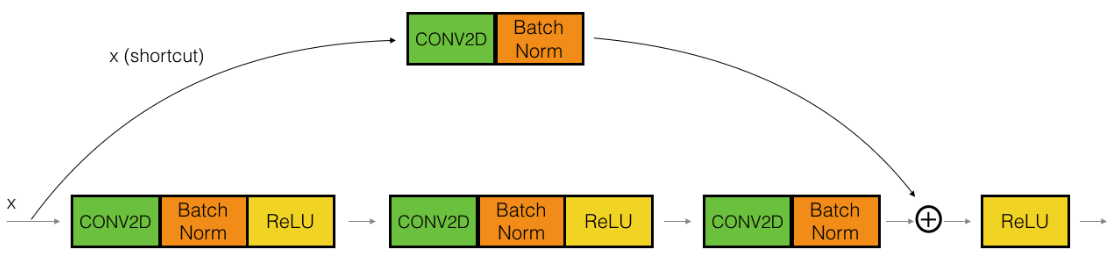


Figure 2.27: Convolutional Block - This figure despic the additinal operations in the shortcut path. The CONV2D layer in the shortcut path is used to resize the input x to a different dimension so that the dimensions match up in the final addition needed to add the shortcut value back to the main path.(Balciunas, n.d.)

Another commonly used block for deeper neural networks is the bottleneck residual block, Figure 2.28. As network depth increases, the number of parameters also increases, leading to a higher risk of overfitting and the vanishing gradient problem. The bottleneck residual block is designed to address these issues by minimizing the width of the block, while increasing its depth to reduce the overall number of parameters. This architecture of this block includes a 1x1 convolutional layer at both the input and output ends, which contributes to reduce the number of input and output channels. In the middle of the block, the 3x3 convolutional layer performes the main computation of the block (Howard et al., 2018).

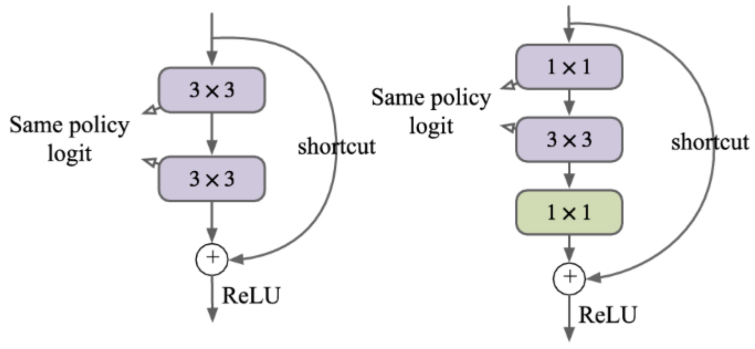
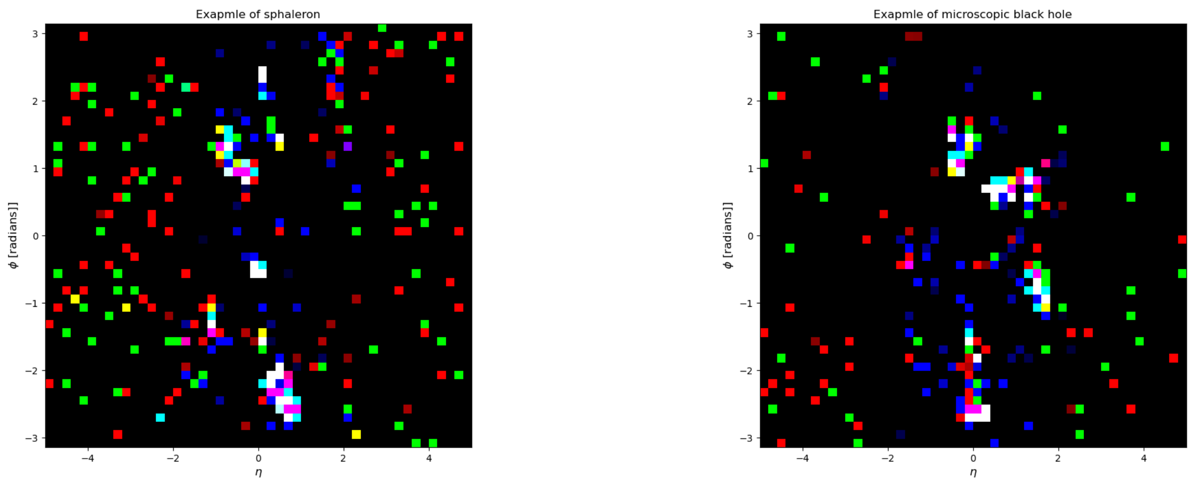


Figure 2.28: Bottleneck Residual Block - The figure depicts a side-by-side comparison between a plain residual block (left) and a bottleneck residual block (right) (Cheng et al., 2023).

2.5 Data

The ATLAS detector is in the form of a cylinder consisting of multiple layers of detector systems. The detector system measure various properties of particles that arise during a proton-proton collision. For the production of data in the form of 2D histograms, three of the detector layers are used Section 2.5.2. The histograms are rectangular pixelated images of the particles' energy after a collision, despite the detector being a cylinder. By "unfolding" the cylinder, the data can be in the form of rectangular images (Wynne, 2013).

The data is generated using Monte Carlo algorithms 2.5.1, which produce simulated proton-proton collisions (ATLAS Collaboration, 2020). The histograms are based on this data, which is analyzed using machine learning algorithms for the classification of microscopic black holes and sphalerons after a proton-proton collision. The following section will explain the use of Monte Carlo Event Generator.



Sphaleron

Microscopic Black Hole

Figure 2.29: 2D Histogram of simulated Events

2.5.1 Monte Carlo Event Generator

Due to hadronization processes that require complex analytical calculations, researchers use Monte Carlo simulations of particle collisions "events" (Webber, 1999). Long and complex calculations are replaced by experiments with random numbers, which produce simulated data of proton-proton collisions. Several simulations are produced, and the end result is the average of the number of observations. The number of simulations increases with the amount of collision data collected from the ATLAS experiment. (ATLAS Collaboration, 2020)

2.5.2 2D Histogram

A 2D histogram visualizes the distribution of data in a two-dimensional space. The data is distributed in a grid of equally sized rectangular bins. Data points fall into their respective bins based on the position of the particles. The histogram's x-axis represents the distance between the detector's center and where the particles hit, measured in Eta. The histogram's y-axis represents the detector's angle, measured in Phi. The energy of each particle is then added up with the rest of the particles that hit an area for one collision (bin box)(Scott, 2010).

The data we have been given consists of three layers of 2D histograms that are stacked on top of each other. The three layers are represented as RBG colors: red, green, and blue. The blue layer represents the innermost sensor in the ATLAS detector that measures the direction of the particles when a jet is produced, known as the ”silicon tracker.” The red layer represents the ”Electromagnetic Calorimeter” sensor in the detector that measures the energy of electrons and photons when they interact with matter. The outermost sensor in the detector is represented ”Hadron Calorimeter” by the green 2D histogram layer. This layer measures the energy of hadrons. These layers is shown in Figure 2.30.

Figure 2.29 contains an image representation of the histogram with the three color channels. The result shows the particles' positions relative to each other. One can identify areas of high or low particle intensity. High particle intensity refers to a jet.

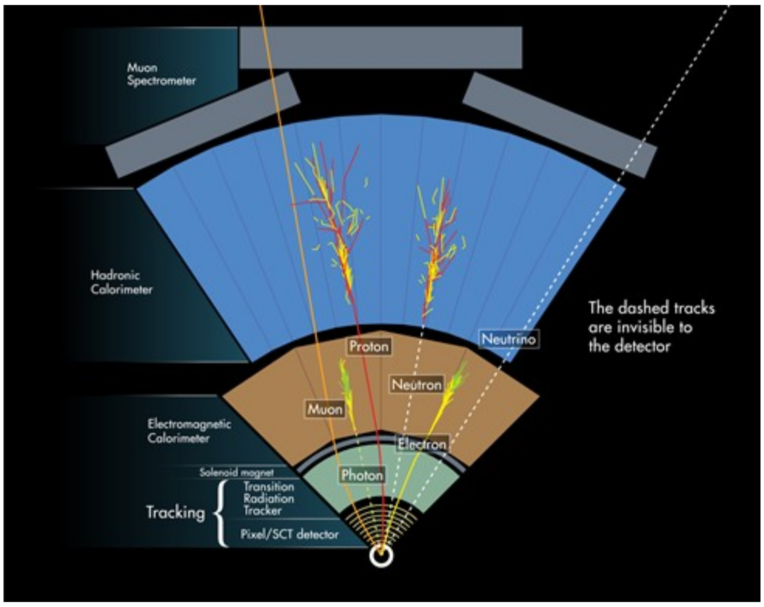


Figure 2.30: Schematic layout of the detector components in the ATLAS detector - The detector layers corresponds to RBG layers in the 2D Histograms. The blue layer is the Hadron Calorimeter. The orange layer is Electromagnetic Calorimeter. And the inner green layer is silicon tracker (Virdee, 2012).

3 Project Description

This chapter presents the problem statement and initial goals of the project, along with the resources and initial solutions that will be utilized to achieve them. Moreover, various relevant literature and earlier work are explored to identify what is already known and what is yet to be explored in this field of study. Furthermore, potential challenges and limitations for the project are acknowledged.

3.1 Earlier Work

Conventionally, collision data has been stored in tabular formats as numerical values. In this project, the data is in the form of 2D histograms that show the energy levels of particles in the detector. 2D histograms have been used in a previous research report for the ATLAS experiment:'End-to-End Physics Event Classification with CMS Open Data.' (Andrews et al., 2020). The project will build upon the approach presented in the research report, which utilizes 2D histograms for image classification.

3.2 Motivation and Goals

The project group has been provided with simulated data, in the form of 2D histograms, that portrays high-energy particle collisions. The dataset was created by a representative from the ATLAS group at Western University of Applied Science, based on theoretical understanding of how collision data directly from the ATLAS detector would appear. The ATLAS group has tasked the project group with the challenge of analyzing and classifying two distinct events that can occur during a particle collision using machine learning algorithms. These two events have many similarities, making it difficult to differentiate between them. Therefore, the project group will employ various machine learning techniques to classify these events. The problem statement for the project is to ”Explore and evaluate the performance of various neural network architectures and loss functions for accurately classifying simulated image data portraying particle collisions in the ATLAS detector." The primary objective is to develop an accurate solution that can differentiate between these two events with high accuracy, enabling researchers to perform further analysis on the resulting data.

The project will also be based on three sub-goals:

- Investigate the importance of different network architectures.
- Investigate the effect of using different loss functions.
- Investigate whether networks pre-trained on ImageNet data are useful for 2D histograms.

3.3 Initial goals and initial solution idea

The ATLAS group has tasked the project group to enhance the sensitivity of the analysis of the ATLAS detector. The task is achieved by utilizing machine learning techniques to classify collision data based on information from the entire detector or parts of it. The provided data for analysis consists of simulated particle collisions resulting from either microscopic black holes or sphalerons.

To accurately classify collisions, the project group will explore different machine learning models. These models will be implemented using Convolutional Neural Networks (CNN) and Residual Networks (ResNet). Furthermore, the ATLAS group expects the project group to train the best-performing models from each architecture on imageNet data. Subsequently, the pre-trained imageNet models will be compared with the project group's own models, which are trained on ImageNet data.

3.4 Resources

Resourses utilized in this project, demonstrate the depth of research, as well as acknowleding the contributions of other researchers, advisors, and tools that assisted with the project.

The research paper 'End-to-End Physics Event Classification with CMS Open Data' is a essential resource for this project, as it provides techniques and processes required to construct and train machine learning models effectively by utilizing 2D histograms.

Furthermore, the project owners served as external advisors, providing academic assistance in the fields of physics and machine learning. Throughout the project, Carsten Gunnar Helgesen provided valuable guidance and support as the internal supervisor.

Aurora Singstad Grefsrud, a PhD student in the team, is an essential resource for the project. She had developed the 2D histogram data used in the project as part of her ongoing PhD research. The project is built upon her work and utilizes the same data, providing an opportunity for collaboration and achieving a result that could also be used in her further research.

Moreover, technical tools that has been utilized saves time and resources. The project group has been granted access to a powerful computer from the masters laboratory at HVL, to ensure that the machine learning model can be executed within a reasonable timeframe.

Additionaly, various Python libraries, including fast.ai and pyTorch, enables the project group to develop and run machine learning models effectively, and to analyze the data and produce accurate results.

3.5 Literature

To understand the field of study for this project, relevant research paper for the problem statement has ben explored. By reviewing relevant literature, the research of this project can contribute on the field and identify potential gaps.

The research paper 'End-to-End Physics Event Classification with CMS Open Data' (Andrews et al., 2020) mentioned in Section 3.1 describes a new way of performing image classification on high-energy particle collisions. This research paper is of great importance to the project, as it introduces a novel machine learning approach to particle physics, as well as utilizing 2D histograms to represent data.

Additionally, the paper "Search for black holes and sphalerons in high-multiplicity final states in proton-proton collisions at $\sqrt{s} = 13$TeV (The-CMS-Collaboration, 2018) offers insightful understanding of sphalerons and microscopic black holes. Althought, the research paper provides a broader scope of physics that is required for this project, the insights gained provides an advances understanding of the fundamental properties in particle physic.

Although the research papers discussed above are based on simulated data from the CMS detector rather than the ATLAS detector used in this thesis, they still provide valuable background information.

Furthermore, the research paper “Specially designed random forest loss function for high energy physics” (Sprindys, 2022) serves as an influential source for developing a custom loss function in this project. The paper provides a comprehensive understanding of how loss functions can be applied to machine learning algorithms to improve model performance, serving as a baseline for designing a custom loss function that could potentially enhance the classification of microscopic black holes and sphalerons using deep learning.

3.6 Limitations

The project focuses on exploring a limited number of machine learning models and architecture, due to the time and resource constraints. Combining two different network architectures would be interesting to explore for this project. The project group has considered combining CNN and RNN architecture. Using RNN would require access to the root file used to generate the 2D histogram dataset. The utilization of the root file format, which contains data in a numerical tree structure, would require additional time to process, which is a significant limitation for this project.

The time limitation also presents a challenge in exploring the efficiency of pre-trained ImageNet models. One of the goals of this project is to evaluate the performance of pre-trained ImageNet models. However, due to the large size of the ImageNet dataset, training a model from scratch would require significant processing power and time. As a result, this project will only compare the performance of already pre-trained models.

Furthermore, the development of machine learning models for this project will not involve data processing techniques like data augmentation. Nevertheless, the main goal of implementing data augmentation methods is addressed in a separate collaborative thesis.

4 Project Setup

This chapter introduces various models and architectures as potential solutions to the research problem, along with the creation of custom loss functions and the use of ImageNet. The proposed solutions are then discussed and evaluated to determine their effectiveness. In addition, this chapter also covers the different tools that will be utilized throughout the project. Furthermore, the project methodology will be presented in detail to provide a comprehensive description of the research steps taken.

4.1 Proposed solution

This section aims to present an overview of the tools and techniques that are planned to be utilized in order to achieve the desired result. The desired result involves the development of a customized loss function that demonstrates significant improvements when implemented in various deep learning models. The ultimate goal is to surpass the performance of the benchmark. In the subsequent section, each component of a potential solution will be introduced, providing a comprehensive understanding of the proposed approach.

4.1.1 Fast.ai

Fast.ai is a deep learning library developed on top of the PyTorch deep learning framework. The library features a high-level API that simplifies the process of building and training deep learning models. Fast.ai also provides a range of pre-trained models that can be utilized for tasks such as image classification. Utilizing pre-trained models can save a significant amount of time, as developers can build on top of existing models instead of creating them from scratch (Howard and Thomas, n.d).

4.1.2 CNN-architecture

Convolutional neural network (CNN) is a network architecture within artificial neural networks, suitable for analyzing and recognizing image objects that involve processing of pixel data. In-depth understanding of the various layers in CNN and their functions is discussed are Section 2.4.1.

4.1.3 ResNet-architecture

Residual network (ResNet) is a CNN architecture that supports a larger number of convolutional layers in its network. This architecture solves the ”vanishing gradient” problem that occurs when the neural network grows. The difference between traditional CNN and ResNet architectures will be discussed in Section 4.1.6.

4.1.4 ImageNet

ImageNet is a database that contains 1.2 million training images of 1000 different classes (ImageNet, n.d). These classes can be anything from animals, plants, and objects. ImageNet is used as the dataset for training and evaluating classification models/algorithms. In Section 4.1.6, the relevant use of ImageNet for the project will be discussed.

4.1.5 Applications of loss functions

Cross-entropy is a commonly used loss function in machine learning that measures the difference between the predicted probability distribution and the true probability distribution of a classification task. It penalizes high-confidence predictions that are incorrect more heavily than low-confidence predictions that are incorrect, and low-confidence predictions that are correct. The goal of cross-entropy is to predict the correct class with high confidence, which makes it an effective approach for classification tasks. By minimizing the cross-entropy loss, the model learns to make more accurate predictions and becomes better at classifying new unseen data. In Section 4.1.6, the application of custom loss functions in relation to cross-entropy will be further discussed.

4.1.6 Discussion of alternatives

CNN is an effective architecture for image classification as it utilizes layers that learn to extract relevant features from input images. This is in contrast to traditional machine learning models, which require a separate feature engineering step to process data.

Despite the effectiveness of CNN in image classification, neither CNN nor traditional neural networks are well-suited for scaling to a larger number of layers. This is due to the connections between all layers can cause the gradient that is back-propagated to become very small, resulting in limited performance. This problem is commonly referred to as the ”vanishing gradient problem”

ResNet addresses the vanishing gradient problem by skipping some connections, reusing the activation function of previous layers, and compressing the network into fewer layers. The remaining layers, called ”residual parts,” are used when the network is retrained. Fast.ai offers a selection of pre-trained models as mentioned in Section 4.1.1, including pre-trained ResNet models, making it a powerful tool for the project.

ResNet is foreseen to be the ideal solution for the research project, and is intended to be utilized either through Fast.ai, or by creating custom models in the ResNet architecture.

Creating custom loss functions will require comparing the results with a benchmark. Cross-entropy will serve as the standard loss function, with the goal of surpassing its results.

The last sub-goal of the project Section 3.2 explore the potential use of pre-trained models on ImageNet data to determine whether the pattern under training recurs in common objects. The ATLAS group has expressed an interest in utilizing models pre-trained models on ImageNet data for classifying microscopic black holes and sphalerons. For this reason, models pre-trained on ImageNet will be used.

4.2 Selected Solution

In order to find the most suitable model for analyzing high-energy particle data that is presented in the form of 2D histograms, a thorough comparison will be studied between various models. The selection of the most appropriate solution will be based on the outcomes that each model delivers. The chosen solution will be the model that achieves the highest level of accuracy in producing optimal results.

4.3 Selection of Tools

The following section provides an overview of the technological tools and services that have been utilized during the development process, along with a brief description of their relevant use in this project.

4.3.1 LaTeX

LaTeX is a typesetting system that is especially suited for creating research papers that feature complex mathematical equations, symbols, and code (*The LaTeX Project*, n.d.).

4.3.2 Python

In the context of deep learning and artificial neural networks, Python is the most common programming language. Python offers a large number of available libraries and frameworks such as fast.ai, PyTorch, TensorFlow. These libraries provide high-level abstractions for creating and training deep neural networks.

4.3.3 Python Notebook

Python notebook is a dynamic programming environment that allows integration of Python code snippets and combines them with explanatory text and graphs in a single document. It has a user-friendly interface for practicing machine learning algorithms and techniques, which makes it ideal for iterative testing.

4.3.4 Visual Studio Code (VS Code)

Visual Studio Code is a text editor for programming languages. VS Code has many available extensions that support tasks related to deep learning, such as code completion, debugging, and data visualization.

4.3.5 GitHub

GitHub is an online platform for collaborative software development. GitHub users can share and store projects with each other, as well as track changes in the project. It makes it easy to share code between project members and different workstations.

4.3.6 WandB

WandB is a website for hyperparameter testing of machine learning models. WandB makes it easier for users to perform hyperparameter testing and compare models.

4.4 Project methodology

This section provides a comprehensive overview of the development process employed in this project by introducing the methodology. This subsequent sections includes the project plan presented in the form of a Gantt chart, a risk assessment to identify and reduce potential challenges, and an evaluation plan to ensure the achievement of project goals.

4.4.1 Development methodology

The scientific working process is employed as the methodology to guide the project's development, providing flexible frameworks for collaboration, communication, testing, and continuous improvement. The project is broken down into iterations, with each iteration comprising a cycle of planning, implementation, design, development, and testing (Ladders, 2022). The duration of each iteration is set at two weeks, during which the goal is to produce a machine learning model or neural network that can be evaluated and improved based on feedback from internal and external advisors.

To ensure effective communication and progress tracking, the project group holds weekly status meetings with the advisors, where feedback is requested and incorporated into the next iteration's research, hypothesis, experiments, and requirements. The advisors' input is also used to refine the report writing process. This process is repeated iteratively until the final product is completed, ensuring that the project remains on track and the final product meets all requirements.

Result
Observation
THE SCIENTIFIC METHOD
Conclusion
Question
Experiment
Hypothesis

Figure 4.1: The scientific method(Education, 2023)

The combination of a scientific working process and effective communication with advisors has resulted in a flexible development process. Through constructive discussions and collaborative efforts, the project group has been able to quickly make changes and incorporate new ideas and feedback without causing disruption to the overall development process.

During one of the weekly status meetings, it was discovered that there may be an error in the dataset. This led to a conversation with the domain expert, which was then escalated to a meeting with the entire ATLAS team. Through this collaborative effort, it was determined that there was indeed an error in the dataset, and a new dataset was provided to the project group shortly after.

This exemplifies the value of open communication and a collaborative approach, which enables the project team to identify and address issues quickly and efficiently.

4.4.2 Project plan

The project plan is presented in the form of a Gantt chart, a visual representation that captures the timeline of the project. The chart presents each task during the development process as horizontal bars, with the length of each bar corresponding to the estimated or actual duration of the task. This chart serves as an effective tool to visualize project milestones, identify any potential delays, and monitor the overall progress of the project against established timelines. The Gantt chart is included in Appendix B.15

4.4.3 Risk assessment

A risk assessment identifies and evaluates potential risks that may arise during the development process. Conducting an overview of possible risks, enables the project group to develop measures to reduce the likelihood of problems occurring and mitigate the impact if they do.

To estimate the probability and consequence of a risk, a score between 1-25 is assigned. The score for probability and consequence is then multiplied together, resulting in a risk product score that provides a quantitative assessment of the overall risk associated with a given scenario. The risk assessment is attached in Appendix B.16

4.4.4 Evaluation plan

The evaluation plan for achieving the project goals Section 3.2 involves comparing the performance of different machine learning models. To measure the performance of the models, the accuracy metric will demonstrates the model's ability to correctly classify sphalerons and microscopic black holes.

Accuracy is a metric that tells the fraction of predictions that are correctly classified. The dataset used for training the models has a class balance, meaning that the amount of MBH and SPH is equal. If the amount were not equal, accuracy would be a bad metric to use as it does not take into account classes with low frequency. The accuracy will be determined by calculating the total correctly predicted outcomes divided by the total number of outcomes.

Hyperparameter testing is a crucial step in the process of optimizing machine learning models. To enhance the performance of the machine learning model, various parameters can be fine-tuned, including loss functions and the learning rate variable. By testing and adjusting these parameters, different results can be generated and evaluated, allowing the selection of the model that produces the most accurate classifications. The ”Grid Search” method is used to test the parameter values for all possible combinations.

When running the various models, the tuning process will be constrained to the loss function. Each model will be trained using different loss functions, which will be evaluated based on their accuracy performance.

Cross entropy will be the standard loss function, serving as the benchmark for evaluating performance. The objective is to develop a custom loss function that surpasses this benchmark. The performance of the custom loss functions will be evaluated by comparing their accuracy against that of the cross-entropy. This analysis will assist in determining the optimal loss function.

The methodology discussed in Section 4.4.1 allows for frequent evaluation of the project's progress by sharing the results obtained from each iteration with the project owners and supervisors. This promotes a flexible and efficient development process with ongoing discussions for continuous improvement.

5 Detailed Solution

This chapter delves into the technical aspects of the solutions implemented for the project, presenting a detailed exploration of the choices made for the model architecture, constant hyperparameter values, and the underlying reasoning behind these decisions. It provides a comprehensive understanding of the final solution developed.

5.1 Architecture construction

The following sections explore the pipeline of the CNN architecture and ResNet architecture that were constructed. The upcoming sections will provide a detailed analysis of these architectures, highlighting their key components and functionalities.

5.1.1 Custom CNN

The pipeline of the CNN model consists of three convolutional layers, each followed by a max pooling operation. Convolutional operations enables the model to transforms the input image in order to extract features from the image, and subsequently reduce the dimensionality of the feature maps through max pooling (O'Shea and Nash, 2015). In the following table Table 5.1, the mathematical operations of how the input data is transformed as it passes through the convolutional layers, are provided.

	First Convolution Layer	**Second Convolution Layer**	**Third Convolution Layer**
Input	[50 50 50 3]	[50 24 24 16]	[50 11 11 64]
Kernel	[3 16]	[3 64]	[3 256]
Stride	[1 1]	[1 1]	[1 1]
Padding	[0 0 0 0]	[0 0 0 0]	[0 0 0 0]
Output height	(50+0+0-3)/1+1=48	(24+0+0-3)/1+1=22	(11+0+0-3)/1+1=9
Output width	(50+0+0-3)/1+1=48	(24+0+0-3)/1+1=22	(11+0+0-3)/1+1=9
Output Size	O_1 =[50 48 48 16]	O_2=[50 22 22 64]	O_3=[50 9 9 256]
Max Pool	max(O_1,2)= [50 24 24 16]	max(O_2,2)= [50 11 11 64]	max(O_3,3)= [50 3 3 256]

Table 5.1: Mathematical operations of the CNN model - The input and output dimension is in the following format: [batchsize height width channels]. The figure shows the construction of the CNN model used in this project.

Starting from the first convolutional layer, the input is comprised of RGB images with dimensions of 50x50x3, representing a stack of three layers with 50x50 pixel images. To ensure compatibility, the filter must also have three dimensions, corresponding to the red, green, and blue channels in the image ([50 50 3] $\circledast$ [3 3 3])[1].

The conv_layer with 3x3 kernel size reduces the dimensionality of the image with two pixels in height and width. To further reduce the dimensionality, max pooling is performed on the convolutional operation output. As shown in the output of the first convolutional layer, the dimensionality of the image is reduced from 50x50 to 48x48 feature map. However, this is followed by a 2x2 max pooling operation, which reduces the dimensionality of the feature maps by half, resulting in a size of 24x24.

This process is repeated for the second and third convolutional operation. The final feature map has a dimensionality of 3x3 pixels, and prior to being passed to the fully connected layer, the feature map is flattened into a one dimensional vector.

A fully connected layer assigns a separate weight to each output node for all input node. In the case of 2304 inputs $(3*3*256)$ and 128 outputs, this results in 294,912 weights. Additionally, the layer has a bias for each output node, increasing the number of parameters to $(2304 + 1) * 128 = 295,040$. The output value is randomly selected from 2^n, where n equals 7 in this case. The second and final fully connected layer takes 128 nodes as input, which is the output from the first fully connected layer. The final output of the network is either a sphaleron or a microscopic black hole, resulting in two possible outputs. Therefore, the number of parameters

[1] $\circledast$ is the symbol for the convolution operator

for this layer is calculated as $(128 + 1) * 2 = 258$.

Furthermore, considerations has been made to ensure that the model is capable of being trained on ImageNet. Two fully connected layers have been constructed, with the first layer consisting of 1000 output layers, and the second and final layer consisting of 2 output layers. By removing the second layer, the model would be able to train on the ImageNet dataset.

The ImageNet dataset contains 1000 classes. Training on this dataset requires a model that has 1000 nodes in the last fully connected layer. The dataset consists of 14 million images, therefore it requires a lot of processing time. Due to the time limitation discussed in Limitations 3.6, no training has been conducted on this dataset (Russakovsky et al., 2015).

The Figure 5.1 below illustrates the pipeline of the CNN model, showcasing the progression from the input layer to the output layer.

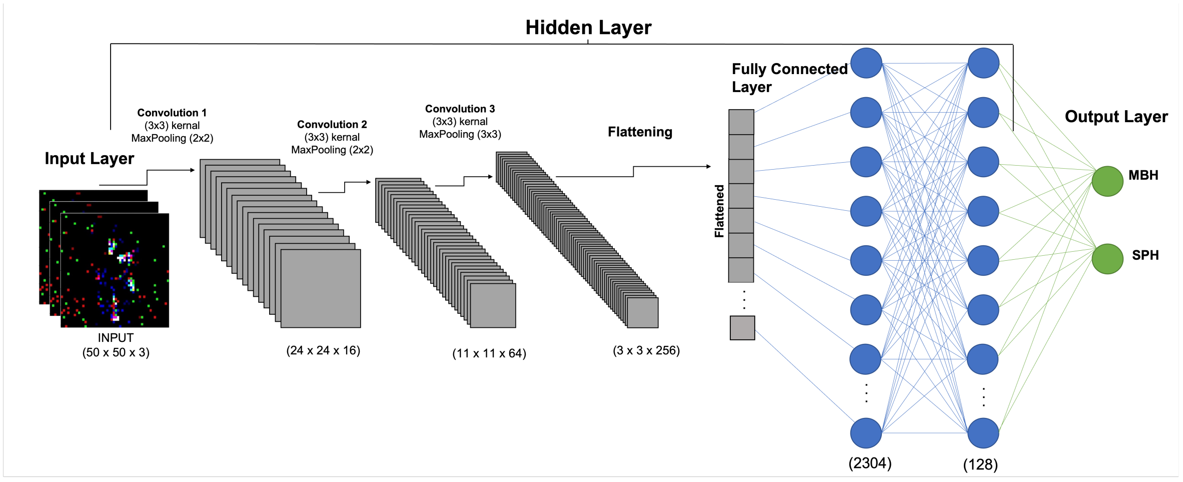


Figure 5.1: CNN model architecture - The CNN model consists of 3 convolution layers and 2 fully connected layers. The input dimension is 50x50x3, by the end of the last convolution layer the shape is 3x3x256. The output of the last fully connected layer is 2, microscopic black hole and sphaleron.

5.1.2 ResNet

In a ResNet architecture, the design of the residual block differs based on the network's depth. For smaller networks such as ResNet-34, each residual block consists of two convolutional layers. In contrast, for deeper networks like ResNet-50 and ResNet-101, each residual block contains three convolutional layers. Deeper ResNet networks use three-layer residual blocks to capture complex features, while shallower networks with two-layer residual blocks are suitable for less complex features or smaller datasets. This projet explores ResNet-34, ResNet-50, and ResNet-101 architecture. The Figure 5.2 below illustrates the structure of three different residual blocks.

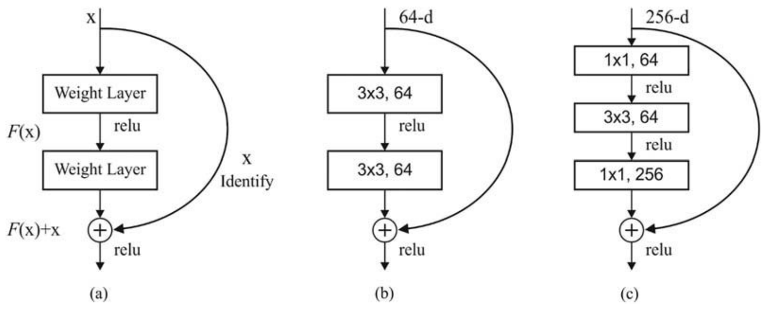


Figure 5.2: (a) Residual block, (b) two layer deep, (c) three layer deep (Venugopal et al., 2021)

The architecture is based on a pipeline that incorporates five distinct convolutional layers, depicted in Table 5.2. However, the main variation between these layers is the number of layers utilized (34, 50, 101), and the type of residual block.

layer name	34-layer	50-layer	101-layer
conv1	$7 \times 7, 64$, stride 2 3×3 max pool, stride 2		
conv2	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
conv4	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$
conv5	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	average pool, 2048-d fc		

Table 5.2: Detailed construction of the different ResNet models - The table contains dimensions and size of each residual block used to construct the ResNet models (Zhang et al., 2015).

ResNet34 Implementation The ResNet34 architecture consists of 16 residual blocks ($3 + 4 + 6 + 3 = 16$), each residual block contains 2 convolutional layers. Additionally, the first layer of the architecture also includes 2 convolutional layers, which are not part of a residual block. Consequently, the total number of convolutional layers in ResNet34 is $((16 * 2) + 2)$. Although it is considered relatively shallow compared to ResNet50 and ResNet101, it remains an effective architecture due to its fewer number of parameters, resulting in faster training and less prone to overfitting.

In this particular architecture, the residual block is implemented as an identity block. The indetity block skips over two layers in the main path. The layers that are skipped consists a conv2d, and a batch normalization as shown in the Figure 5.3. The first layer also consists of a ReLu function. The shortcut connection in the block skips the two layers and forwards the output of one layer as the input to the next layers. The resulting output is then passed through a ReLU function.

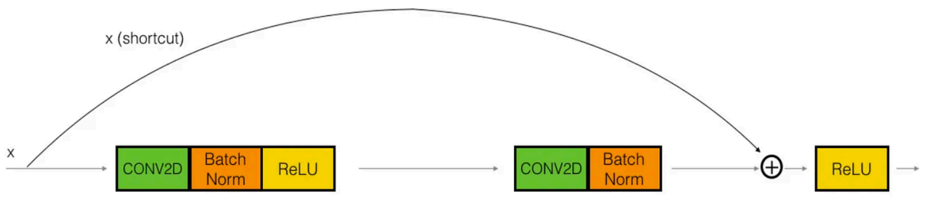


Figure 5.3: Architecture of the identity block (Jjuinni, 2023)

ResNet50 and ResNet101 Implementation ResNet50 and ResNet101 have a larger number of parameters, enabling them to handle more complex tasks compared to ResNet34. In these models, the residual block is implemented as a bottleneck block, further reducing the number of parameters while increasing the depth of the network.

5.2 Selection of Loss Functions

5.2.1 Custom Loss function

When developing a custom loss function, a benchmark will be significant to compare against. Cross-entropy will serve as the benchmark for evaluating the custom loss functions.

In the initial stage of selecting a loss function, the zero-one loss function was considered as a potential approach for the model. This function does not penalize the model for making correct predictions with low confidence, in contrast to cross-entropy which heavily penalized the model for making correct but low-confidence predictions.

The idea is to make the penalty of cross-entropy resemble that of the zero-one loss. To achieve this, the softmax activation function in the model can be modified. One approach is to apply a root function to the input, which would flatten the predicted estimates and make the model's confidence less important. By applying this modification, it is expected to decrease the penalty for accurate predictions made with low confidence and increase the penalty for inaccurate predictions made with high confidence.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\Sigma_j \exp(x_j)}$$

Figure 5.4: Original implementation of the softmax function

The modified softmax function introduces a new variable that determines the root to be applied to the input value.

$$\mathrm{Softmax}(x_i, r) = \frac{\exp(\mathrm{sgn}(x_i) * \sqrt[r]{|x_i|})}{\Sigma_j \exp(\mathrm{sgn}(x_i) * \sqrt[r]{|x_j|})}$$

Figure 5.5: Custom Loss Function - Modified implementation of the softmax function. The parameter X_i is the model output, and r is the root index. The function sgn() is used to negative or positive value of a number(outputs -1 or +1)

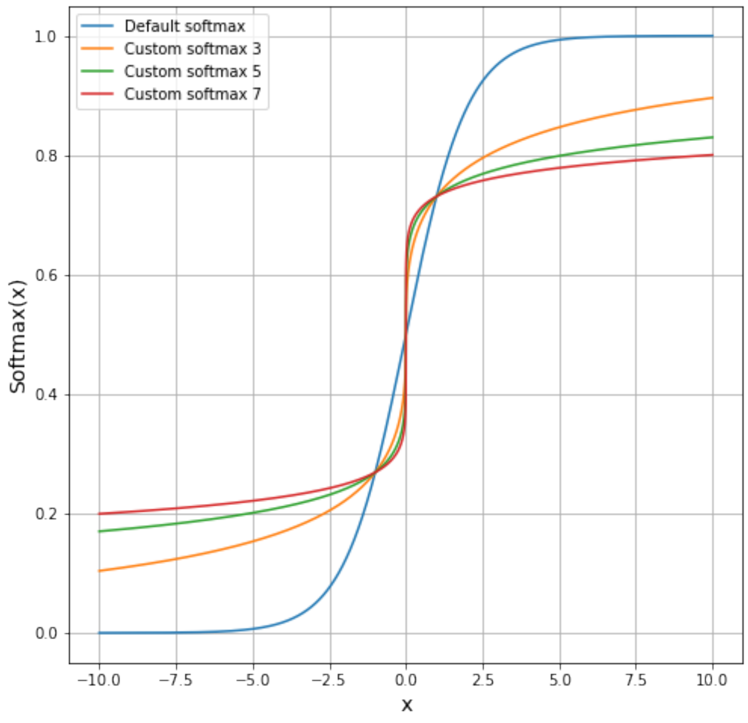


Figure 5.6: Comparison between softmax and modified version of softmax

In this classification task, the input of the softmax function is 2-dimensional, representing the model weight for microscopic black hole and sphalerons. Figure 5.6 depicts various modified softmax functions in comparison to the original softmax function. The X-axis represents the weight given to the microscopic black hole when the weight for sphalerons is 0, while the Y-axis represents the output for microscopic black hole. The legend [2] indicates which softmax function is being applied, with the number behind the custom softmax indicating the root index that has been used.

[2]Legend is the textbox that labels the data by color.

By comparing the different softmax functions, it becomes evident that the modification we made to the softmax function reduces the impact of model confidence as the root value increases. When a higher root index is applied, the softmax function demonstrates a trend of converging closer to $y = 0.5$. For positive x-values standard softmax converges to $y = 1$, whilst softmax-7 converges closer to $y = 0.8$. A similar pattern is seen for the negative x-values, were softmax converges to $y = 0$, whilst softmax-7 converges closer to $y = 0.2$. This modification is intended to bring the penalty of cross-entropy closer to that of the zero-one loss function.

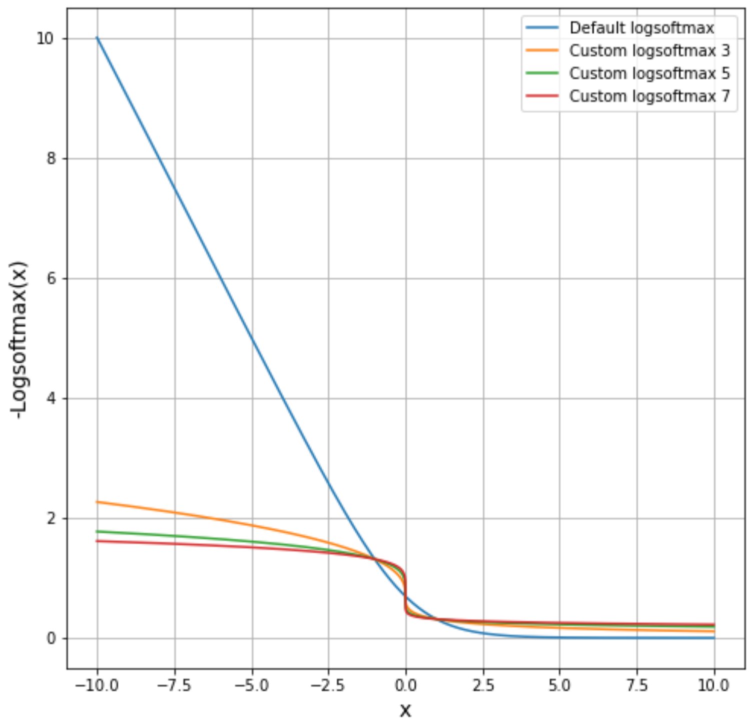


Figure 5.7: Comparison between -logsoftmax and modified version of -logsoftmax. Note that -logsoftmax and cross entropy is mathematical the same (de Brébisson and Vincent, 2016).

Figure 5.7 shows a comparison between cross-entropy and the custom loss function. The x-axis represents the model output for MBH when the value for SPH is 0. The y-axis represents the loss value if the correct label is MBH. For instance, when evaluating an output with a confidence score represented by $x = -5$, the custom loss-3 function would yield a loss value of approximately 2, whereas the standard cross-entropy loss function would result in a higher loss value of approximately 5. However, with an output confidence score of $x = -10$, the custom loss-3 function would yield a slightly above 2 loss value, while the standard cross-entropy loss function would yield a significantly higher loss value of 10. This behavior exhibits similarity to the zero-one loss, where both a highly confident incorrect model output and a moderately confident incorrect output yield similar loss values. In contrast, the standard cross-entropy loss results a significantly higher loss value to a highly confident incorrect model output.

```
1  def logSoftmax(x, root):
2      x_pow = x.sign() * x.abs().pow(1/root)
3      x_exp = torch.exp(x_pow)
4      x_exp_sum = torch.sum(x_exp, 1, keepdim=True)
5      return torch.log(x_exp/x_exp_sum)
6  
7  class CustomLoss(nn.Module):
8      def __init__(self, root):
9          super(CustomLoss, self).__init__()
10         self.root = root
11 
12     def forward(self, input, targets):
13         batch_size = input.size()[0]
14         input = logSoftmax(input, self.root)
15         input = input[range(batch_size), targets]
16         return -torch.sum(input) / batch_size
```

Listing 5.1: Implementation of the custom loss function

Listing 5.1 above shows the implementation of the custom loss function. The modification made to the original implementation in the pyTorch library involves the addition of line 2 and the introduction of a function parameter called ”root”.

5.3 ImageNet

In Section 2.2.3, it is discussed that utilizing pre-trained models from ImageNet can enhance performance. The suggested approach involves initially freezing all layers, except for the last fully connected layer before the output layer. The final fully connected layer should consist of 1000 input nodes and 2 output nodes. Subsequently, this last layer is trained using the projects dataset.

The following Listing 5.2 demonstrates the implementation of the ResNet34 model using the PyTorch library. The weights are imported from the torchvision.models module. The implementation for ResNet50 is identical.

```python
class Resnet34(nn.Module):
    def __init__(self):
        super(Resnet34, self).__init__()
        #Initialize weights from a resnet34 model pre-trained on ImageNet:
        self.layers = resnet34(weights=ResNet34_Weights.IMAGENET1K_V1)
        #Fully connected layer with 1000 input nodes and 2 output nodes:
        self.new_layer = nn.Sequential(nn.Linear(1000, 2))

    def forward(self, x):
        x = self.layers(x)
        x = self.new_layer(x)
        return x
```

Listing 5.2: ResNet34 pre-trained on ImageNet

The Listing 5.3 below demonstrates the initialization of the model and the process of freezing its layers

```python
#Initialized the model and moves to gpu
model = Resnet34().to(device)
#Freez the layers, look at implementation
model.layers.requires_grad = False
```

Listing 5.3: Initialization of the model

6 Results

This chapter presents the results obtained from the implemented models. Results from each model architecture will be analyzed, aiming to comprehend their relevance to the problem statement and the sub-goals of the project. Additionally, the results will be compared against a benchmark to evaluate their performance. Furthermore, observations will be discussed to understand underlying patterns that has been discovered that influence the results. By evaluating these results, this chapter lays the groundwork for the subsequent discussion, which will address the problem statement and the sub-goals of the project.

6.1 Analysing the CNN Model

The constructed CNN model has a relatively simple structure, comprised of three convolutional layers and two fully connected layers. The incorporation of custom loss functions introduces an additional layer of complexity to this model. The additional complexity is expected to achieve a better performances than the standard cross-entropy loss function.

The custom loss function is essentially a modified softmax function that applies a root on the output value. The index of the root is an additional parameter that is adjusted. As previously mentioned in Section 4.4.4, the hyperparameter tuning process in this study is limited to the loss function. The index values are arbitrarily values, ranging from 3 to 9, as seen in Figure 6.1 below.

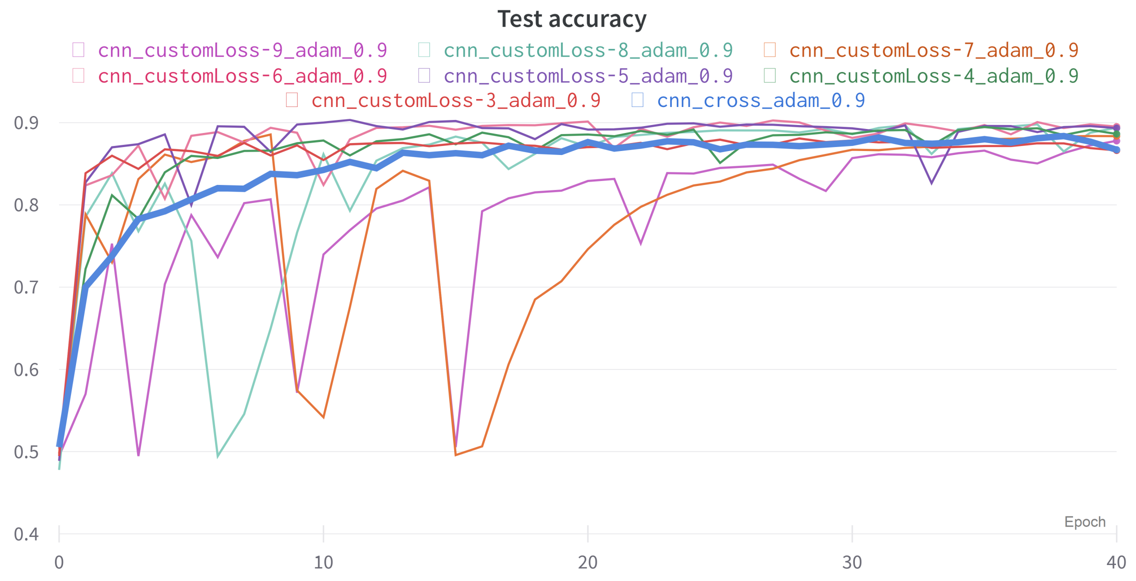


Figure 6.1: CNN Loss function comparison - Graph containing accuracy comparison of 8 loss functions. This graph is one of three other used for CNN comparison. The blue highlighted line is cross-entropy loss which is the benchmark.

The line graph displays the training performance of the implemented loss functions for each epoch. This graph effectively captures noticeable changes in the trend over the time of 40 epochs. Towards the upper end of the range, where the index increases, it becomes evident that the custom loss functions display instability during the early stages of training.

The line graph illustrates the performance of a single run. However, to accurately evaluate the model's performance, it is necessary to conduct at least three runs for each model. By calculating the average accuracy across these three runs, a more reliable estimate of the model's accuracy can be obtained. Subsequently, the results are evaluated in comparison to the standard cross-entropy benchmark. The accuracy for each run is listed in the Table 6.1, and the average accuracy for each model is illustrated in a bar plot in Figure 6.2, with the standard deviation is represented as error bars.

Loss Function	Run	epoch 10	epoch 20	epoch 40
Cross-entropy	Run 1	84.2%	87.6%	86.7%
	Run 2	82.8%	86.9%	88.7%
	Run 3	85.2%	85.0%	87.7%
Custom-Loss-3	Run 1	85.4%	87.0%	86.6%
	Run 2	86.5%	87.0%	87.4%
	Run 3	86.1%	87.4%	88.1%
Custom-Loss-4	Run 1	87.8%	88.5%	88.6%
	Run 2	85.3%	86.1%	88.2%
	Run 3	88.1%	88.1%	86.1%
Custom-Loss-5	Run 1	90.0%	89.1%	89.3%
	Run 2	88.3%	89.2%	90.5%
	Run 3	89.9%	89.2%	89.1%
Custom-Loss-6	Run 1	82.4%	90.1%	89.5%
	Run 2	50.6%	87.0%	89.7%
	Run 3	89.0%	89.0%	90.6%
Custom-Loss-7	Run 1	54.1%	74.6%	88.3%
	Run 2	88.8%	88.7%	81.8%
	Run 3	88.6%	88.9%	89.9%
Custom-Loss-8	Run 1	86.1%	87.0%	89.3%
	Run 2	85.9%	50.5%	90.0%
	Run 3	83.5%	88.8%	88.2%
Custom-Loss-9	Run 1	73.9%	82.9%	87.7%
	Run 2	75.2%	83.6%	88.2%
	Run 3	84.5%	85.8%	87.5%

Table 6.1: CNN Loss function comparison table - Table containing accuracy comparison of 8 loss functions over 3 runs.

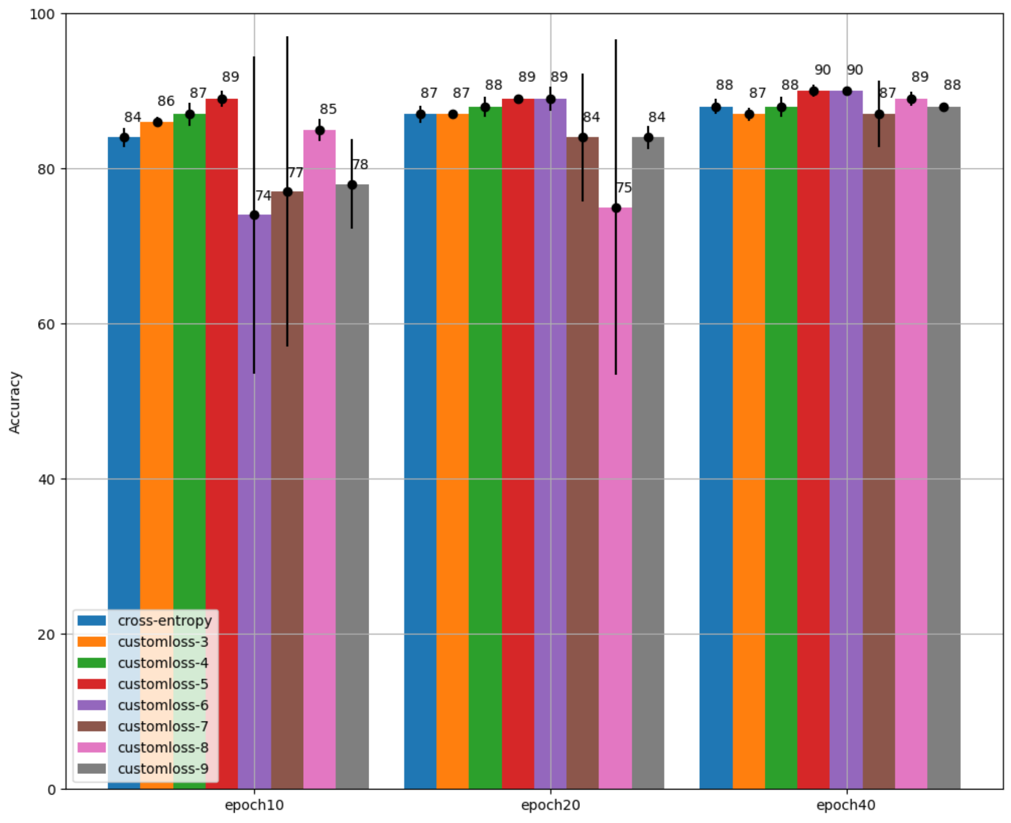


Figure 6.2: CNN Loss function bar plot - Bar plot containing average accuracy of the loss functions with standard deviation as error bar.

The bar graph provides an comprehensible visualization of the average performance of accuracy for each model at epochs 10, 20, and 40, along with the display of standard deviation. At epoch 10, it can be observed that there is slightly more efficient training when the custom loss function's root index increases from 3 to 5, compared to the standard cross-entropy benchmark. However, when the index increases from 6 to 9, there is a sudden drop in performance. A reason for this instability may be correlated to Figure 5.6. A higher root index will lead to a steeper derivative, meaning that optimizer may take too large steps when adjusting the model parameters. Eventually, at epoch 40, all models converge to a similar level of accuracy, but Custom-loss-5 exhibits the most stable training performance and achieves high accuracy.

Custom-loss-4 and custom-loss-5 consistently demonstrate a consistent performance throughout all epochs. This suggests that these custom loss functions have the potential to decrease the required number of epochs to achieve a point where further performance improvements are no longer significant.

6.2 Analysing the ResNet Model

When choosing the best ResNet model, the focus shifts towards the architecture itself rather than the performance of the custom loss function, which differs from the approach taken with CNN models.

The attached document Appendix B.13 and Appendix B.14 reveals that ResNet101 exhibit instability, whilst all ResNet models show a tendency to overfit. Although the models overfit, there is little to no difference in accuracy. As the amount of layers increase in the time it takes train also increases, therefore only ResNet34 is utilized for further evaluation of the impact of the custom loss function.

As the number of layers increases, the models become more susceptible to overfitting. However, in Figure 6.3, it is evident that ResNet34 displays relatively less instability and overfitting in comparison. As a result, only ResNet34 is utilized for further evaluation of the impact of the custom loss function.

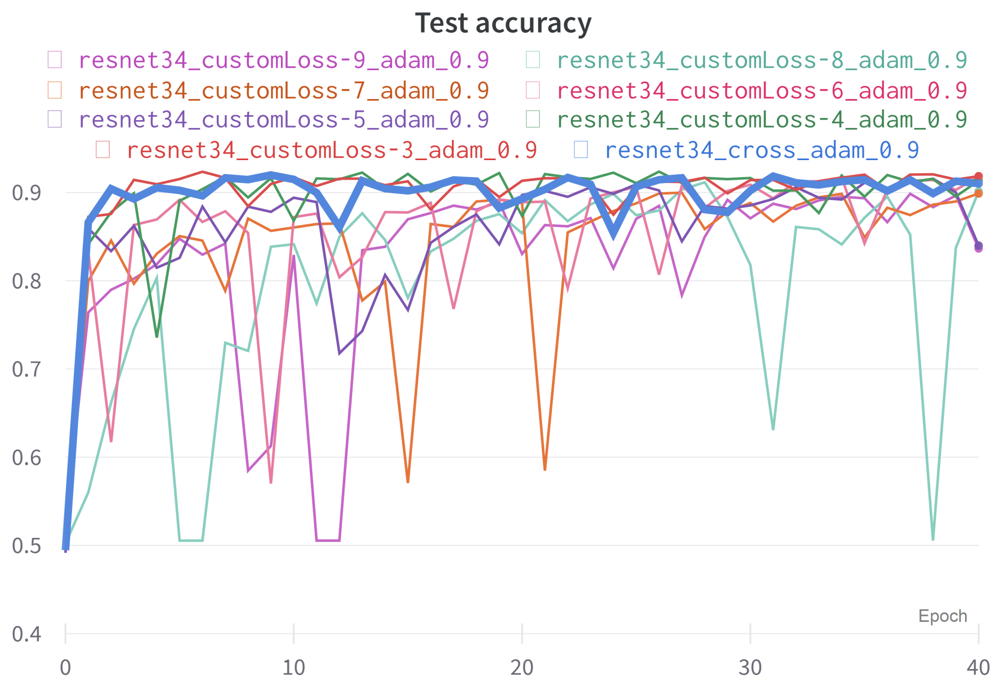


Figure 6.3: ResNet Loss function comparison - Graph containing accuracy comparison of 8 loss functions. This graph is one of three other used for ResNet34 comparison. The blue highlighted line is cross-entropy loss.

A graph illustrating the training performance of the implemented loss functions for ResNet34 is also provided. Similar to the CNN, root indices 3, 4, 5, 6, 7, 8, and 9 have been tested out. However, compared to the CNN run, the graph shows increased instability for several of functions. This instability appears to persist across the epochs. Notably, ResNet-Custom-Loss-8 appears to reach an accuracy of 50% around epoch 38. This observation suggests that the ResNet-Custom-Loss-8 model likely fails to reach a flattening, indicating an unstable training performance.

The loss functions with higher root index values, such as 6, 7, 8, and 9, demonstrate the most instability. This observation aligns with the findings discussed during the analysis of the CNN model in Figure 6.1. It becomes apparent that increasing the root index leads to unstable behavior for both architectures.

ResNet tends to exhibit overfitting as the number of layers increases. Therefore, it is reasonable to compare the test accuracy with the train accuracy to analyze the model's performance. The Figure 6.4 below depicts the accuracy on the training dataset. Standard cross-entropy, custom-loss-3, and custom-loss-4 achieves a accuracy of 99% by epoch 40, indicating that they overfit. Comparing these findings to the accuracy results provided in Table 6.2, it is observable that for epoch 40, the accuracy for these functions hovers around 91%. This substantial difference between the training and test accuracy suggests that the models have learned the patterns present in the training set.

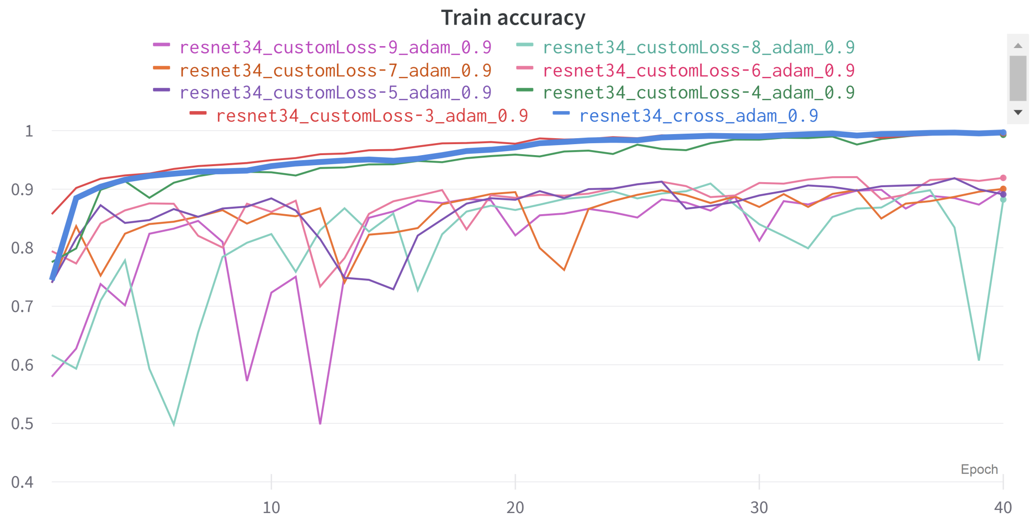


Figure 6.4: ResNet Custom-loss functions comparison - Figure shows the accuracy on the train dataset for Figure 6.4. The blue highlighted line is cross-entropy loss.

Loss Function	**Run**	**epoch 10**	**epoch 20**	**epoch 40**
Cross-entropy	Run 1	91.5%	89.3%	91.0%
	Run 2	92.5%	91.5%	88.8%
	Run 3	92.0%	90.5%	91.1%
Custom-Loss-3	Run 1	91.7%	91.3%	91.9%
	Run 2	91.4%	89.4%	91.0%
	Run 3	90.0%	91.1%	91.3%
Custom-Loss-4	Run 1	86.8%	87.4%	91.4%
	Run 2	87.4%	90.6%	90.8%
	Run 3	91.6%	92.3%	88.4%
Custom-Loss-5	Run 1	89.4%	89.7%	84.0%
	Run 2	82.5%	89.5%	92.0%
	Run 3	89.3%	89.4%	89.0%
Custom-Loss-6	Run 1	87.2%	88.9%	91.8%
	Run 2	84.5%	89.9%	88.8%
	Run 3	85.4%	86.0%	79.7%
Custom-Loss-7	Run 1	86.0%	88.9%	89.9%
	Run 2	84.1%	65.0%	89.6%
	Run 3	83.1%	87.6%	90.2%
Custom-Loss-8	Run 1	84.2%	85.4%	90.1%
	Run 2	87.0%	50.6%	89.1%
	Run 3	86.4%	87.5%	88.1%
Custom-Loss-9	Run 1	82.9%	83.0%	83.7%
	Run 2	51.8%	83.9%	84.0%
	Run 3	81.7%	88.0%	86.7%

Table 6.2: ResNet Loss function comparison table - Table containing accuracy comparison of 8 loss functions over 3 runs.

The bar plot in Figure 6.5, representing the ResNet34 models, exhibits notable differences compared to the CNN model. An interesting observation is that as the root index increases, the performance decreases, whereas the CNN models demonstrated improved performance. Additionally, the standard cross-entropy benchmark shows a gradual decrease in performance by epoch 20 and 40.

The ResNet-Custom-Loss-7 and ResNet-Custom-Loss-8 models, which demonstrate signs of instability, exhibit a noticeable drop in performance around epoch 20. However, it is interesting to observe that by epoch 40, ResNet-Custom-Loss-7 demonstrates an increase in performance by 6%, while ResNet-Custom-Loss-8 demonstrates a 3% increase. This contrasts with the behavior of the cross-entropy benchmark, which experiences a decrease of 2% by epoch 40. This observation could potentially be explained by the overfitting observed in the training graph.

ResNet-Custom-Loss-9, similar to the other unstable models, also displays instability. However, it exhibits poor performance by epoch 10, followed by an improvement by epoch 20. Unlike the CNN models, there are no notable discernible trends in the accuracy patterns. One consistent observation remains that as the index value increases, both the ResNet and CNN models tend to become more unstable.

Seemingly, the custom loss function does not appear to benefit the performance of the model. However, it is noteworthy that the highest average accuracy achieved is 92% for ResNet34-Custom-Loss-3, while the highest average accuracy for the CNN model is 90% for CNN-Custom-Loss-5. This indicates that ResNet34-Custom-Loss-3 performs well, but it also suggests that there is potential for further improvement.

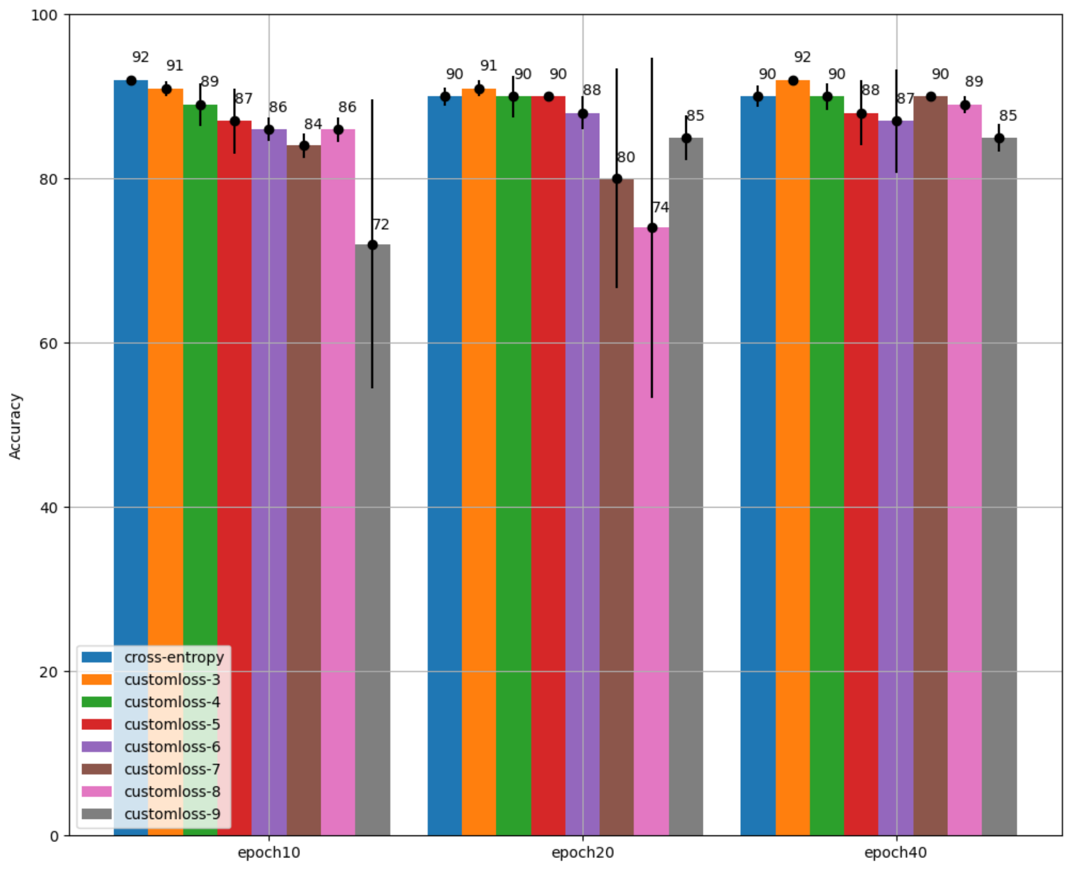


Figure 6.5: ResNet Loss function bar plot - Bar plot containing average accuracy of the loss functions with standard deviation as error bar.

6.3 Models pre-trained on ImageNet

In order to implement transfer learning, the weights from models pre-trained on ImageNet have been extracted and used to replace the weights of the best performing model. This approach aims to potentially decrease the number of epochs required to reach a peek point of accuracy.

The ResNet34-Custom-Loss-3 model will be compared to a pre-trained ResNet34-Custom-Loss-3 model. Below in Figure 6.6, the graph of the custom-loss model can be observed. The model have been run three times to analyze the average performance. The aim is to observe a peek point of accuracy in an early epoch, compared to the not pre-trained model.

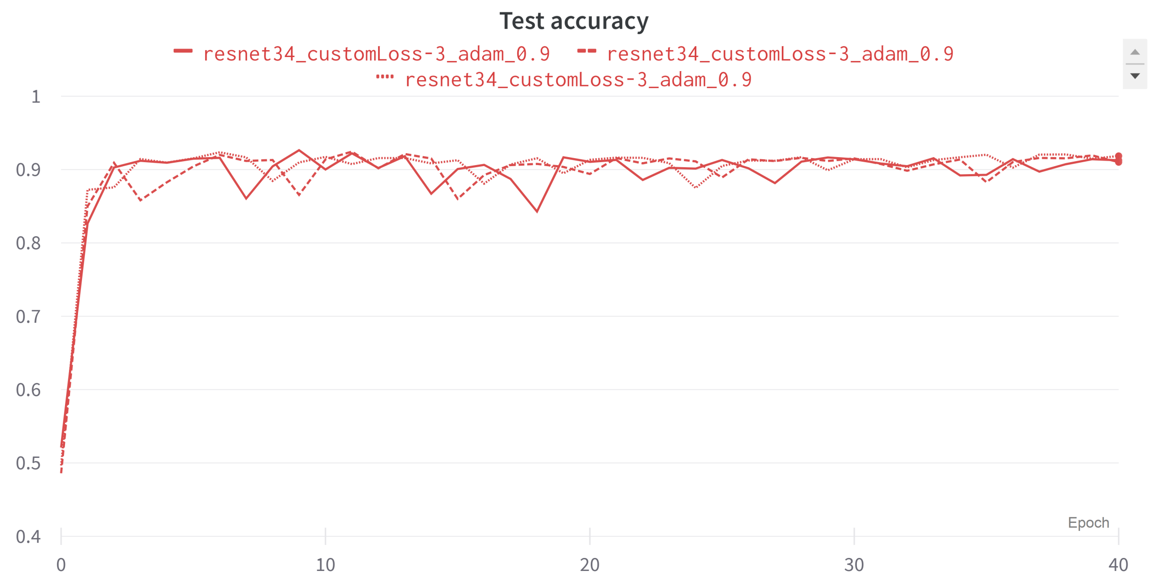


Figure 6.6: ResNet34 custom-loss-3 - Three runs each trained for 40 epochs.

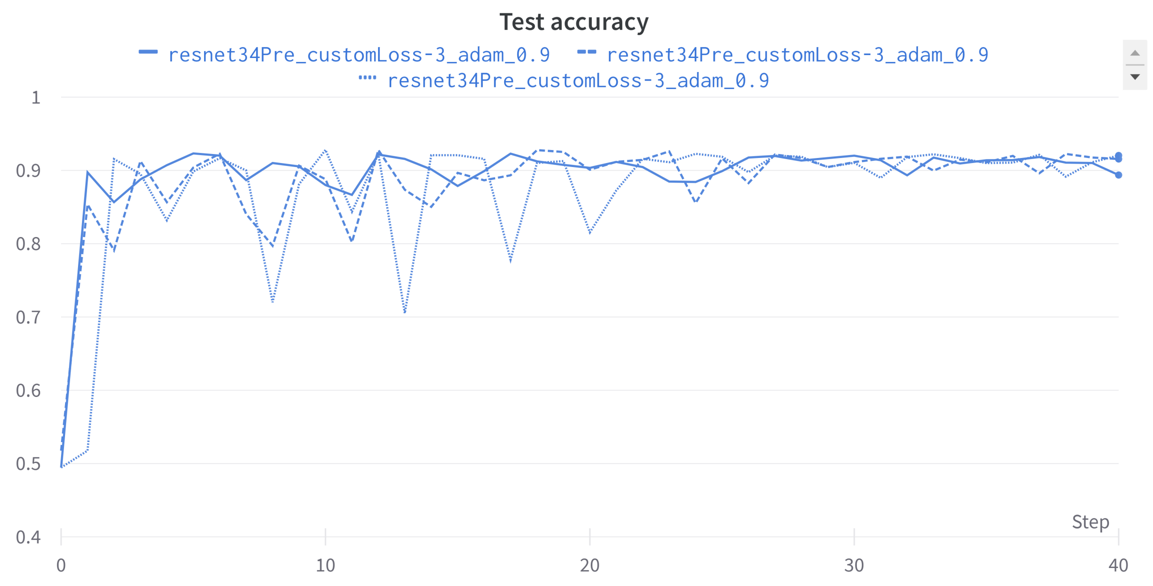


Figure 6.7: ResNet34Pre custom-loss-3 - Three runs each trained for 40 epochs.

Comparing the graphs from Figure 6.6 and Figure 6.7, it becomes evident that there is no distinct difference between the models. Both graphs exhibit a flattening trend around epoch 30, with the training remaining relatively stable. This observation suggests that the model pre-trained on ImageNet does not necessarily reach its peak accuracy in earlier epochs. Instead, it demonstrates a consistent level of stability with good accuracy throughout the training process.

A comprehensive comparison of the average accuracy for each model is presented in the Figure 6.3 below. The figure displays the average accuracy for three runs at epoch 10, 20, and 40. This numerical comparison allows for clear observation of the point in the training process where convergence begins to occur.

Model	**Average Accuracy on Three Runs (Epoch 10)**	**Average Accuracy on Three Runs (Epoch 20)**	**Average Accuracy on Three Runs (Epoch 40)**
ResNet34-Custom-Loss-3 (pre-trained)	90%	87%	91%
ResNet34-Custom-Loss-3 (Not pre-trained)	91%	91%	92%

Table 6.3: Comparison between 3 different ResNet34 models - The first model used cross-entropy and is pre-trained with ImageNet. The second model uses custom-loss-3 and is pre-trained with ImageNet. The third model uses custom-loss-3 and is not pre-trained.

As mentioned earlier, the table reveals that the accuracy remains consistent across all three epochs. Although this result does not indicate any noticeable differences, it indicates that for ResNet34-Custom-Loss-3, running 40 epochs for training may not be necessary.

6.4 Discussion

The research conducted in this study delve into the significance of various network architectures, examined the impact of diverse loss functions, and compare the models with those pre-trained on ImageNet Section 6.3. The results obtained provides insight into the performance of the models and their ability to generalize on unseen data. This section will examine how the obtained results address the problem statement and the corresponding subgoals.

This project has aimed to address the problem statement *"Explore and evaluate the performance of various neural network architectures and loss functions for accurately classifying simulated image data portraying particle collisions in the ATLAS detector"*. By analyzing the results in light of the research subgoals, insight of implications can be gained. The following sections will discuss the findings of this project for each subgoal:

Investigate the importance of different network architectures

Initially it was considered to use precision and recall as metrics in combination with accuracy. These metrics are particularly useful when dealing with class imbalance in a dataset, where one class has significantly more instances than the other. However, in the scenario described, where both classes have the same frequency in the dataset, it was concluded that precision and recall were not necessary.

The CNN model and the ResNet models express significantly different performance. The CNN model is more stable, with minimal overfitting and fairly good accuracy, considering its simplicity. However, the architectural construction demands more manual work for the mathematical operations in each layer. The CNN model's performance improved with the custom loss function, potentially due to its simpler architecture. The added complexity of the custom loss function may have contributed to the improved performance.

In contrast, the ResNet models tend to be unstable for each run. It can learn something incorrectly, leading to poor results, or produce good accuracy but still overfit. Possible solutions include using an early stopping algorithm or a scheduler to prevent poor performance. Under the development process, a scheduler was employed to reduce the learning rate. This slightly improved the model's instability but did not address the overfitting issue. Additional tuning is required, which demands more developing time.

Preprocessing of the data was not performed prior to training the models. However, employing methods like data augmentation could potentially reduce the overfitting problem observed for the ResNet models.

Considering that we have compared the models on the test dataset under the training process, it is possible that the models are overfitted to perform well on the test dataset. Ideally, a portion of the dataset should have been held aside for final evaluation. However, since we are not developing a model for practical use, this should not pose a significant issue.

Investigate the effect of using different loss functions

When selecting a benchmark loss function for the evaluation of a custom loss function, mainly cross-entropy were taken into consideration. The reason for this choice was that the custom-loss is a modification of cross-entropy. As a result, it was concluded that cross-entropy was a suitable loss function to determine if the modification made any improvement to the model.

The custom loss function is a modified version of cross-entropy that apply different roots index values on the input. Applying different exponents were also tested. However, the model failed to train with exponents, suggesting the need for further experimentation as the project group is uncertain why this is.

The ResNet models demonstrated only slight improvement when the custom loss function was incorporated. Given their inherently complex architecture, it's possible that the custom loss function isn't an ideal fit for these models.

It would be interesting to add more layers to the CNN model to increase its complexity and observe the impact of the custom loss function as the number of layers grows. For the ResNet model, creating a different custom loss function that can better handle its complex architecture might be more effective.

Investigate whether networks pre-trained on ImageNet data are useful for 2D histograms

When comparing pre-trained models on ImageNet, only ResNet34 were considered since it had the best resulting model. Although pre-training the CNN model was initially considered, it was ultimately considered to be unfeasible. The reason for this is the fact that the CNN model is custom-built and has a relatively simple structure. There are no corresponding model that are pre-trained on ImageNet. Even a slightly altered version of a CNN model could not be employed due to the mismatch in the number of weights. Hence, pre-training the CNN model was not a viable option.

On the contrary, ResNet models have a fixed depth of layers. The expectation was that extracting the weights from an already pre-trained ResNet model would lead to some improvement. However, the results revealed that the model experienced stable training that flattened out around epoch 30, similar to the original non-pre-trained model. Unfortunately, the results did not indicate any improvement in an early peak of accuracy. The only observation made was that the ResNet34-Custom-Loss-3 model did not require 40 epochs for training, thereby reducing the training time for this particular model.

7 Conclusion

This concluding chapter will reflect back on the findings and discussion of the project. The project aimed to address the problem statement *"Explore and evaluate the performance of various neural network architectures and loss functions for accurately classifying simulated image data portraying particle collisions in the ATLAS detector"*. Throughout the project, several significant findings have been uncovered that contribute to this statement.

This project introduces a deep learning approach for the classification of microscopic black holes and sphalerons. Through the utilization of deep learning, the project overcomes the limitations associated with traditional classification methods, which often demands expertise in particle physics and advanced mathematics, in addition to being time-consuming.

The data provided for this project consists of simulated proton-proton collision data presented as 2D histograms. The paper 'End-to-End Physics Event Classification with CMS Open Data' (Andrews et al., 2020) introduced the utilization of 2D histograms in the fields of physics and machine learning. This project draws inspiration from that paper and aims to accurately classify the simulated 2D histogram images.

The project examined the application of CNN (Convolutional Neural Network) architecture and ResNet (Residual Network) architecture in analyzing 2D histogram data. Furthermore, a customized loss function was developed to improve the performance of these models. The outcomes obtained from these two models exhibit notable differences.

From the CNN models, the results demonstrated a significant improvement in accuracy when using the custom loss function, whereas the ResNet models did not exhibit the same results. This might be explained by the simplicity of the CNN model, which has only three layers, in contrast to the more complex, multi-layered ResNet models. The custom loss function adds an additional layer of complexity to the CNN model, leading to better performance compared to the standard cross-entropy loss function. However, the ResNet34-CustomLoss-3 achieves a 2% higher accuracy compared to CNN-CustomLoss-5. It is worth noting that the custom loss function may not be as well-suited for the ResNet architecture, resulting in less significant accuracy improvements compared to the benchmark. Nevertheless, the ResNet model has the potential to further enhance its performance by employing data processing techniques such as data augmentation and developing a more suitable custom loss function.

Additionally, an alternative approach was explored using transfer learning. The ResNet34-CustomLoss-3 model was trained with weights extracted from pretrained ImageNet models. Unfortunately, the results did not demonstrate any noticeable improvement in terms of early peak accuracy. However, it was observed that the ResNet34-Custom-Loss-3 model required fewer than 40 epochs for training. This indicates that the ResNet34-Custom-Loss-3 model requires less training time.

8 Suggestion for further research

This project has explored various aspects of utilizing deep learning for the classification of microscopic black holes and sphalerons. However, there are areas that require further research. Taking into account of the findings and limitations of this project, the following suggestions for further research are proposed.

I **Creating other loss functions -** The CNN model demonstrates improved performance through the utilization of a custom loss function. Conversely, the ResNet model did not exhibit a substantial improvement. Therefore, it would be advantageous to create a custom loss function that is more suitable for the ResNet model.

II **Increase the complexity of the CNN model -** By adding more layers to the CNN model, it would improve the complexity. This would be interesting to further explore, and to see if it yields different results and to observe the impact of using a custom loss function on the model. By delving into this approach, a better understanding of the reasons behind the positive results obtained from this loss function can be gained.

III **Implementing regularization -** To reduce overfitting in the ResNet model, it is worth considering the exploration of regularization methods for image classification. One potential idea is to incorporate L2 regularization into the custom loss function.

IV **Testing out the custom loss function on newer models -** To gain a more comprehensive understanding of the loss function's performance, it would be interesting to evaluate it on newer and more complex networks such as ConvNeXT(Liu et al., 2022).

V **Exploring the use of data augmentation -** The overfitting issue of the ResNet model could potentially be reduced by incorporating data processing techniques like data augmentation. Implementing data augmentation methods is addressed in a separate collaborative thesis. Combining the findings from these two projects would offer interesting possibilities for further research.

Bibliography

[1] Amram, N. and Etzion, E. (2023), 'Hough transform track reconstruction in the cathode strip chambers in atlas'. Retrieved from: http://cds.cern.ch/record/1118033?ln=en(Accessed: May 2, 2023).

[2] Andrews, M., Paulini, M., Gleyzer, S. and Poczos, B. (2020), 'End-to-end physics event classification with cms open data.'. Retrieved from: https://arxiv.org/abs/1807.11916 (Accessed: January 11, 2023).

[3] ATLAS Collaboration (2020), 'Atlas open data: Monte carlo datasets'. Retrieved from: http://opendata.atlas.cern/release/2020/documentation/datasets/mc.html(Accessed: April 26, 2023).

[4] ATLAS-Collaboration (2022), 'Atlas experiment records "first physics" at new high-energy frontier'. Retrieved from: https://atlas.cern/Updates/Press-Statement/Run3-first-collisions (Accessed: March 28, 2023).

[5] ATLAS-Experiment (n.d), 'The atlas detector'. Retrieved from: https://atlas.cern/Discover/Detector (Accessed: March 28, 2023).

[6] Bahr, D. (2012), Miniature Ambulatory Skin Conductance Monitor and Methods for Investigating Hot Flash Phenomena, PhD thesis. Retrieved from: 10.13140/RG.2.1.5166.8885(Accessed: May 7, 2023).

[7] Balciunas, P. (n.d.), 'Keras resnet tutorial'. Retrieved from: https://pylessons.com/Keras-ResNet-tutorial(Accessed: May 7, 2023).

[8] Biswal, A. (2023), 'The complete guide on overfitting and underfitting in machine learning'. Retrieved from: https://www.simplilearn.com/tutorials/machine-learning-tutorial/overfitting-and-underfitting(Accessed: May 6, 2023).

[9] Camp, D. (2021), 'Convolutional neural networks'. Retrieved from: https://databasecamp.de/en/ml/convolutional-neural-networks(Accessed: May 6, 2023).

[10] CERN (2022), 'Run 3 of the large hadron collider'. Retrieved from: https://home.cern/press/2022/run-3 (Accessed: March 28, 2023).

[11] CERN (2023), 'Atlas observes and measures h->ww'. Retrieved from: https://press.cern/news/news/cern/atlas-observes-and-measures-h-ww(Accessed: March 3, 2023).

[12] Cern. (n.d*a*), 'how-detector-works'. Retrieved from: https://home.cern/science/experiments/how-detector-works (Accessed: April 12, 2023).

[13] Cern. (n.d*b*), 'Recreating big bang matter on earth'. Retrieved from: https://home.cern/news/series/lhc-physics-ten/recreating-big-bang-matter-earth (Accessed: March 9, 2023).

[14] Chaudhuri, A. and Khlopov, M. (2021), 'Balancing asymmetric dark matter with baryon asymmetry by sphaleron transitions'. Retrieved from: https://arxiv.org/pdf/2107.12611 (Accessed: March 16, 2023).

[15] Cheng, H., Wang, Z., Ma, L., Wei, Z., Alsaadi, F. E. and Liu, X. (2023), 'Differentiable channel pruning guided via attention mechanism: a novel neural network pruning approach'. Retrieved from: https://doi.org/10.1007/s40747-023-01022-61(Accessed: May 7, 2023).

[16] Cunningham, P., Cord, M. and Delany, S. J. (2008), 'Machine learning techniques for multimedia: Case studies on organization and retrieval - supervised learnin'. Retrieved from: https://doi.org/10.1007/978-3-540-75171-7_2(Accessed: April 26, 2023).

[17] Dai, Y., Xue, C. and Zhou, L. (2022), 'Visual saliency guided perceptual adaptive quantization based on hevc intra-coding for planetary images'. Retrieved from: https://doi.org/10.1371/journal.pone.0263729(Accessed: April 26, 2023).

[18] de Brébisson, A. and Vincent, P. (2016), 'An exploration of softmax alternatives belonging to the spherical loss family'. Retrieved from: https://arxiv.org/abs/1511.05042(Accessed: May 19, 2023).

[19] Dumoulin, V. and Visin, F. (2018), 'A guide to convolution arithmetic for deep learning'. Retrieved from: https://arxiv.org/pdf/1603.07285.pdf (Accessed: April 26, 2023).

[20] Education, A. E. (2023), 'What is the scientific method?'. Retrieved from: https://www.australianenvironmentaleducation.com.au/education-resources/what-is-the-scientific-method/(Accessed: March 03, 2023).

[21] Gannouji, R. (2022), 'Introduction to electroweak baryogenesis'. Retrieved from: https://www.mdpi.com/2075-4434/10/6/116(Accessed: May 4, 2023).

[22] Howard, A., Zhmoginov, A., Chen, L.-C., Sandler, M. and Zhu, M. (2018), 'Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation'. Retrieved from: https://arxiv.org/abs/1801.04381v4(Accessed: April 26, 2023).

[23] Howard, J. and Gugger, S. (2020), *Deep Learning for Coders with fastai and PyTorch: AI Applications Without a PhD*.

[24] Howard, J. and Thomas, D. R. (n.d), 'Practical deep learning for coders - practical deep learning'. Retrieved from: https://course.fast.ai/ (Accessed: April 16, 2023).

[25] Huber, A., Hu, Y., Anumula, J. and Liu, S.-C. (2019), 'Overcoming the vanishing gradient problem in plain recurrent networks'. Retrieved from: https://arxiv.org/pdf/1801.06105.pdf(Accessed: April 26, 2023).

[26] HVL-ATLAS-Group (n.d), 'Learning dark matter'. Retrieved from: https://learningdarkmatter.com/ (Accessed: March 28, 2023).

[27] ImageNet (n.d), 'Imagenet large scale visual recognition challenge 2012 (ilsvrc2012)'. Retrieved from: https://www.image-net.org/challenges/LSVRC/2012/ (Accessed: April 16, 2023).

[28] IndoML (2018), 'Student notes: Convolutional neural networks (cnn) introduction'. Retrieved from: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/(Accessed: May 7, 2023).

[29] InterviewBit (2021), 'Convolutional neural network (cnn) architecture'. Retrieved from: https://www.interviewbit.com/blog/cnn-architecture/(Accessed: May 6, 2023).

[30] Janiesch, C., Zschech, P. and Heinrich, K. (2021), 'Machine learning and deep learning'. Retrieved from: https://link.springer.com/article/10.1007/s12525-021-00475-2(Accessed: May 4, 2023).

[31] Jjuinni (2023), 'Cnn architecture from scratch: Resnet50 with keras'. Retrieved from: https://jjuinni.medium.com/cnn-architecture-from-scratch-resnet50-with-keras-4414539521d1(Accessed: March 03, 2023).

[32] Jones, D. G. (2023), 'Large hadron collider'. Retrieved from: https://www.britannica.com/technology/Large-Hadron-Collider (Accessed: March 28, 2023).

[33] Kandel, I. and Castelli, M. (2020), 'Transfer learning with convolutional neural networks for diabetic retinopathy image classification. a review'. Retrieved from: https://www.mdpi.com/2076-3417/10/6/2021(Accessed: May 4, 2023).

[34] Karpathy, A., Johnson, J. and Li, F.-F. (2016), 'Cs231n convolutional neural networks for visual recognition: Convolutional neural networks'. Retrieved from: https://cs231n.github.io/convolutional-networks/ (Accessed: April 26, 2023).

[35] Kaya, E. and Baştemur Kaya, C. (2021), 'A novel neural network training algorithm for the identification of nonlinear static systems: Artificial bee colony algorithm based on effective scout bee stage'. Retrieved from: https://www.mdpi.com/2073-8994/13/3/419(Accessed: March 10, 2023).

[36] Ladders (2022), 'What are the 7 scientific method steps?'. Retrieved from: https://www.theladders.com/career-advice/the-7-scientific-method-steps (Accessed: April 16, 2023).

[37] Li1, A. (2017), 'Introduction to the atlas detector at the lhc'. Retrieved from: https://doi.org/10.2991/icmmita-16.2016.53 (Accessed: March 28, 2023).

[38] Liashchynskyi, P. and Liashchynskyi, P. (2019), 'Grid search, random search, genetic algorithm: A big comparison for nas'. Retrieved from: https://arxiv.org/abs/1912.06059(Accessed: April 26, 2023).

[39] Liu, T., Fang, S., Zhao, Y., Wang, P. and Zhang, J. (2015), 'Implementation of training convolutional neural networks'. Retrieved from: https://arxiv.org/abs/1506.01195 (Accessed: April 26, 2023).

[40] Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T. and Xie, S. (2022), 'A convnet for the 2020s'.

[41] MathWorks (2021*a*), 'Deep learning'. Retrieved from: https://se.mathworks.com/discovery/deep-learning.html(Accessed: May 5, 2023).

[42] MathWorks (2021*b*), 'Overfitting'. Retrieved from: https://se.mathworks.com/discovery/overfitting.html(Accessed: May 6, 2023).

[43] May, A. (2022), 'What is the atlas experiment?'. Retrieved from: https://www.livescience.com/cern-atlas-experiment (Accessed: March 28, 2023).

[44] OpenData-ATLAS (n.d), 'Glossary: Jet'. Retrieved from: http://opendata.atlas.cern/books/current/get-started/_book/GLOSSARY.html (Accessed: April 26, 2023).

[45] O'Shea, K. and Nash, R. (2015), 'An introduction to convolutional neural networks'. Retrieved from: http://arxiv.org/abs/1511.08458 (Accessed: April 26, 2023).

[46] Pan, X., Shi, J., Luo, P., Wang, X. and Tang, X. (2018), 'Spatial as deep: Spatial cnn for traffic scene understanding'. Retrieved from: https://ojs.aaai.org/index.php/AAAI/article/view/12301(Accessed: May 6, 2023).

[47] Peng, J., Jury, E. C., Dönnes, P. and Ciurtin, C. (2021), 'Machine learning techniques for personalised medicine approaches in immune-mediated chronic inflammatory diseases: Applications and challenges'. Retrieved from: https://www.frontiersin.org/articles/10.3389/fphar.2021.720694(Accessed: May 4, 2023).
URL: *https://www.frontiersin.org/articles/10.3389/fphar.2021.720694*

[48] Pinochet, J. (2019), "black holes ain't so black': an introduction to the great discoveries of stephen hawking'. Retrieved from: https://doi.org/10.1088%2F1361-6552%2Fab0e9a(Accessed: April 26, 2023).

[49] Popsci (2015), 'How it works: Large hadron collider'. Retrieved from: https://www.popsci.com/how-it-works-large-hadron-collider/(Accessed: March 3, 2023).
URL: *https://www.popsci.com/how-it-works-large-hadron-collider/*

[50] Pramoditha, R. (2021), 'Overview of a neural networks learning process'. Retrieved from: https://medium.com/data-science-365/overview-of-a-neural-networks-learning-process-61690a502fa(Accessed: May 6, 2023).

[51] Qu, H., Li, C. and Qian, S. (2022), ‘Particle transformer for jet tagging’. Retrieved from: https://arxiv.org/abs/2202.03772(Accessed: May 5, 2023).

[52] Rossi, L. (2016), ‘The hl-lhc: a bright vision’. Retrieved from: https://home.cern/fr/node/2995 (Accessed: March 28, 2023).

[53] Ruby, D., Theerthagiri, P., Jacob, D. and Dr.Y.Vamsidhar (2020), ‘Binary cross entropy with deep learning technique for image classification’. Retrieved from: https://www.warse.org/IJATCSE/static/pdf/file/ijatcse175942020.pdf(Accessed: May 20, 2023).

[54] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. and Fei-Fei, L. (2015), ‘Imagenet large scale visual recognition challenge’. Retrieved from: https://link.springer.com/article/10.1007/s11263-015-0816-y(Accessed: May 20, 2023).

[55] Scott, D. W. (2010), ‘Histogram’. Retrieved from: https://doi.org/10.1002/wics.59(Accessed: March 03, 2023).

[56] Sidiya, A. and Li, X. (2020), ‘Style-based unsupervised learning for real-world face image super-resolution’. Retrieved from: https://www.intechopen.com/chapters/72399(Accessed: March 10, 2023).

[57] Smith, L. N. (2018), ‘The complete guide on overfitting and underfitting in machine learning’. Retrieved from: https://www.arxiv-vanity.com/papers/1803.09820/(Accessed: May 6, 2023).

[58] Sprindys, D. (2022), ‘Specially designed random forest function for high energy physics’. Retrieved from: https://bora.uib.no/bora-xmlui/bitstream/handle/11250/3015858/Thesis_final_v4.pdf?sequence=1&isAllowed=y (Accessed: March 9, 2023).

[59] Sumida, T. (2011), ‘Atl-phys-slide-2011-332’. Retrieved from: https://cds.cern.ch/record/1365731/files/ATL-PHYS-SLIDE-2011-332.pdf(Accessed: May 2, 2023).

[60] Teacher, Y. D. (2021), ‘Hyperparameter tuning: Grid search and random search’. Retrieved from: https://www.yourdatateacher.com/2021/05/19/hyperparameter-tuning-grid-search-and-random-search/(Accessed: March 10, 2023).

[61] The-CMS-Collaboration (2018), 'Search for black holes and sphalerons in high-multiplicity final states in proton-proton collisions at 13tev'. Retrieved from: https://arxiv.org/abs/1805.06013 (Accessed: March 9, 2023).

[62] *The LaTeX Project* (n.d.). Retrieved from: https://www.latex-project.org/(Accessed: May 20, 2023).

[63] Venugopal, D., Thangaiyan, J., Sikkandar, M., Waly, M., Pustokhina, I., Pustokhin, D. and Shankar, K. (2021), 'A novel deep neural network for intracranial haemorrhage detection and classification'. Retrieved from: http://www.techscience.com/cmc/v68n3/42461(Accessed: March 03, 2023).

[64] Vieira, D. and Paixao, J. (2019), 'Vector field neural networks'. Retrieved from: https://arxiv.org/abs/1905.07033(Accessed: March 10, 2023).

[65] Virdee, T. S. (2012), 'Physics requirements for the design of the atlas and cms experiments at the large hadron collider'. Retrieved from: https://royalsocietypublishing.org/doi/10.1098/rsta.2011.0459(Accessed: March 03, 2023).

[66] Webber, B. (1999), 'Fragmentation and hadronization'. Retrieved from: https://cds.cern.ch/record/419784/files/9912292.pdf (Accessed: March 9, 2023).

[67] Wynne, B. (2013), 'Measurement of the underlying event in pp collisions using the atlas detector and development of a software suite for bayesian unfolding'. Retrieved from: https://inspirehep.net/files/59bef42e3f579b7ff07359ed7f2cd343(Accessed: April 26, 2023).

[68] Xu, H. (2022), 'Vacuum state'. Retrieved from: https://encyclopedia.pub/entry/29492 (Accessed: March 16, 2023).

[69] Xu, P., Guo, Z., Liang, L. and Xu, X. (2021), 'Msf-net: Multi-scale feature learning network for classification of surface defects of multifarious sizes'. Retrieved from: https://www.mdpi.com/1424-8220/21/15/5125(Accessed: May 6, 2023).

[70] Zagoruyko, S. and Komodakis, N. (2017), 'Wide residual networks'. Retrieved from: https://arxiv.org/abs/1605.07146(Accessed: April 26, 2023).

[71] Zhang, X., He, K., Ren, S. and Sun, J. (2015), 'Deep residual learning for image recognition'. Retrieved from: https://arxiv.org/abs/1512.03385(Accessed: April 26, 2023).

List of Figures

List of Tables

A Code

A.1 CNN model

```python
class ConvModel(nn.Module):
    def __init__(self, dropout):

        super(ConvModel, self).__init__()
        self.conv1 = nn.Conv2d(
            in_channels=3, out_channels=16, kernel_size=3, padding=0)
        self.conv2 = nn.Conv2d(
            in_channels=16, out_channels=64, kernel_size=3, padding=0)
        self.conv3 = nn.Conv2d(
            in_channels=64, out_channels=256, kernel_size=3, padding=0)

        self.fc1 = nn.Linear(3*3*256, 1000)
        self.fc2 = nn.Linear(1000, 2)

    def forward(self, x: Tensor):
        x = self.conv1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.conv3(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 3)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        return x
```

Listing A.1: Implementation of the Convolutional model

A.2 ResidualBlock(34)

```python
class ResidualBlock(nn.Module):

    def __init__(self, in_channels, out_channels, stride=1, i_downsample=
   None):

        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels, out_channels,
                      kernel_size=3, stride=stride, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU())
        self.conv2 = nn.Sequential(
            nn.Conv2d(out_channels, out_channels,
                      kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(out_channels))
        self.i_downsample = i_downsample
        self.relu = nn.ReLU()
        self.out_channels = out_channels

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.conv2(out)
        if self.i_downsample:
            residual = self.i_downsample(x)
        out += residual
        out = self.relu(out)
        return out
```

Listing A.2: Implementation of the Residual Block

A.3 Bottleneck(50,101)

```
class Bottleneck(nn.Module):
    expansion = 4

    def __init__(self, in_channels, out_channels, i_downsample=None,
   stride=1):
        super(Bottleneck, self).__init__()

        self.conv1 = nn.Conv2d(in_channels, out_channels,
                               kernel_size=1, stride=1, padding=0)
        self.batch_norm1 = nn.BatchNorm2d(out_channels)

        self.conv2 = nn.Conv2d(out_channels, out_channels,
                               kernel_size=3, stride=stride, padding=1)
        self.batch_norm2 = nn.BatchNorm2d(out_channels)

        self.conv3 = nn.Conv2d(
            out_channels, out_channels*self.expansion, kernel_size=1,
   stride=1, padding=0)
        self.batch_norm3 = nn.BatchNorm2d(out_channels*self.expansion)

        self.i_downsample = i_downsample
        self.stride = stride
        self.relu = nn.ReLU()

    def forward(self, x):
        identity = x.clone()
        x = self.relu(self.batch_norm1(self.conv1(x)))
        x = self.relu(self.batch_norm2(self.conv2(x)))
        x = self.conv3(x)
        x = self.batch_norm3(x)

        # downsample if needed
        if self.i_downsample is not None:
            identity = self.i_downsample(identity)
        # add identit\n,y
        x += identity
        x = self.relu(x)
        return
```

Listing A.3: Implementation of the Bottleneck block

A.4 Residual network model(34)

```python
class ResNetModel(nn.Module):
    def __init__(self, block, layers, num_classes=2):

        super(ResNetModel, self).__init__()
        self.conv1 = nn.Conv2d(
            in_channels=3, out_channels=64, kernel_size=7, stride=2,
 padding=3)
        self.batch1 = nn.BatchNorm2d(64)

        self.inplanes = 64
        self.layer0 = self._make_layer(block, 64, layers[0], stride=1)
        self.layer1 = self._make_layer(block, 128, layers[1], stride=2)
        self.layer2 = self._make_layer(block, 256, layers[2], stride=2)
        self.layer3 = self._make_layer(block, 512, layers[3], stride=2)


        self.fc0 = nn.Linear(2048, num_classes)

    def forward(self, x: Tensor):
        x = self.conv1(x)
        x = self.batch1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)

        x = self.layer0(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)

        x = torch.flatten(x, 1)
        x = self.fc0(x)

        return x

    def _make_layer(self, block, planes, blocks, stride=1):
        downsample = None
        if stride != 1 or self.inplanes != planes:

            downsample = nn.Sequential(
                nn.Conv2d(self.inplanes, planes, kernel_size=1, stride=
 stride),
                nn.BatchNorm2d(planes),
            )
        layers = []
        layers.append(block(self.inplanes, planes, stride, downsample))
        self.inplanes = planes
        for i in range(1, blocks):
            layers.append(block(self.inplanes, planes))

        return nn.Sequential(*layers)
```
Listing A.4: Implementation of the ResNet34 model

A.5 Residual network model(50,101)

```
class ResNet(nn.Module):
    def __init__(self, ResBlock, layer_list, num_classes, num_channels=3):
        super(ResNet, self).__init__()
        self.in_channels = 64

        self.conv1 = nn.Conv2d(
            num_channels, 64, kernel_size=7, stride=2, padding=3, bias=
False)
        self.batch_norm1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU()
        self.max_pool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.layer1 = self._make_layer(ResBlock, layer_list[0], planes=64)
        self.layer2 = self._make_layer(
            ResBlock, layer_list[1], planes=128, stride=2)
        self.layer3 = self._make_layer(
            ResBlock, layer_list[2], planes=256, stride=2)
        self.layer4 = self._make_layer(
            ResBlock, layer_list[3], planes=512, stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc0 = nn.Linear(512*ResBlock.expansion, 1000)
        self.fc1 = nn.Linear(1000, num_classes)

    def forward(self, x):
        x = self.relu(self.batch_norm1(self.conv1(x)))
        x = self.max_pool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.fc0(x)
        x = self.fc1(x)

        return x

    def _make_layer(self, ResBlock, blocks, planes, stride=1):
```

```python
        ii_downsample = None
        layers = []

        if stride != 1 or self.in_channels != planes*ResBlock.expansion:
            ii_downsample = nn.Sequential(
                nn.Conv2d(self.in_channels, planes*ResBlock.expansion,
                          kernel_size=1, stride=stride),
                nn.BatchNorm2d(planes*ResBlock.expansion)
            )
        layers.append(ResBlock(self.in_channels, planes,
                    i_downsample=ii_downsample, stride=stride))
        self.in_channels = planes*ResBlock.expansion

        for i in range(blocks-1):
            layers.append(ResBlock(self.in_channels, planes))

        return nn.Sequential(*layers)
```

Listing A.5: Implementation of the class used to generate ResNet 50 and ResNet101 model

A.6 Pre-trained resnet34 imagenet model

```
class Resnet34(nn.Module):
    def __init__(self):
        super(Resnet34, self).__init__()
        self.model = resnet34(weights=ResNet34_Weights.IMAGENET1K_V1)
        self.my_new_layer = nn.Sequential(nn.Linear(1000, 2))

    def forward(self, x):
        x = self.model(x)
        x = self.my_new_layer(x)
        return x
\end{verbatim}

\section{Pre-trained resnet50 imagenet model}
\begin{verbatim}
class Resnet50(nn.Module):
    def __init__(self):
        super(Resnet50, self).__init__()
        self.model = resnet50(weights=ResNet50_Weights.IMAGENET1K_V2)
        self.my_new_layer = nn.Sequential(nn.Linear(1000, 2))

    def forward(self, x):
        x = self.model(x)
        x = self.my_new_layer(x)
        return x
```

Listing A.6: Implementation of the pre-trained ResNet34 and ResNet50 model

A.7 Custom cross-entropy and custom softmax

```python

def logSoftmax(x, root):
    x_pow = x.sign() * x.abs().pow(1/root)
    x_exp = torch.exp(x_pow)
    x_exp_sum = torch.sum(x_exp, 1, keepdim=True)
    return torch.log(x_exp/x_exp_sum)


class CustomLoss(nn.Module):
    def __init__(self, root):
        super(CustomLoss, self).__init__()
        self.root = root

    def forward(self, input, targets):
        batch_size = input.size()[0]
        input = logSoftmax(input, self.root)
        input = input[range(batch_size), targets]
        return -torch.sum(input) / batch_size
```

Listing A.7: Implementation of the custom loss function

B Appendix

B.0.1 GitHub Repository

Here is the GitHub repository of the code: https://github.com/591291-hvl/Bachelor

B.0.2 Figures

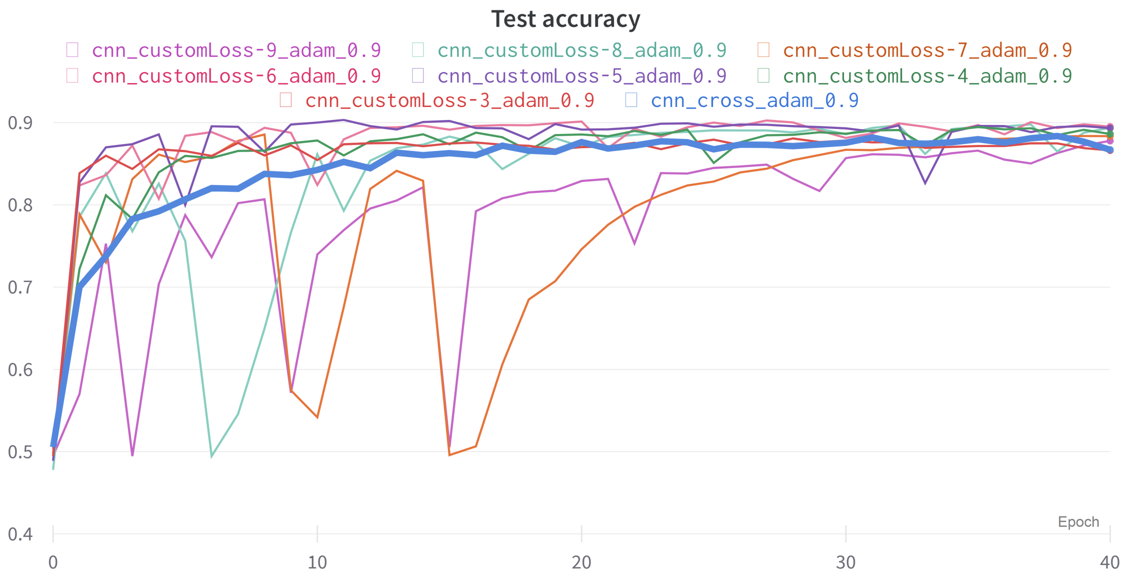


Figure B.1: CNN LossFunctions Test Accuracy results run 1

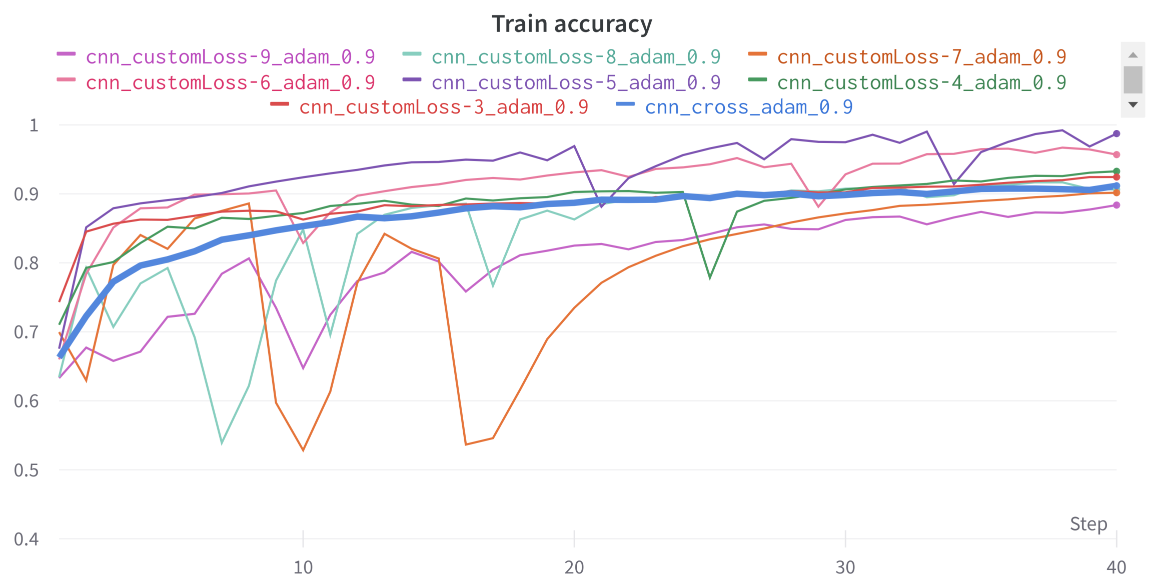


Figure B.2: CNN LossFunctions Train Accuracy results run 1

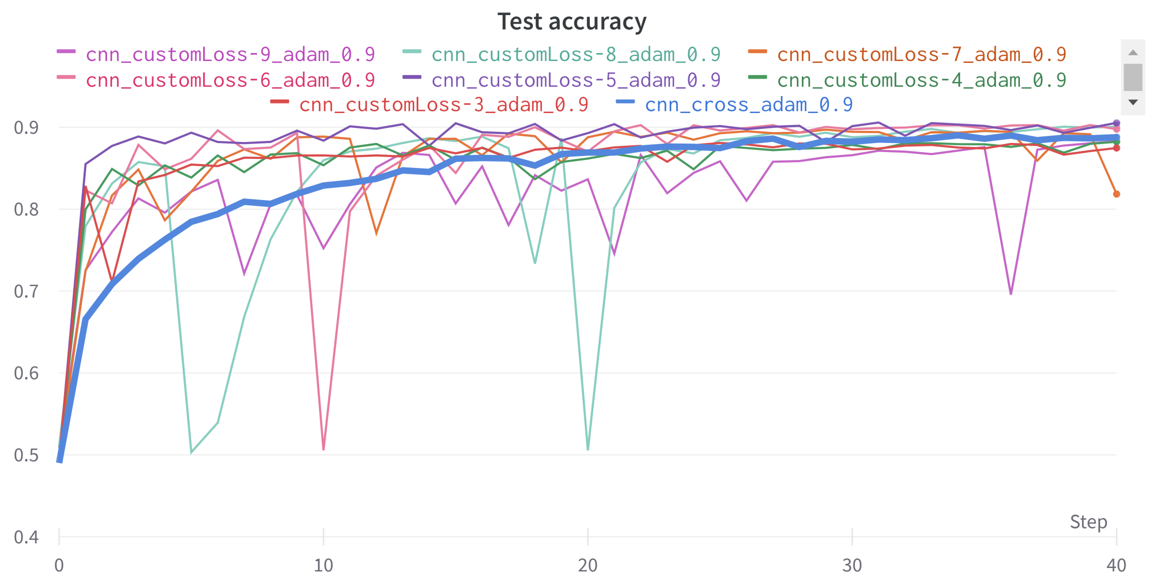


Figure B.3: CNN LossFunctions Test Accuracy results run 2

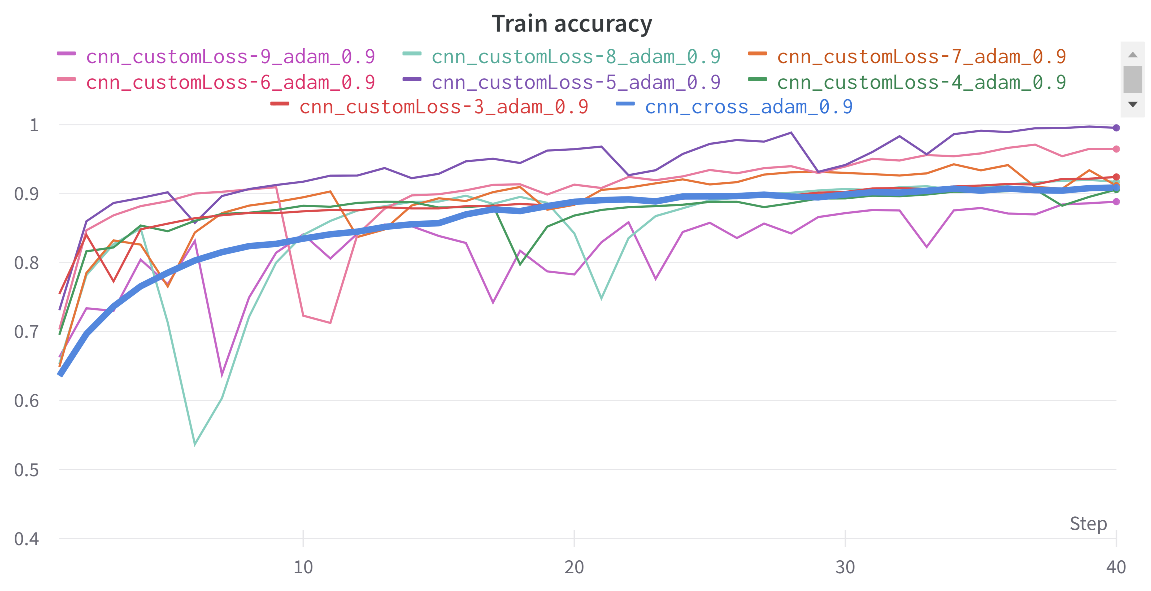


Figure B.4: CNN LossFunctions Train Accuracy results run 2

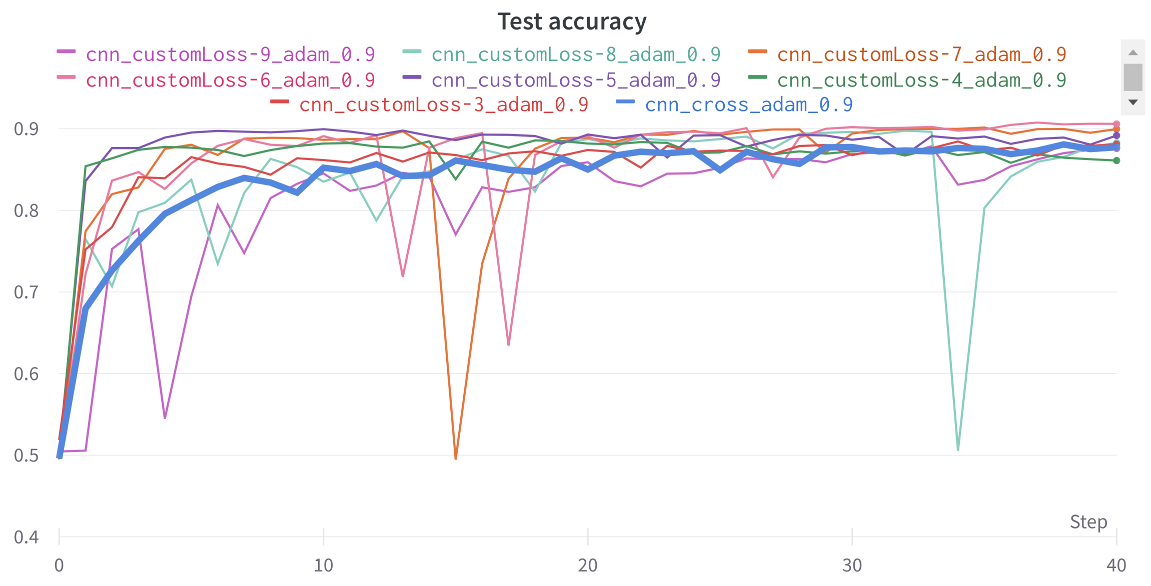


Figure B.5: CNN LossFunctions Test Accuracy results run 3

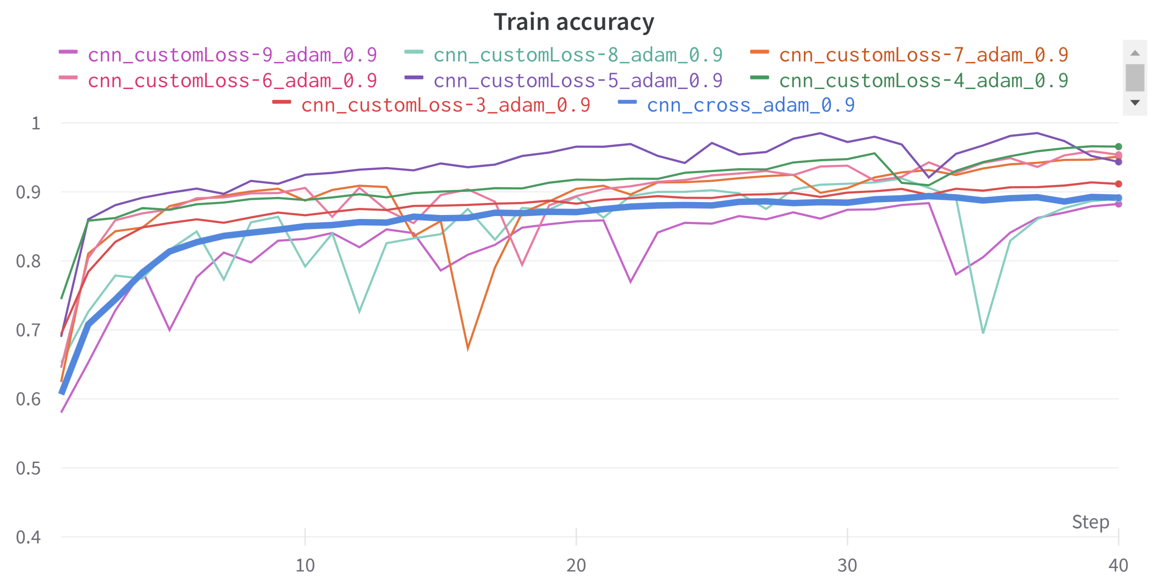


Figure B.6: CNN LossFunctions Train Accuracy results run 3

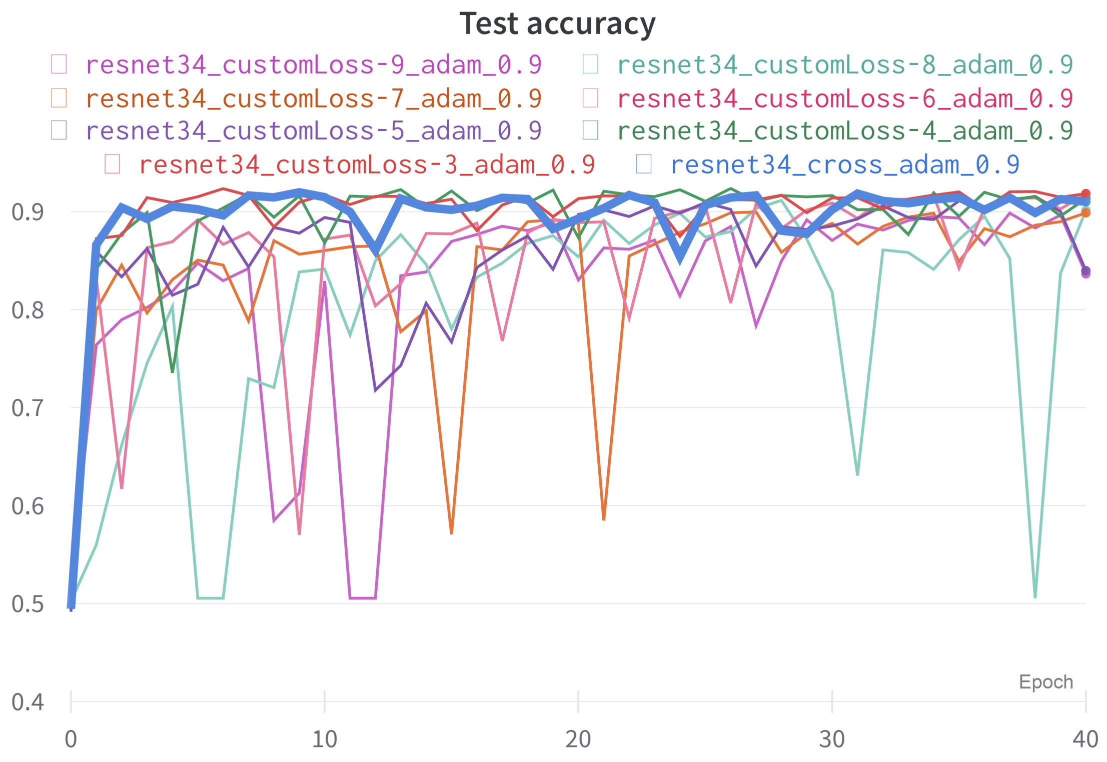


Figure B.7: ResNet LossFunctions Test Accuracy results run 1

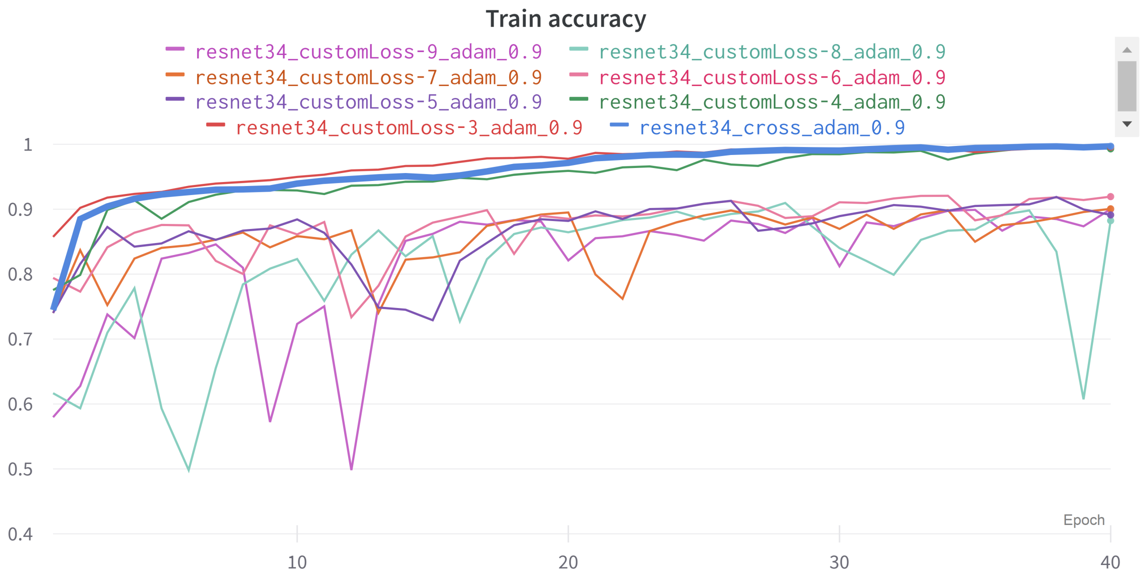


Figure B.8: ResNet LossFunctions Train Accuracy results run 1

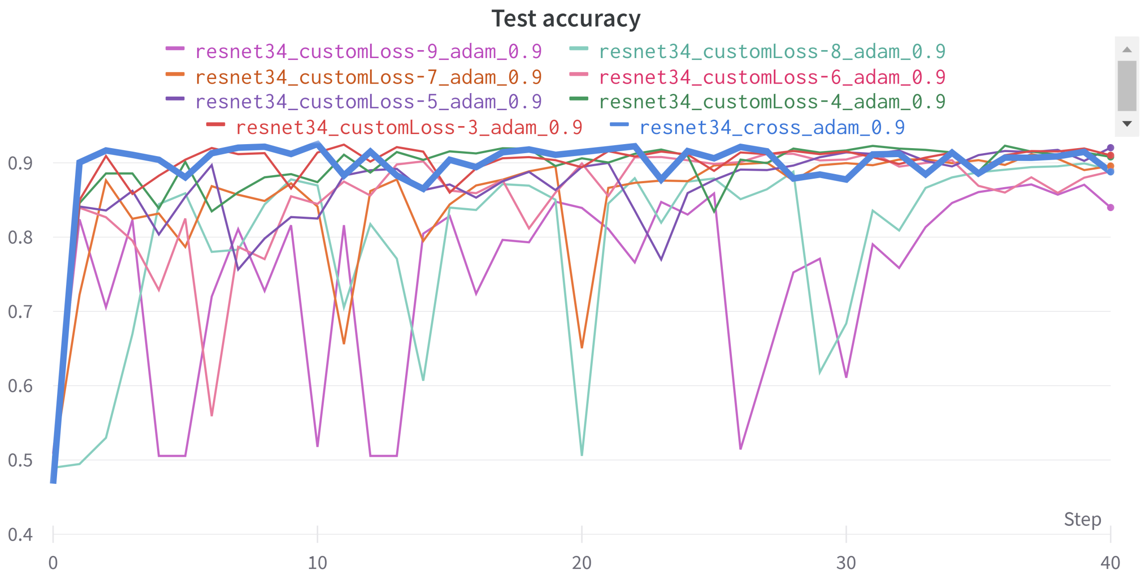


Figure B.9: ResNet LossFunctions Test Accuracy results run 2

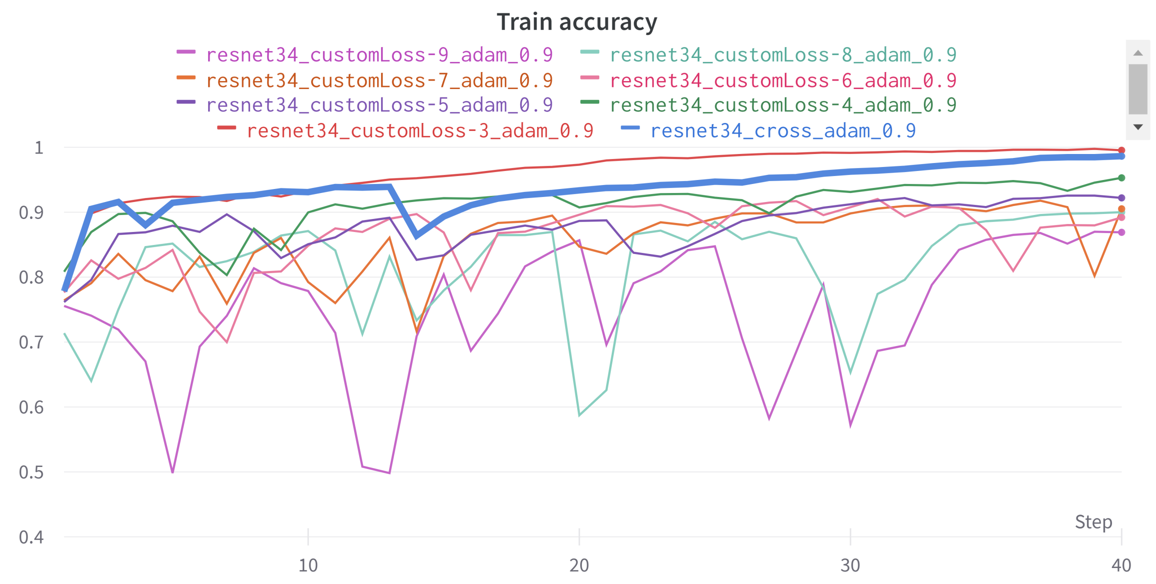


Figure B.10: ResNet LossFunctions Train Accuracy results run 2

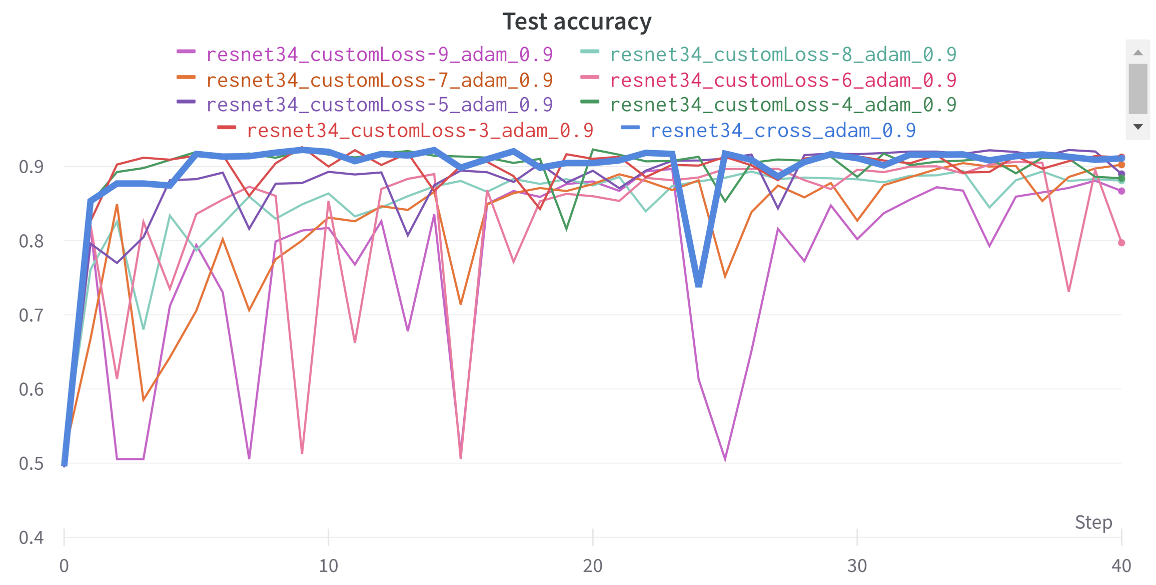


Figure B.11: ResNet LossFunctions Test Accuracy results run 3

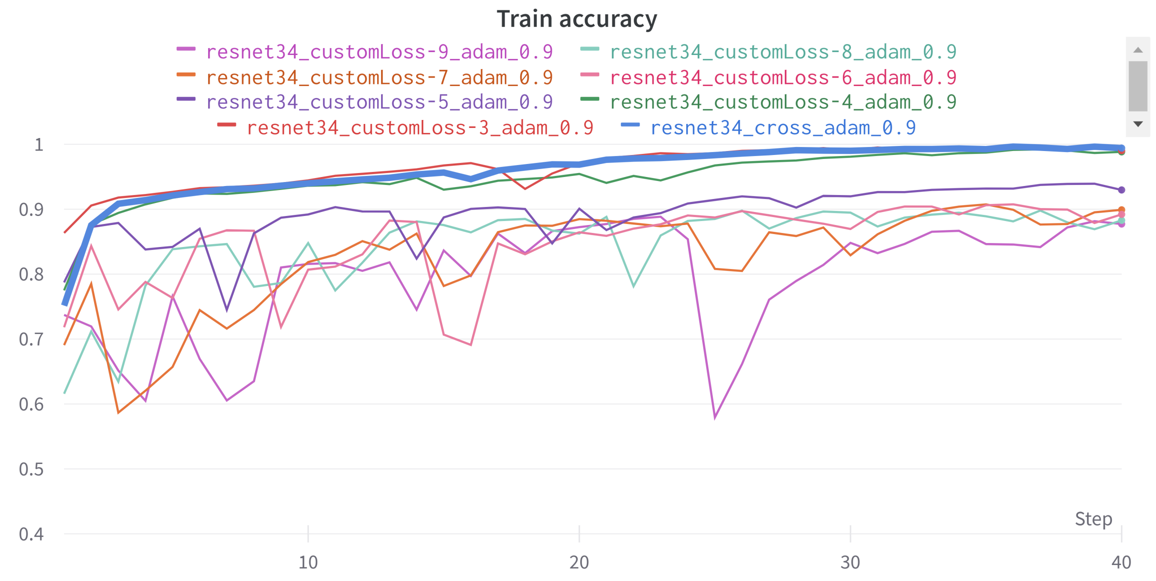


Figure B.12: ResNet LossFunctions Train Accuracy results run 3

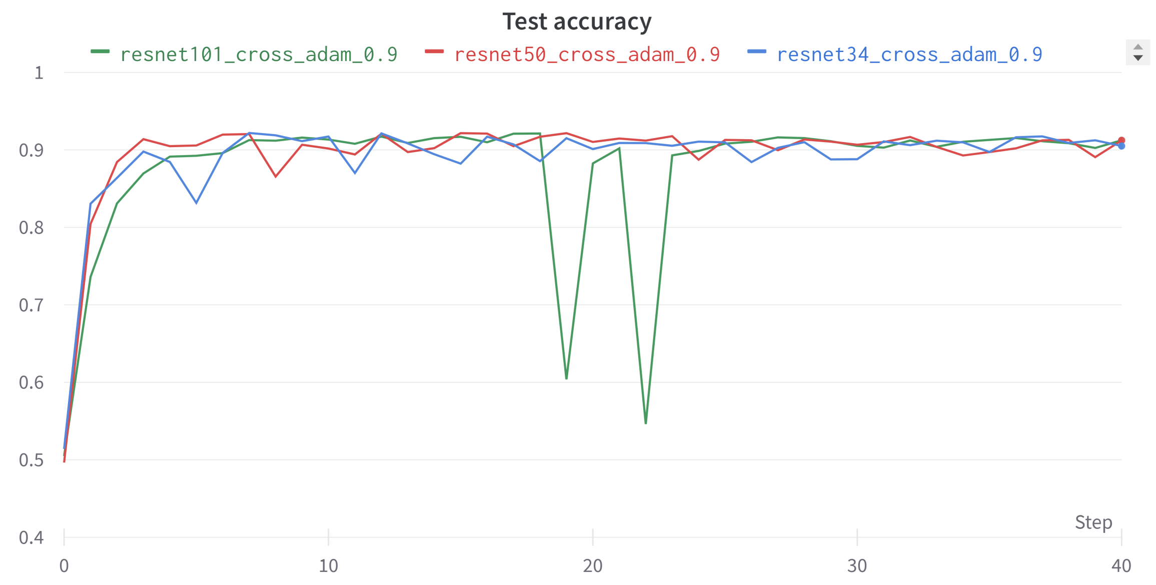


Figure B.13: Comparison of ResNet34, 50, 101 - Accuracy on test dataset for 40 epochs

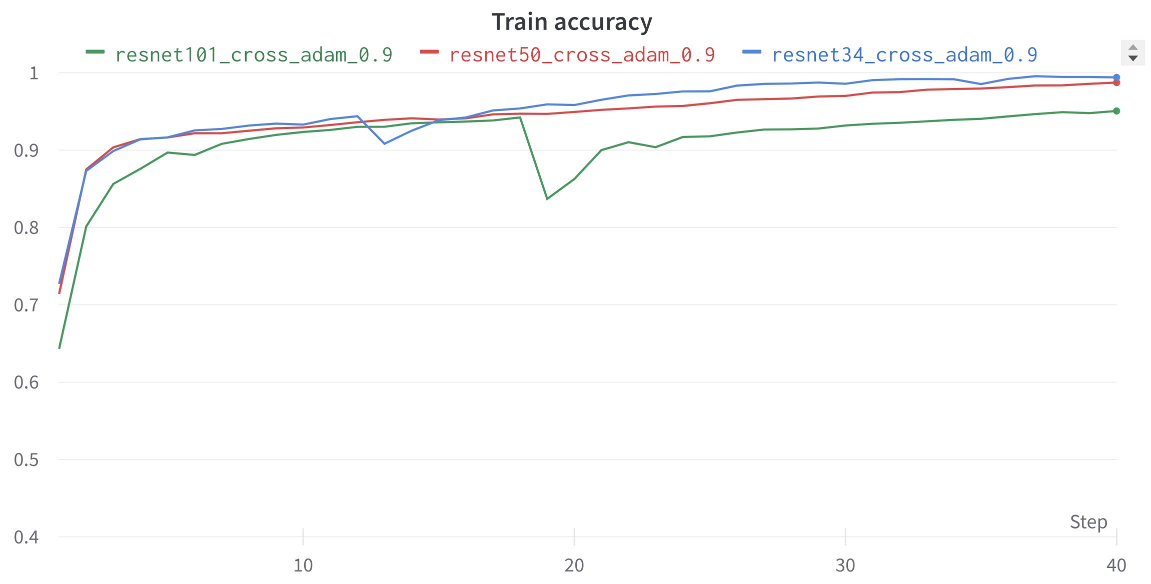


Figure B.14: Comparison of ResNet34, 50, 101 - Accuracy on train dataset for 40 epochs

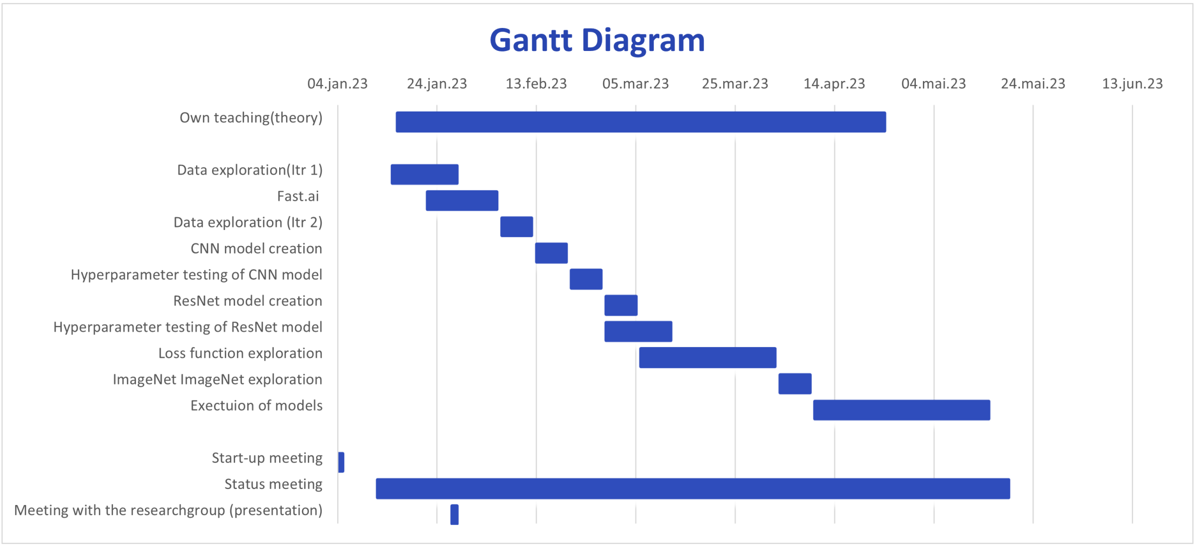


Figure B.15: GANTT Chart

	Risk	Cause	Probability	Consequences	Risk product	Measure
1	Theory about what we should do is misunderstood.	Lack of understanding of the physics behind what we do.	5	3	15	Regular contact with project advisors where we can ask questions about anything we are curious about.
2	Collision regarding the times groups use the shared computer.	Poor planning and communication between the groups.	2	3	5	Set days where a group is supposed to use the shared computer.
3	Internal conflicts	The group does not realize that they have a common goal and only with eachother will the result become good	3	4	12	Using an internal advisor as a mediator.
4	Not enough time during the development process	Bad use of time or bad estimation of time needed to complete a task	3	3	9	Make sure we have development iterations to spare incase we does not reach the goals set for a iteration
5	Disease	Bad hygiene habits	3	1	3	Start having good hygienic habits

Figure B.16: Risk assessment