Master Thesis

Neural Networks for Mixed Image Classification

Valon JASHARI
01634834

December 2023

Supervisor:
Univ.Prof. Dipl.-Ing. Dr.-Ing. Norbert GÖRTZ

This thesis has been submitted in compliance with the requirements for the degree of Diplom-Ingenieur at

Technische Universität Wien
Faculty of Electrical Engineering and Information Technology
Institute of Telecommunications

I confirm that this master thesis is my own work and I have documented all sources and material used.

Ich versichere, dass ich diese master thesis selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Vienna, December 2023 Valon JASHARI

Acknowledgments

I would like to thank my advisor, Georg Pichler, very much for the enjoyable work we've done together, his endless enthusiasm, his helpful suggestions, and the stimulating conversations we've had throughout this project. He gave me his unwavering support and was always available when I needed guidance, and his comments were crucial to finishing my thesis.

Professor Norbert Görtz deserves special thanks for inviting me to participate in this research project and for his insightful remarks and constructive feedback. His guidance and encouragement were invaluable to me at every step.

Finally, I want to thank my family, especially my wife, Festina, for always being there for me and supporting me. I also want to thank my friends Fares, Pajtim, and Petrit for always having my back. Their love and support have helped me stay inspired and on track with my goals, and I'm thankful for that. I would not have been able to do it without your love and support.

Sincere appreciation goes out to everyone who helped in any way with this research. Thank you for your unwavering support and encouragement throughout this journey.

Abstract

Convolutional Neural Networks (CNNs) are specialised Neural Networks (NNs) that are particularly well-suited to image recognition tasks.

Therefore, this thesis investigates how to combine digits in an image and then recognise them using a CNN trained on a dataset of images containing digit combinations.

For the Initial Model (M1), the results of a trained CNN for the classification of single-digit images when fed with mixed input images. Then, image recognition experiments are performed both with images containing the single digits for which the CNNs were trained and with mixed images, and the recognition results are evaluated numerically.

In addition, the new concept of Improved Model (M2) (this combinations model used to have 55 outputs), in which overlayed images of two digits are randomly combined, is used to train the CNN for recognising two digits in overlayed images and thus improve its performance in recognising mixed images.

In the end, we compare M1 single-digit and multi-digit results to M2 results.

Key words: CNNs, single-digit, multi-digits, models (M1 and M2).

Kurzfassung

Konvolutionäre Neuronale Netzwerke (Konvolutionäre Neuronale Netzwerkes (KNNs)) sind spezialisierte NN, die besonders gut für Bilderkennungsaufgaben geeignet sind. Daher untersucht diese These, wie man Ziffern in einem Bild kombiniert und sie dann mithilfe eines KNNs erkennt, das auf einem Datensatz von Bildern mit Zifferkombinationen trainiert wurde.

Für das Initial Model (M1) werden die Ergebnisse eines trainierten KNNs zur Klassifizierung von einzelnen Bildern ausgewertet, wenn es mit gemischten Eingangsbildern versorgt wird. Dann werden Bilderkennungsexperimente sowohl mit Bildern durchgeführt, die die einzelnen Ziffern enthalten, für die die KNNs ausgebildet wurden, als auch mit gemischten Bildern. Die Ergebnisse der Erkennung werden numerisch ausgewertet.

Zusätzlich wird das Konzept des Verbesserten Modells (M2) verwendet, bei dem mehrere Ziffern zufällig kombiniert werden, um das KNN darauf zu trainieren, doppelte Ziffern zu erkennen und somit die Leistung bei der Erkennung von gemischten Bildern zu verbessern. Darüber hinaus wird das neue Konzept von M2 (dieses Kombinationsmodell wird 55 Ausgänge haben), bei dem überlagerte Bilder von zwei Ziffern zufällig kombiniert werden, verwendet, um die KNN für die Erkennung von zwei Zahlen in überlagerten Bildern zu schulen und so ihre Leistung bei der Erkennung gemischter Bilder zu verbesser

Abschließend werden die Ergebnisse von M1 für Einzelziffern und Mehrfachziffern mit den Ergebnissen von M2 verglichen.

Schlüsselwörter: KNNs, Einzelziffer, Mehrfachziffern, Modelle (M1 und M2).

Contents

1. Introduction

Recently, interest in Artificial Intelligence (AI) and Machine Learning (ML) has increased. Global tech giants such as Microsoft, Google, Amazon, Netflix and Facebook are optimising the use of AI or ML in healthcare, logistics, finance, music, revenue generation, real estate and communications

The classification of images in computer vision is an important topic of Deep Learning (DL), focusing on NNs and CNNs. DL algorithms produce NNs that map high-dimensional input data (such as images) to low-dimensional output vectors that can be used to classify image content. In this thesis, we use CNNs for image recognition tasks [1].

1.1. Problem Statement

The problem is what classification results CNNs provide when they are trained with image databases containing certain single digits (e.g., the handwritten numbers 0, 1,..., 9) in each image of a training database such as Modified National Institute of Standards and Technology (MNIST), and then mixed images (e.g., the numbers "1" and "2" in an image) are fed into the network for recognition. The training process for CNNs is designed so that after several pooling steps, the final fully connected recognition network produces a large output in a particular vector component and small outputs in all others for each element to be recognised (i.e. 10-dimensional output vectors for the classification of the handwritten characters 0, 1,..., 9).

The main motivation in this thesis is what results a CNNs trained to classify single digits produces when fed with mixed input images. Subsequently, CNNs are trained for both mixed images containing single digits and multi-digit images with the aim of evaluating the recognition results for single digits and multi-digit images.

New images are generated by using a CNN trained on single-digit digits to recognise single-digit and then two-digit digits with the initial model (M1). Then, using an improved approach (M2), the multi-digits are used to train a CNN to recognise multi-digits (a new concept two-digit combination-generating method produced 55 pairings)

Thus, in this thesis, simulations and reports on the results for single-digit recognition are performed. Concepts are developed to use a CNN trained on single-digit

digits and extended for multi-digit digits, and the new concept is compared with the performance of CNNs trained on multi-digits [2].

In this thesis, we were interested in combining digits and determining their accuracy after training and testing the models. We also experimented with different network parameters, CNN layers and approaches to improve the accuracy of the models.

1.2. Thesis overview

This thesis is organised into six chapters.

- Chapter 2 gives a brief introduction to the basics of NNs types, applications, architecture, components, loss and activation functions of neural networks.
- Chapter 3 introduces the basics of CNNs, such as convolutional operations, pooling, max-pooling and fully connected layers.
- Chapter 4 introduces the classification approaches, starting with single-digit classifications, then multi-digit classifications with M1, briefly explaining the methods used to train CNNs to recognise single and multi-digit images, and the use of parameters (epochs, batch size and threshold). The final part of this chapter looks in more detail at the combination model (M2), where randomly combined multi-digits are used to train CNNs to recognise multi-digits.
- In chapter 5, we discuss the accuracy and performance results for both single-digit and multi-digits with M1 and new concepts (M2).
- Last but not least, in chapter 6, conclusions based on the outcomes of chapter 5 and an outlook for future work.

2. Neural Networks Fundamentals

2.1. Background

This chapter presents the basic concepts of AI technologies used in this thesis, such as ML, DL, NNs, tensors, and data transformation techniques.

2.2. Artificial Intelligence

The relationships between AI, ML, DL and NN are shown in figure 2.1. Each of these components is a constituent aspect of the previous method. NNs serves as a basic framework for deep DL algorithms, which in turn represent subdomains within the broader domain of ML. In addition, ML itself is a subdomain within the larger domain of AI.

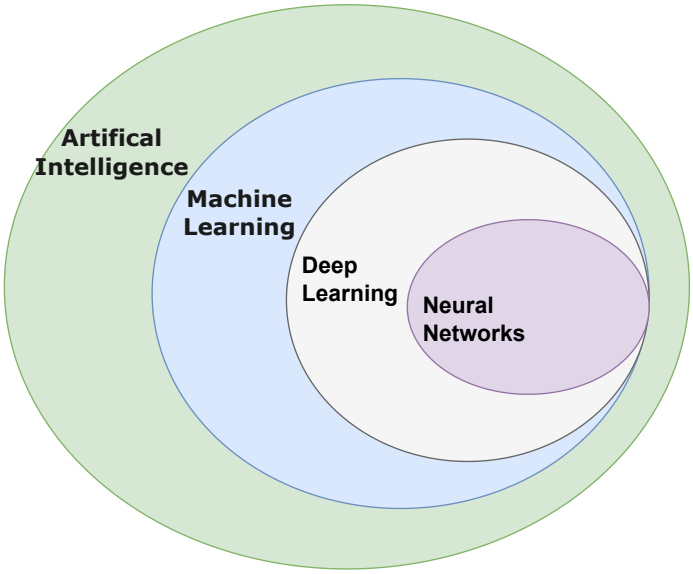

Figure 2.1.: Relation of AI, ML, DL and NN

AI encompasses learning, reasoning, perception, language intelligence, problem solving and the adaptation of new solutions to the system that intelligently facilitates

user learning, demonstration and consultation. AI refers to modern machines that can perform complex functions and do muscle work in an "intelligent" way, sometimes defined as a discipline of computer science that creates "intelligent" smart machines that can perform tasks [3].

2.2.1. Applications of AI

AI has applications for corporate intelligence, data science, cybersecurity, health plans and patient care. Drug research, clinical trials, speech recognition, hospitality advertising, marketing automation, entertainment, etc. They are also used in autonomous vehicles, robots, financial research and trading, virtual assistants and personalised recommendation systems. Agriculture, education and environmental monitoring use AI to improve efficiency and decision-making [4].

2.3. Machine Learning

ML enables computers to learn on their own by using training or sample data to create mathematical models that enable decision making without scripting. Computers in ML learn autonomously from interactions and actions so that they can learn from experience [5].

ML takes a unique approach to input elements compared to traditional programming. In a training phase, exact answers are collected from labeled data and trained with other data to obtain an answer. In the second phase, the model ML uses the rules from the first phase and new data, as shown in 2.2 [6].

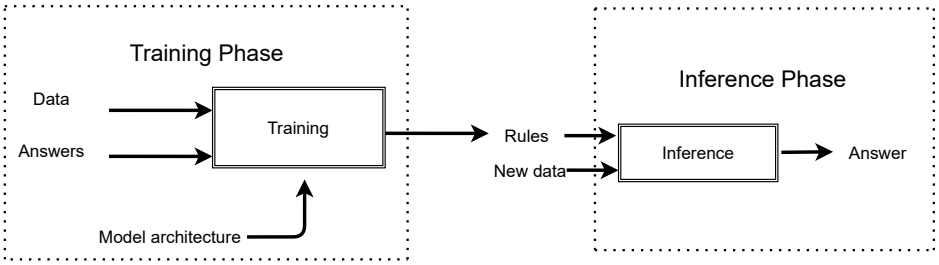

Figure 2.2.: Machine Learning model

2.3.1. Applications of ML

The applications of ML are wide-ranging, such as image and speech recognition, natural language processing, medical diagnosis, characterization of physical objects,

etc. ML revolutionises problem solving and assessment in various fields. Its versatility and social impact can be seen in applications in banking, cyber security and driverless cars, also in many other areas [7].

- Agriculture, Chemistry, Mathematics, Physics
- Cognitive Modeling and Computer programming
- Education, Music, Game playing (chess, checkers, etc)
- Image and Speech Recognition, Natural Language Processing
- Medical Diagnoses, Physical Object Characterizations
- Expert Systems (AI programs, high-performance systems)

2.3.2. Types of Machine Learning

The classes of ML algorithms are based on the training method and the availability of results during training. The first category we used is explained along with three others [8].

1. Supervised Learning (SL) is one of the most popular methods to learn from multiple examples. The function SL is used to train a mapping function to estimate y from the input parameter x. Table 2.1 lists the applications of SL, including the categorization of images based on this thesis. CNNs is used for image data, as explained in the chapter 3, Hybrid Neural Networks (HNNs) are used in autonomous driving and Recurrent Neural Networks (RNNs) used for speech recognition and machine translation [9].

Input(x)	Output(y)	Application	Type of NN
Home features	Price	Real Estate	Standard NN
Ad, user info	Click on add?	Online Advertising	Standard NN
Image	Object (1,...,100)	Classification	CNN
Audio	Text transcript	Speech recognition	RNN
English	German	Machine translation	RNN
Image, Radar info	Connected cars	Autonomous driving	HNN

Table 2.1.: Supervised Learning Applications

2. Unsupervised Learning (USL) uses a statistical approach to derive rules from input data that identifies transformations of pre-existing patterns without knowing the targets or outputs.

3. Semi-Supervised Learning (SSL) is an unique algorithm that combines USL and SL, where in the first case all observations have output labels and in the second none. This paradigm works well when some parameters are labelled but most of the data is not.

4. Reinforcement Learning (RL) is one of the three core paradigms of ML, an intermediate learning approach that allows agents to gather environmental information, perform actions and maximise rewards..

Data models in SL applications have two basic types of data [10].

1. Structured Data (SD) - represents clearly defined data, similar to a database.

2. Unstructured Data (USD) - No database is defined or maintained; this type also includes music and images as "data"; text or image pixels are used as input. Table 2.1 shows that USD are more common than SD, even though they are more difficult to train; this type of data is used in this thesis [11].

2.4. Deep learning

DL, a subset of AI, applies multiple layers to extract features from input to output, mimicking the data processing process of the human brain. Within a few days or hours, DL can analyse huge amounts of unstructured data that would take humans decades [12]. It automates processes using Artificial Neural Network (ANN), a non-linear approach resembling the human brain's network of neural nodes, which allows for faster and more efficient decision-making.

2.4.1. Applications

The component DL is a state-of-the-art solution that can be applied to many different domains. Although it uses hierarchical methods and has a large number of layers to analyse, it has found wide application in many areas [13].

- Computers and machine vision
- Speech and audio recognition
- Social network filtering and natural language translation

- Medical research and biometrics
- Image processing and aerospace engineering

Autonomous vehicles and robotics have also found applications for DL, especially in the areas of object recognition and path planning.

2.5. Logistic Regression

Logistic Regression (LR) or a logit model is a statistical approach used for classification and prediction analysis to estimate the probability of an event (belonging to one or two classes). It predicts binary outcomes. It is used to SL models. [14].

2.5.1. Notation of Logistic Regression

The estimated $\hat{y}$ represents the probability that a cat image is represented in a picture, expressed as $\hat{y} = P(y = 1|\vec{x})$ $0 \leq \hat{y} \leq 1$, ($\vec{x}$ is a $\vec{\eta_x}$ dimensional vector)[15].

The output $\hat{y}$ indicates the probability that a cat is present in the image, using two parameters $\vec{w}$ and b, where $\vec{w}$ is a dimensional vector ($\vec{w} \in \mathbb{R}^{\vec{\eta_x}}$) and b is a real number (b $\in \mathbb{R}$), respectively. The formula for calculating $\hat{y}$ is expressed in equation 2.1, which is intended to ensure that $\hat{y}$ is equal to 1, as shown in figure 2.3. The symbol σ(z) stands for the sigmoid function, as explained earlier in this chapter. It is used as a σ(z), while the LR has a core function, the logistic function, exhibiting an S-shaped curve, and transforms input values into probabilities within the range 0 to 1.

$$\hat{y} = \mathrm{a} = \sigma(\vec{w}^T\vec{x} + \mathrm{b}) = \sigma(z). \tag{2.1}$$

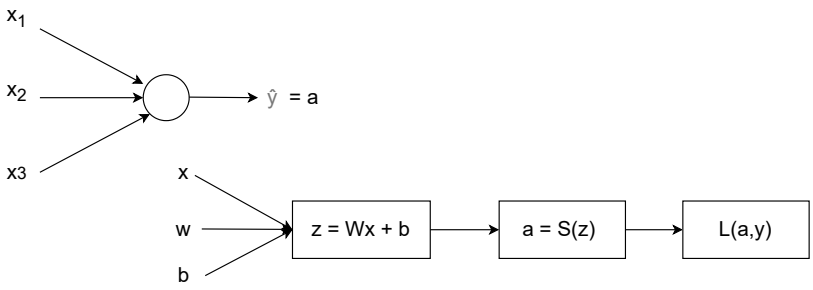

Figure 2.3.: Logistic regression overview

In figure 2.3, the expression a = S(z) represents the application of the sigmoid activation function to the output $\hat{y}$, while L represents the loss function L(a,y).

2.6. Types of Loss Function

The Loss Function (LF) measures the error per observation by measuring the error between the predicted value of the model and the actual value in that sample. It has two values: y the true label and $\hat{y}$ the prediction of the model. Only the error per sample is calculated. If the difference between the predicted result and the target value is very small (the performance of the trained model is better if the predictions are closer to the target values), then the loss value is also small, otherwise the loss value is very high.

In my thesis, I discuss various LFs for faster image training, including Mean Square Error (MSE) and Cross-Entropy (CE) functions, which are better suited to Tensorflow and Keras, to minimize the difference between actual and expected output [16]. In classification problems, there are three types. In this thesis, different LFs for faster image training are discussed, including MSE and CE functions, which are more suitable for Tensorflow and Keras to minimize the difference between actual and expected output [16]. Two categories of classification problems are used in this thesis.

- Binary classification: only two class labels, for example [1], [0]. The task of binary classification is to predict one of two class labels.
- Multi-class classification: more than two class labels are in the target, but only one class label is assigned to an input, for example [010], [001], [100]. Multi-class classification uses the task of predicting one of more than two class labels. The categorical cross-entropy is used in this classification task.

2.6.1. Mean Square Error

MSE is a regression method that computes the difference between expected and estimated values, squares this difference, and then calculates the mean of the squared differences, as represented in equation 2.2, where 'n' denotes the number of data points [17].

$$\mathrm{MSE}(x) = \frac{1}{\mathrm{n}}\sum_{i=1}^{\mathrm{n}}(y_i - \hat{y}_i)^2 \tag{2.2}$$

The Root Mean Square Error (RMSE) is a derivative of the MSE, which is the square root of the MSE's scalar value, expressed as $\mathrm{RMSE} = \sqrt{\mathrm{MSE}} = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}}$.

2.6.2. Cross-Entropy

The CE, also known as the logistic loss function, is mostly used for classification tasks. It calculates the difference between two probability distributions, expressed as

in equation 2.3, where p(x) is the probability of a true event, q(x) is the probability of an estimated event, and N is the number of classes. Categorical and binary CE are used in this thesis.

$$\text{Cross-Entropy (p,q)} = -\sum_{x=1}^{N}[p(x)log(q(x))] \tag{2.3}$$

2.6.2.1. Binary Cross-Entropy

In binary CE, the target class label is binary (0 or 1), which represents the loss function for the binary classification task [18].

$$\text{Binary Cross-Entropy (p,q)} = -[p(x)\log(q(x)) + (1-p(x))log(1-q(x))] \tag{2.4}$$

It is a special type of categorical CE, applicable when there are only two classes. It is commonly used with a sigmoid activation function in the output layer, as expressed by equation 2.4 and illustrated in the two cases below, where L represents the loss function for this type of CE [19].

1 $p(x) = 1 \rightarrow L(q,p) = -\log(q)$, with q optimized to be as large as possible.

2 $p(x) = 0 \rightarrow L(q,p) = -\log(1-q)$, with q optimized to be as small as possible.

In the above two cases, the objective of the binary CE loss is aligned by minimizing the loss when the estimated probability (q(x)) matches the true class label (p(x)).

2.6.2.2. Categorical Cross-Entropy

Categorical CE is used for classification tasks with multiple classes. This is a special case of CE where the target is a one-hot vector (1 is only in one place, the rest is 0) and the loss is calculated only for the hot class (N=1), which means that the probability distribution of a true event is 1 (denoted as p(x)=1), expressed as in equation 2.5 [20].

$$\text{Categorical Cross-Entropy (p,q)} = -\sum_{x=1}^{N}[p(x)log(q(x))] = -log(q(x)) \tag{2.5}$$

2.6.3. Cost Function

The Cost Function (CF) calculates the average of the loss functions over a training dataset to optimize model performance by aggregating these values and adjusting the model parameters ($\vec{w}$ and b) during training and to determine the performance of the

NN over the entire dataset [21]. The cost function determines the optimal parameters of the NN as expressed by equation 2.6, where m is the number of training examples.

$$J(w, b) = \frac{1}{m}\sum_{i=1}^{m} L(q(x), p(x)) = -\frac{1}{m}\sum_{i=1}^{m}[p(x)\log(q(x)) + (1 - p(x))\log(1 - q(x))] \tag{2.6}$$

2.6.4. Gradient Descent

The gradient descent aims to find the global best value for parameters $\vec{w}$ and b (the goal is to minimise the CF). These parameters are shown by partial derivatives and learning rate (α), as shown in equation 2.7 [22].

$$w \leftarrow w - \alpha\frac{\partial J(w, b)}{\partial w} \qquad \& \qquad b \leftarrow b - \alpha\frac{\partial J(w, b)}{\partial b} \tag{2.7}$$

NN calculations are organised into forward and backward propagation steps, with the first computing the NN's output, with the calculation from left to right, and the second calculating gradients (derivatives), with the calculation from right to left (red arrow in figure 2.4).

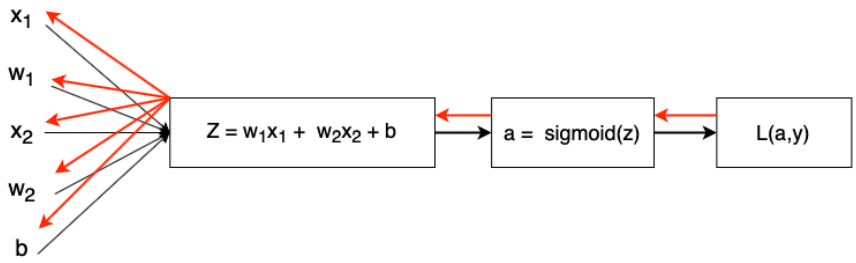

Figure 2.4.: Calculating gradient for one training example

The main equations for determining gradient descent for LR on a single training example involve changing parameters w and b to lower loss. Appendix A.1 provides an example of backward propagation, whereas A.2 shows a computation graph (see picture A.1). Equation 2.8 shows the steps for forward propagation.

$$Z = w_1x_1 + w_2x_2 + b \;\Rightarrow \hat{y} = a = \sigma(z) \Rightarrow L(a, y) \tag{2.8}$$

2.7. Neural Networks

NNs, inspired by the human brain, are ML models with interconnected artificial neurons and the perceptron as their output generator. AI applications benefit from NN's ability capacity to learn from and incoming data [23]. The main components include the input layer, hidden layers, and output layer.

NNs are effective in constructing functions that precisely map from input (x) to output (y), particularly useful in supervised learning environments where input is mapped to output, as illustrated in the figure 2.5.

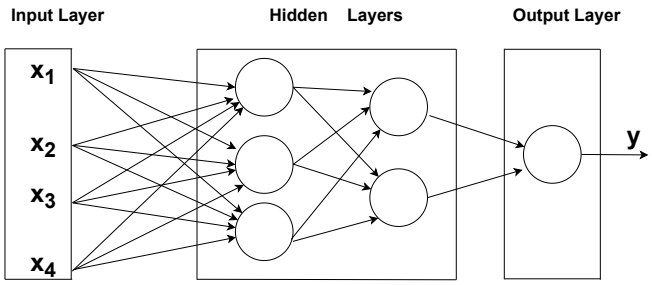

Figure 2.5.: Basic ANNs

2.7.1. Basics of Neural Networks

The basic building block is a perceptron, which takes multiple input values, applies weights, sums them up, and passes the result through an Activation Function (AF) to produce an output. The depth of the ANNs model depends on its layers, which transform data using weights and replicate a linear regression twice. In Figure 2.5, a simple ANNs has one input layer, two hidden layers, and one output layer [24].

2.7.2. How neural networks works

In NNs, layers are used to map inputs to targets, with the weight of the layer being the most important transformation element. The goal is to determine the weight values for the layers that effectively map examples from input to target, but this is not possible when thousands of layers are used. The optimizer uses the loss value as a feedback signal and updates the weights to minimise the LF [25]. A training loop that modifies the values for each run weight and lowers the loss value follows the initial random assignment of weights (see figure 2.6).

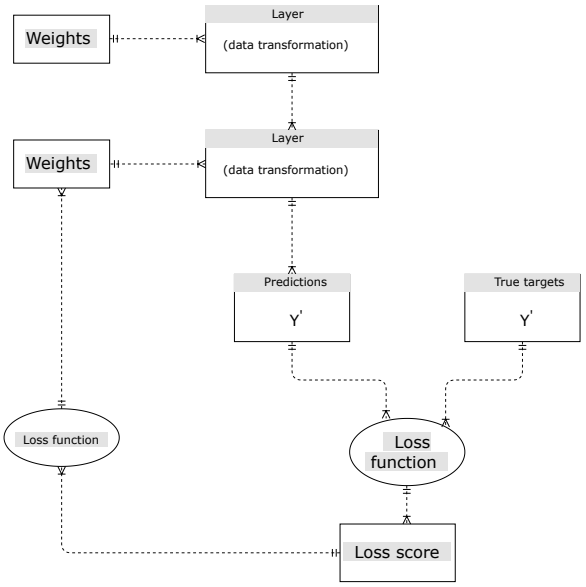

Figure 2.6.: Weight, loss and optimizer

2.7.3. Neural Networks Output and Computing

NNs work similarly to LR, except that the calculations are repeated twice. It uses the feature x and the parameters W and b, calculates $z = W^T x + b$ and $a = \sigma(z)$ twice for each layer, and uses the sigmoid function $a = \sigma(z)$ to calculate the loss L(a,y). The formula for one layer stands for layers one and two (superscript [1] and [2]), and the round superscript x[i] stands for the i-th training example, as shown in figure 2.7.

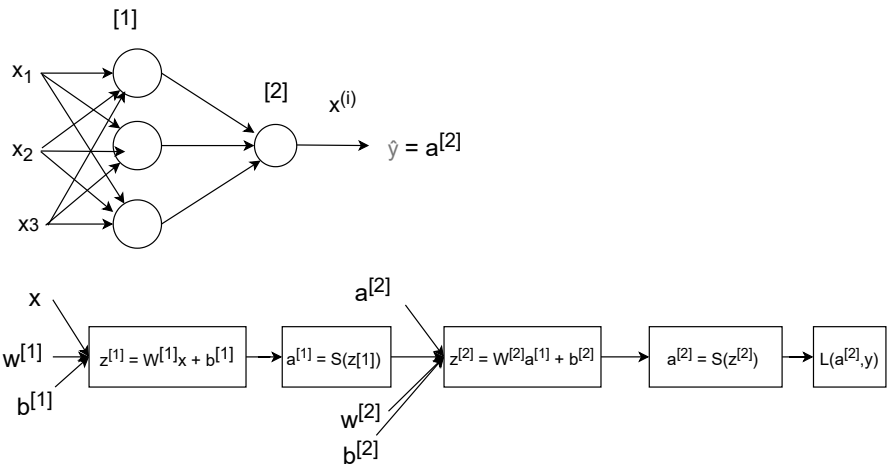

Figure 2.7.: Neural networks overview

In NNs, calculate the linear equations of $z^{[1]}$, $a^{[1]}$, $z^{[2]}$ and $a^{[2]}$ using equations below.

Then calculate the loss $L(a^{[2]}, y)$ [26].

- $z^{[1]} = W^{[1]}x + b^{[1]}$, $a^{[1]} = \sigma(z^{[1]})$
- $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$, $a^{[2]} = \sigma(z^{[2]})$

2.7.4. Data Representation - Tensors

The ML data is represented using a tensor, which is an N-Dimensional (D) data container for generalisations of N-D space matrices, in particular the matrix (2D tensor). Tensors are more than just data containers; they include true linear transformations between tensors, such as cross products and dot products. The number of axes of a tensor reflects its rank. The tensors used in NNs range from 0D to 4D, with the 3D tensor (height, width, and colour) representing the Google images I included in this thesis [27].

2.7.5. Activation Function

AF, or the transfer function, defines neuron output before LF. The non-linearity properties of AF optimize NNs and CNNs performance, enabling complex computations in hidden layers, a crucial aspect of CNNs.

Non-Linear Functions (NLFs) are popular because they are utilized by NNs with a non-linearity feature at the network's final output. Their differentials (derivatives) and monotonic functions are very helpful. According to the derivative (the slope of the changes in the x and y axes with respect to each other), a monotonic function is one whose value does not change over time [28].

2.7.5.1. Sigmoid Function

Figure 2.8 depicts the S-shaped sigmoid function's curve, and equation 2.9 represents its formula.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.9}$$

As a nonlinear function, the sigmoid function ($\sigma(z)$) gives probabilities between 0 and 1. This makes it useful for predicting probabilities in different models, as the AF of an output layer in NN, and for both binary and multi-label classification tasks. Its slope can be found between any two points and is monotonic. The two most commonly used cases are when z is very large positive ($z \to \infty$), approximated to 1 ($z \approx 1$), and when z is very small ($z \to -\infty$), close to zero ($z \approx 0$) [29].

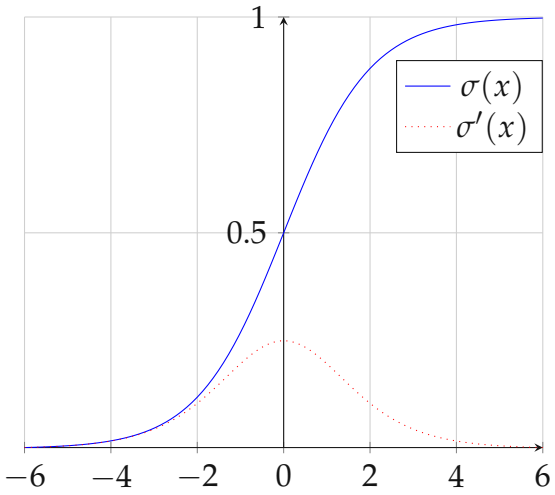

Figure 2.8.: Sigmoid Function

2.7.5.2. Rectified Linear Unit Function

Rectified Linear Unit Function (ReLU) has a slope derivative of one for positive x and zero for negative x, eliminating vanishing gradient problems as noticed by equation 2.10. Equation 2.10 shows that ReLU has a slope derivative of one for positive x and zero for negative x, which eliminates problems with vanishing gradient issues. Its fastest computation without exponentials or divisions gives it a major advantage over sigmoid for training and prediction, as shown in figure 2.9, making it more effective and efficient [30].

$$\text{ReLU} = \max(0, x) \equiv \begin{cases} x_i & \text{for } x_i \geq 0 \\ 0 & \text{for } x_i < 0 \end{cases} \tag{2.10}$$

2.7.5.3. Softmax Activation Function

The softmax AF is used in the output layer to calculate the probability distribution from a vector of real values. The output probabilities range from 0 to 1, and their sum is equal to 1, as expressed in equation 2.11. Since the target class has the highest probability, softmax is commonly used for multi-class classification [31].

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.11}$$

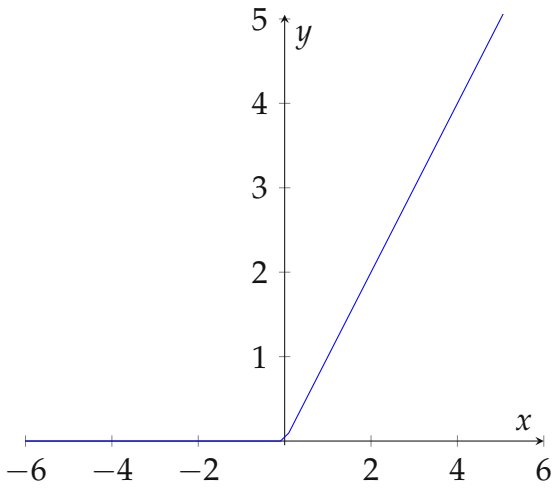

Figure 2.9.: ReLU Function

2.8. Types of Neural Networks

NNs) are used for various tasks, including classification, regression, image processing, natural language processing and computer vision. Common types include CNNs, RNNs, ANNs and HNNs. The choice of neural network architecture depends on the type of data and the requirements of the task.

2.8.1. Artificial Neural Networks

ANNs are indeed designed to mimic the structure and behavior of biological neurons, with a particular focus on the human brain. The organization into layers (input, hidden, and output) and the presence of artificial neurons (interconnected nodes) are key components of them.

2.8.2. Perceptron

A perceptron is a type of NN with one layer and no hidden layer, limiting its ability to approximate linear problems. It has multiple inputs $(x_1, x_2, ...x_n)$, a single output (y), bias input, and weights $(w_0, w_1, ..., w_n)$, as shown in figure 2.10. The perceptron calculates the weighted sum of inputs and applies an AF [32].

2.8.3. Convolutional Neural Networks

Image processing technologies known as CNNs utilize convolutional layers to automatically and adaptively learn spatial hierarchies of features [33]. I've written

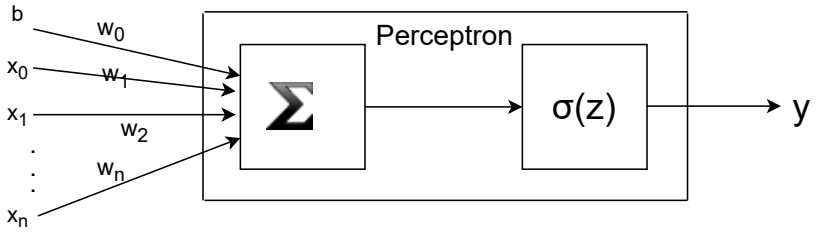

Figure 2.10.: Perceptron

extensively about CNNs in chapter 3.

3. Convolutional Neural Networks Fundamentals

In this chapter, the layers (convolution, pooling, max pooling, ReLU, and fully connected layers), training, and types of CNN are introduced, along with other basic components of CNN. The creation of a functional CNN model relies heavily on these concepts, which are essential for the model's capacity for learning and prediction accuracy, as shown in Figure 3.1.

3.1. From deep neural networks to CNNs

CNNs are a NN type commonly used in image-related tasks such as image classification, semantic segmentation, and object recognition [34].

As mentioned in chapter 2, CNN uses the convolution feature to find patterns around images. This allows the network to learn and extract higher and higher-level models of what is present in an image, starting with its primary pixel data. Using filters and building layers, CNN can identify edges, shapes, and textures in images. Because of this, they are beneficial for tasks like classifying images, as shown in figure 3.1.

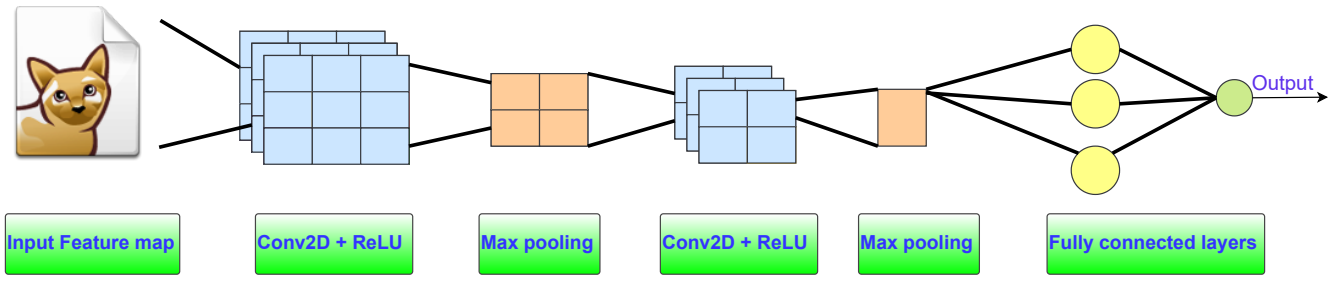

Figure 3.1.: Structure of CNN

3.2. Convolution

After the input layer, the next layer in a CNN is typically the convolution layer. This layer is responsible for identifying features in the input images and calculating matched feature points. The convolution layer filters the images, allowing only relevant features to pass through to the subsequent layers.

The convolution operation makes CNN's ability to extract features from images possible. To do this, convolutional layers are used, which are optimized for finding and extracting patterns from pictures, as shown in figure 3.2 [35].

The convolution operation is used to compute a convolved feature (output feature map) by sliding a convolution filter over the tiles of the input feature map pixel by pixel extracting each tile, and computing the feature.

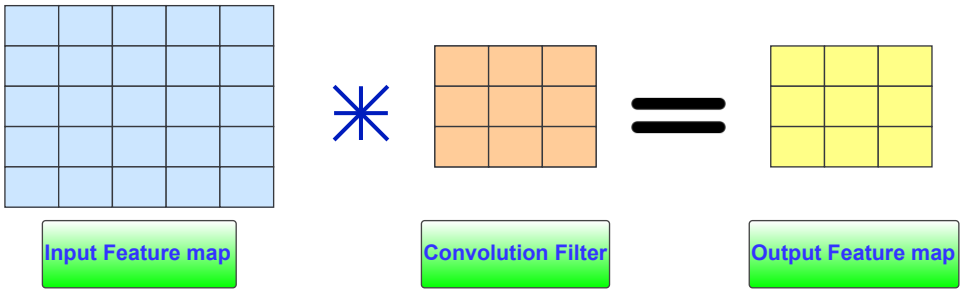

Figure 3.2.: Convolution in CNN

3.3. Pooling

The next layer in a typical CNN is pooling, a downsampling process that reduces the dimensions of the output feature maps while retaining the most relevant features. This thesis uses max pooling as a technique for this purpose.

3.3.1. Max pooling

It is used max pooling to reduce input data dimensionality while keeping fundamental properties. By sliding over the input feature map and removing tiles, max pooling reduces output feature map dimensions while retaining the most essential information. The new feature map receives the maximum tile value. Figure 3.3 shows the filter size of 2x2 pixels and stride 2 (shifting input matrix tile extraction by 2), while appendix A.5 shows the maximum pooling calculation [36].

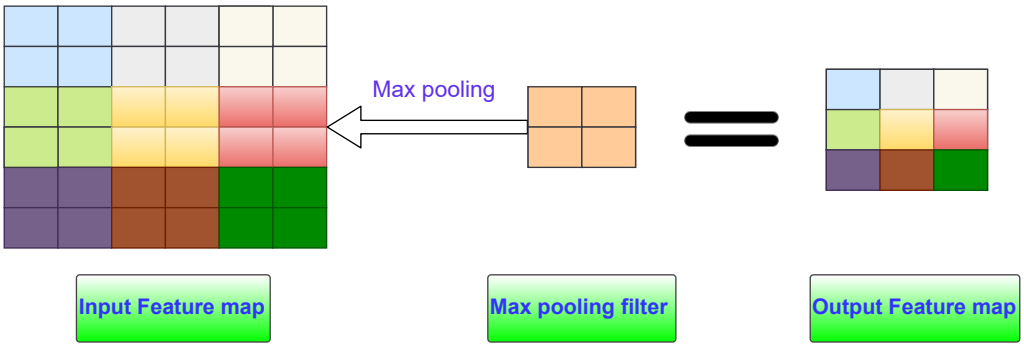

Figure 3.3.: Max pooling

3.4. Fully Connected Layer

A fully connected layer means that every node in the first layer is connected to every node in the second one (also called the dense layer), used to translate filtered images into categories with labels. In CNNs the previous layers generate features for the fully connected layer to classify images.

4. Single vs. Multi-Digit Classification

4.1. Dataset

Datasets are the coal-steam engine behind DL's successful story. The MNIST dataset includes images of handwritten digits in grayscale and consists of a three-dimensional tensor of 8-bit integers and an array of 60000 matrices of 28-by-28 integers.

Figure 4.1 displays a plot of nine randomly selected photos from the dataset using Python code. There are 70,000 28x28-pixel square pictures, 60,000 of which are training data and 10,000 test data. Using this dataset, models were trained and tested for recognising handwritten digits. The models are able to accurately identify handwritten digits with high precision.

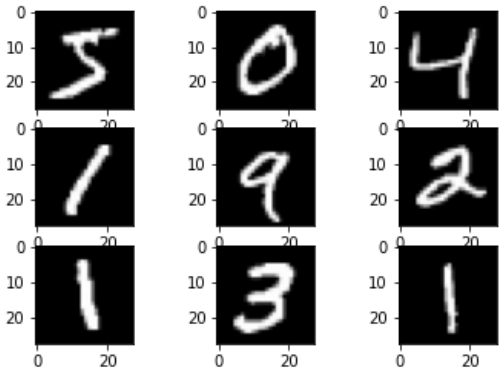

Figure 4.1.: An image plot of a subset of the MNIST dataset's records.

4.2. Training Procedure

This section details the training methodology that was used. Models of one-digit, overlayed images of digits and combinations are used in the network's training process.

The next sections detail the architecture of implementing those procedures for CNNs, where either a single image or a mixture of images are fed into the network for detection.

4.2.1. One Digit Model (M1)

After developing the CNN model for the one-digit model, it was able to recognise a single digit. Using layer manipulation, the accuracy of identifying single items was improved. The network is trained on a single image and overlayed images, and it is also evaluated for its performance on a single image. [37].

4.2.1.1. Architecture

For a one-digit model, a three-layer sequential model of CNN was created, taking a 28x28 pixel image and producing a 10-length vector as output. . In the first layer, convolution is used (Conv2D with 32 filters and a 3x3 convolution filter with a stride of 1), where ReLU is applied as AF and the input shape is 28x28 pixels with one channel. Following that, 2D max pooling was applied using a 2x2 filter and a stride of 2. Next, a fully connected layer with 100 neurons and the same AF as the preceding layer was employed.

In the third layer, a dense layer was used, but unlike the first two layers, 10 neurons and Softmax were used as the AF, as stated in listing 4.1.

```
def one_digit_model(optimizer='adam', loss='categorical_crossentropy'):
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,
        1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
    return model
```

Listing 4.1: Define model with one digit

4.2.1.2. Training model with one digit

For handwritten digit recognition, training a network with one digit is crucial. On one digit, the trained network performed very well; the findings are in the following

chapter 5. The following example of code trains a network with one digit as shown in listing 4.2.

```
def train_one_digit_model(X_train, y_train, X_test, y_test, model):
   history = model.fit(X_train, y_train, epochs=opt.epochs,
      batch_size=opt.batchSize, validation_data=(X_test, y_test),
      verbose=1)
   return history, model
 #Train a model with one digit
   if opt.task == '1':
 # first build a model
      model = common_model(loss=opt.loss)
        if opt.use_common
        else one_digit_model(loss=opt.loss)
 # train the previous model
      history, model = train_one_digit_model(X_train, y_train, X_test,
         y_test, model)
 # Save the model
      model.save('{}_{}_{}.h5'.format(int(time.time()), opt.task,
         opt.loss))
```

Listing 4.2: Training a model with one digit

4.2.2. Two Digits Model (M1)

Another key part of this thesis is training a network to identify two digits. Combining images and accurately detecting two digits was the goal. The model architecture and training are shown below. A minimum of 50% two-digit matching is expected for detection.

The goal was to train the network using a pair of images and check for one-digit matches, two-digit matches, and no matches at all. A very high detection rate was excepted for both one and two digits and an extremely low match rate for no match accuracy [38].

4.2.2.1. Architecture of two digits model

To find two digits, the same architecture as for one-digit models was used, except that the binary cross-entropy loss type was used. Also, the sigmoid function was applied to the last layer, identified as AF, as shown in the listing 4.3.

```
def two_digit_model(optimizer='adam', loss='binary_crossentropy'):
```

```
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,
        1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(10, activation='sigmoid'))
    model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
    return model
```

Listing 4.3: Define model with two digits

4.2.2.2. Training model with two digits

Training a network with two digits in a way to generate a new image by combining two digits. Additionally, the combination shown in listing for training a model with two digits was used in subsequent listing 4.4.

```
def train_two_digit_model(X_train, y_train, X_test, y_test, model):
    X_train_combinations, y_train_combinations =
        generate_combinations(X_train, y_train, n=X_train.shape[0])
    X_test_combinations, y_test_combinations =
        generate_combinations(X_test, y_test, n=X_test.shape[0])

    y_train_combinations_binary = np.zeros((len(y_train_combinations),
        10))
    y_test_combinations_binary = np.zeros((len(y_test_combinations), 10))

    for i, j in enumerate(y_train_combinations):
        for digit in j:
            y_train_combinations_binary[i, digit] = 1

    for i, j in enumerate(y_test_combinations):
        for digit in j:
            y_test_combinations_binary[i, digit] = 1

    history = model.fit(X_train_combinations,
        y_train_combinations_binary, epochs=opt.epochs,
        batch_size=opt.batchSize, validation_data=(X_test_combinations,
        y_test_combinations_binary), verbose=1)
```

```
    return history, model
```

Listing 4.4: Training a model with two digits

4.2.3. Training model with parameters

The NN model has been evaluated using various parameters, including batch size and epochs, as detailed in chapter 5. Training with varying batch sizes and epochs maximised accuracy. The batch size is the amount of data utilised in one epoch to train a neural network. It affects model quality, training time per epoch, and overall training time [39].

A batch size of 64 has been chosen, and it is expected to observe fluctuations in loss and accuracy, as shown in table 5.4, for varying numbers of epochs.

4.2.4. Proposed Concept (M2)

Another model named the combination model (M2), has been applied in this thesis. Within a combination model, a model has been created that produces 55 outputs. Training in the overlapping images generated 55 potential pairs and a unique vector for each pair was received [40].

4.2.4.1. Architecture of combinations model

In this model, as seen in listing 4.5, the same architecture as in the two-digit model was used, with the exception that an extra layer of 100 neurons was added, and softmax was used at the final layer.

```
def combinations_model(mapping, optimizer='adam',
    loss='binary_crossentropy'):
    # generate a model that has 55 outputs
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,
        1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(len(mapping), activation='softmax'))
    model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
    return model
```

Listing 4.5: Defining combinations model

To outperform 55 pairs of combinations, a training model may aim to produce a function with no duplicates (such as [0,1], [0,2], [0,3],... [0.2]) and no repetitions of the same digit (such as [1,1]). The expectation was that no repetitions or occurrences of the same digit would be encountered during the training of this model. After training, the model provides the combinations and list of labels ([[1,0], [2,1],... [3,4], [5,6],...]). The default weights for combining in this generation are 0.5 for picture 2 and 0.5 for picture 2, as shown in listing 4.6.

```
def generate_combinations(X, y, n=10000, weights=[0.5, 0.5]):
   # define size of output
   X_out = np.zeros((n, 28, 28, 1))
   # defined the list of labels
   labels = []
    possibilities = list(range(X.shape[0]))
   for i, c in enumerate(range(n)):
      # get two number randomely
      index_1, index_2 = random.sample(possibilities, 2)
      while np.argmax(y[index_1]) == np.argmax(y[index_2]):
         index_1, index_2 = random.sample(possibilities, 2)
      X_out[i,:,:,:] = weights[0]*X[index_1] + weights[1]*X[index_2]
      labels.append([np.argmax(y[index_1]), np.argmax(y[index_2])])
   return X_out, labels
```

Listing 4.6: Generating combinations model

4.3. Evalution Procedure

In this thesis, multiple test modules have been used, which will be elaborated on in the following items.

- Test a model with one digit: The NN model is tested by using only one digit, as shown in listing 4.7, and the expected result was to check how accurate the model is at detecting one digit.
- Test a model with overlayed images of digits (two digits): The NN model is evaluated also by utilizing mixed images of two digits, as depicted in listing 4.7. The expected outcome was to determine the accuracy of detecting two digits [41].

```
def main():
   X_train, y_train, X_test, y_test =
      load_dataset(which=opt.dataset)
   X_train, X_test = normalize(X_train, X_test)
   model = load_model(opt.model)
#test a model with one digit
   if opt.task == '1':
      evaluate_one_digit_model(X_train, y_train, X_test, y_test,
         model)
#test a model with two digits
   if opt.task == '2':
      evaluate_two_digit_model(X_train, y_train, X_test, y_test,
         model)
#test a model with the new outputs method
   if opt.task == 'combinations_model':
      mapping = generate_mapping(10)
      evaluate_combinations_model(X_train, y_train, X_test, y_test,
         model, mapping)
```

Listing 4.7: Test Combinations /One & two digits

- Test a model with one picture: After the network was trained on the single image using one or two digits, it is tested on a single image, using two types of evaluation, one is the naive evaluation and the other one is threshold evaluation as shown in the chapter 5.
- Test a model with two pictures: After the network was trained on the double images using one or two digits, it has to come to testing this network on two images, using two types of evaluation, the naive evaluation and threshold evaluation [42].
- Test a model with all possible combinations: The NN model is tested also with pairs of combinations (55 outputs are generated).

5. Experiments and Results

In this chapter, the experimental setting used for this thesis has been shown. The results of the training and testing procedures are briefly explained, along with the accuracy percentages for single-digit, multi-digit and combination model classifications. Also, results of other parameters, such as batch size, epochs and thresholds, are presented. It is shown that these parameters have a significant impact on the accuracy of models.

5.1. Experimental setup

Several pieces of hardware and software were used to complete this thesis. A MacBook(OS:MacOS Big Sur, Processor:i7-10750H CPU @2.60GHz (12 CPUs)) was used. Through a remote connection to the institute of telecommunications at Vienna University of Technology (TUW) was able to run simulations with a large number of features and combinations. In addition, Python 3.7, Keras, and the TensorFlow framework have been used.

The Graphics Processing Unit (GPU) (NVIDIA GeForce RTX 3060 (12GB)) was used to speed up calculations and analyses. The GPU has substantially reduced the processing time of the data and enabled more complicated computations to be conducted. Other packages used include NumPy and Pandas for data analysis and manipulation, along with Matplotlib for visual representation.

5.1.1. Evaluation Environment

The effectiveness of model training and testing in this evaluation environment has been demonstrated. First, models were evaluated during training, including single-digit using M1. By using CNN and layer modifications we got over 90% detection accuracy. Playing with epochs, batch sizes, and predictions to find single images, blended images (overlaying images of two digits using this model (M1)) are performed. A new suggested approach was again trained using model M2 for mixed images of two digits (blended images). Additionally, models were trained with various LFs. Several parameter combinations were tried to improve the model's performance.

A second approach involves the evaluation of single-digit images. Also, when digits are combined (such as combining the digits "1" and "2" in one image), then a new image is generated through the combination of two digits (representing multi-digit classification) using models M1 and M2.

M1 was tried for images with single-digit analysis. Without same digit detection in a new image, the accuracy of one match, two matches and zero matches were identified. The same model (M1) was applied to test the accuracy detection when overlayed images of two digits and found the accuracy of one match, two matches and zero matches were identified. This same procedure of overlaying images of two digits was repeated 10 and 100 times to check how accuracy changed. Nex, multi-digit classification using the proposed approach (M2) was explored and compared to random guessing for identifying two digits in a blended image. In the end, models are tested with various values. The curiosity lies in seeing if the proposed approach detects two digits in an overlayed image better than random guessing.

5.2. Results of Training Procedure

In the next parts, the outcomes of single-digit and multi-digit classification model training evaluation, epochs and batch size are presented.

5.2.1. Single Digit Performance with M1

Refer to table 5.1 for single-digit training results. As displayed, single-digit detection accuracy is 97.8%, and loss is 0.0033. Results show that the basic features of CNNs are effective in identifying individual digits

Accuracy	Val_accuracy	Loss	Val_loss
0.9510	0.9780	0.0075	0.0033

Table 5.1.: Single-digit with an initial model

5.2.2. Multi-Digit Classification Performance with M1

The same model (M1) was trained with two digits, as illustrated in table 5.2. Training the original model M1 yields a 55.24% accuracy in identifying digits in overlayed images.

Accuracy	Val_accuracy	Loss	Val_loss
0.5524	0.5138	0.0429	0.0263

Table 5.2.: Multi-digit with an initial model

5.2.3. Multi-Digit Classification Performance with M2

Table 5.3 shows a mixed-image evaluation of the new concept, or upgraded model (M2), called the combinations model. Two-digit detection accuracy is 71.30%, which is better than the baseline model. Loss is also substantially decreased. These findings show that M2 is more accurate than glsm1a.

Accuracy	Val_accuracy	Loss	Val_loss
0.7130	0.8409	0.0269	0.0156

Table 5.3.: Multi-digit with an improved model

5.2.4. Parameter Performance with M1

The first model (M1) was trained with multiple parameters for the highest level of precision. The accuracy and loss difference while adjusting epochs and batch sizes was elaborated, especially for 100 epochs and 64 batch sizes. Table 5.4 shows 99.9% accuracy and 0.002 loss in recognising one digit. Modifying those settings to identify a single image was a good approach.

Epochs	Batch size	Accuracy	val_accuracy	Loss
100	64	0.9990	0.9892	0.0020

Table 5.4.: Adjustable single-digit parameters (M1)

As numerous predictions and epochs were tested, it was found that prediction 4 of epoch 6 gave the best outcomes, as shown in table 5.5. For both predictions and epochs, this led to the best accuracy, at 99.16% and 0.0271 loss, respectively. In terms of accuracy and loss, prediction 4 of epoch 6 performed best. These data show that the specified parameters provide good prediction model performance.

6,000 epochs were tested using that prediction and epoch. Table 5.6 shows an accuracy of 99.12% accuracy and 0.028 loss. This suggests that adding epochs beyond 6,000 did not increase model performance. In this dataset, prediction 4 of epoch 6

Epoch	loss	accuracy
1/6	0.1434	0.9559
2/6	0.0467	0.9859
3/6	0.0350	0.9891
4/6	0.0271	0.9916
5/6	0.0218	0.9929
6/6	0.0174	0.9946

Table 5.5.: Outcomes for particular predictions and epochs

Epoch	Test loss	Test accuracy
6000/6000	0.0280	0.9912

Table 5.6.: Epoch-based accuracy loss

resulted in the highest accuracy and lowest loss among all other combinations.

5.2.5. Loss Function Computation

The model was trained using three loss functions in this thesis.

The loss functions trained were MSE as given in table 5.7, then binary cross-entropy loss, and finally categorical cross-entropy. Accuracy and loss results were then compared. Table 5.7 shows that categorical cross-entropy loss has the highest accuracy and MSE the lowest loss. This shows that MSE loss may work better in certain instances. However, MSE accuracy may not always be ideal. Thus, loss function selection can significantly affect model performance.

Type_Loss	Accuracy	Val_accuracy	Loss	Val_loss
MSE	0.9489	0.9746	0.0079	0.0038
Binary	0.9514	0.9803	0.0277	0.0114
Categorical	0.9540	0.9770	0.1559	0.0681

Table 5.7.: Loss function accuracy and loss

5.3. Outcomes of Testing Procedure

This thesis focuses on CNN model testing to ensure better classification performance. This section presents the results of different methods and models.

5.3.1. Single Digit Classification Performance (M1)

The outcomes of testing the M1 with a single digit, a key study goal and accuracy for recognising one digit are shown in table 5.8. Testing with one digit in the image outputs high accuracy, as seen in table 5.8 at 94.94%.

loss	accuracy	val_loss	val_accuracy
0.0077	0.9494	0.0036	0.9745

Table 5.8.: Single-digit outcomes for the initial model M1

The averaged outcomes for detecting one match, two matches, and no matches at all after running 500,000 times are presented in 5.1 below.

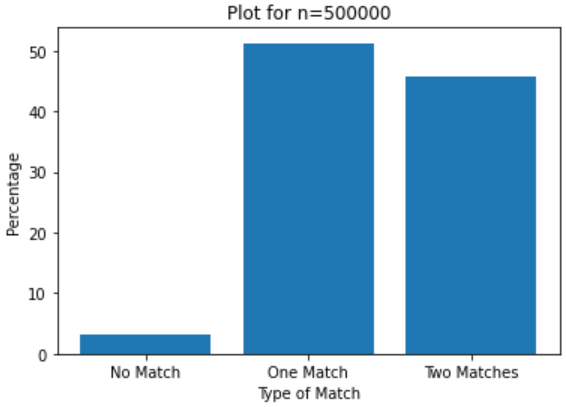

Figure 5.1.: Single-digit outcomes for n-runs

Avoiding the same digits during the testing process. By eliminating the same pictures, it was able to make more accurate results. Table 5.9 shows that the prediction for two matches has gone up by 50%, the prediction for zero matches has gone down by 50% (which means we have fewer losses, which is better accuracy), and the prediction for one match has gone down by about 22%. This shows that my predictions are better accurate now than they were before.

	Zero match	One match	Two matches
Avoid	0.0307	0.5122	0.4569
Not Avoid	0.0608	0,6334	0.3056

Table 5.9.: Avoid the same digits

5.4. Multi-Digit Classification Performance with Initial Model (M1)

Testing M1 with two digits was another important part of this thesis. By experimenting with M1 with two pictures, we found out how well it could find a single digit, two digits, or no match. According to the results, the M1 model was good at finding two digits (58.01%), and it had a low rate of zero matches (1.34%). Table 5.10 shows the average accuracy of finding two digits.

Zero match	One match	Two matches
0.0134	0.4065	0.5801

Table 5.10.: Mutli-digit's outcomes with M1

In the same way, M1 was examined 10 times to see how much the average accuracy finding would change. The accuracy testing results for finding one match, two matches, and no match are shown in the table 5.11. They are almost the same as the average accuracy found by running the multi-digit M1 recognition methods just once.

Zero match	One match	Two matches
0.0153 ± 0.182	0.3991 ± 0.336	0.5854 ± 0.379

Table 5.11.: Running 10-times multi-digit M1.

The M1 was run with multi-digits 100 times to get the average accuracy for each type of match. It turned out that there wasn't a big difference between this evaluation and the previous multi-digit M1, run one and ten times, as shown in table 5.12.

Zero match	One match	Two matches
0.0155 ± 0.112	0.4007 ± 0.49	0.5836 ± 0.496

Table 5.12.: Running 100-times multi-digit M1

5.5. Multi-Digit Classification Performance with Improved Model (M2)

The probability of detecting two digits with the new model M2, which contains combinations, was investigated. After running the M2 model, the average accuracy of two-digit detection was 84.92%. This represents the best two-picture detection outcome till now. Random guessing accuracy of recognising multi-digits was at 2.650%. M2 multi-digit detection surpasses 32 times the average accuracy of random guessing. This demonstrates the superiority of the M2 model in recognising overlayed images with multi-digit.

5.6. Threshold Evaluation with Initial Model (M1)

Outcomes of testing M1 with different threshold values are shown in table 5.13. When using a threshold, the average accuracy for identifying digits is high for no match and one match but lower for two. This shows that the threshold value significantly affects the model's accuracy.

n	Zero match	One match	Two matches
0.10	26.82	39.17	34.01
0.13	16.69	50.03	33.28
0.15	12.26	55.51	32.23
0.175	8.62	60.8	30.58
0.20	5.62	66.76	27.62
0.25	5.39	74.29	20.32
0.75	64.44	35.56	0.0

Table 5.13.: Different threshold's percentage values

Figure 5.2 shows that the highest effective results occur at the 25% threshold and

the least effective occur at the 75% threshold when no two matches are found. This shows a lower threshold value improves model performance.

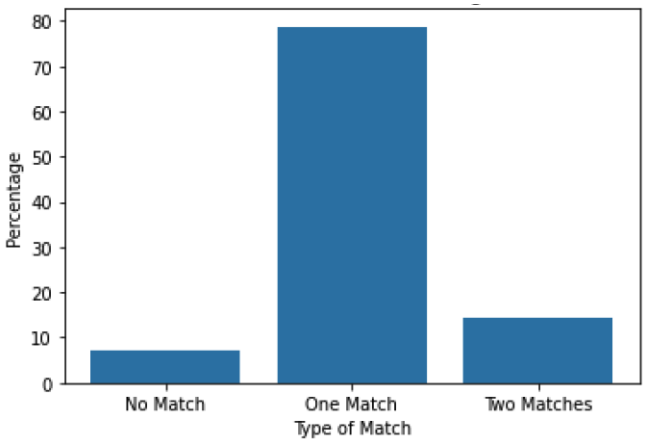

Figure 5.2.: Results for threshold value 0.25

6. Conclusion

6.1. Conclusion

An improved model (M2) during training shows better accuracy than the initial model (M1) for overlayed images of multi-digit classification performance. Depending on the task, the loss function can be chosen as it will work best for particular results.

The prediction for zero matches has gone down by 50% (indicating fewer losses, which is good accuracy) and the prediction for two matches has gone up by 50%, which means we have better accuracy in recognising two digits in overlayed images and fewer losses by avoiding the same pictures.

Tables 5.10 and 5.11 show that the accuracy difference between running ten times and running one time is almost the same. We got nearly the same result for no match, one match, and two matches. These results proved that the software is reliable and accurate in its predictions. Also, by running M1 100 times, we got almost the same averaged accuracy for matches.

Results with M2 model for detecting overlayed images of two-digits show that they are 32 times better than those achieved through random guessing.

6.2. Future Work

In this thesis, several cases of CNNs applications are conducted in MNIST database. If time permits, future work will include many other datasets and classifications, such as.

1. Compare the same models with CIFAR-10 dataset.
2. Compare the same models with CIFAR-100 dataset.
3. Compare the same models with ImageNet dataset.
4. Compare the same models with EMNIST (Extended MNIST) dataset.
5. Use Single-digit classification performance with improved model M2.
6. Run the multi-digit with M2 for 10 and 100 times, then compare those results with the multi-digit with M1.

A. General Addenda

In this chapter, additional information and examples from the above chapters of this thesis are .

A.1. Backward Propagation Example

Using 2.6.4, figure 2.4 and equation 2.4, this section demonstrates the backward propagation step calculation. This example shows how gradient descent updates neural network weights during training. Calculate backwards to compute derivatives, first computing the derivative of LF with respect to a.

- Calculate LF's derivative of a as $\frac{dL(a,y)}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$.
- The derivative of a with respect to Z is $\frac{da}{dZ} = \frac{dL(a,y)}{da}\frac{da}{d\sigma(z)} = (-\frac{y}{a} + \frac{1-y}{1-a})\frac{[a(1-a)]^2}{a-y} = a(1-a)$.
- LF derivative with respect to Z is calculated as $dz = \frac{dL}{dZ} = \frac{L(a,y)}{dz} = \frac{dL}{da}\frac{da}{dz} = (-\frac{y}{a} + \frac{1-y}{1-a})a(1-a) = a - y$

Calculate the derivatives of w_1, w_2 & b, using 2.7 to determine w and b changes.

- Derivative of b is the same as dz $\rightarrow db = dz$
- Derivative of w_1 as $dw_1 = \frac{dL}{dw_1} = x_1 dz$
- Derivative of w_2 as $dw_2 = \frac{dL}{dw_2} = x_2 dz$

Update equation 2.7. After computing the derivatives of w_1, w_2 & b, the updated equation will incorporate derivative values.

- $b \leftarrow b - \alpha db$
- $w_1 \leftarrow w_1 - \alpha dw_1$
- $w_2 \leftarrow w_2 - \alpha dw_2$

A.2. Example of Computation Graph

In this section, a computation graph is shown using formulas 2.4 and 2.6. In figure A.1, the red arrow represents the backward propagation step from right to left, and the corresponding derivatives are calculated in the formulas below.

- Calculating the derivative of CF over v ($\frac{dJ}{dv}$) for $u = 6,\ v = 11,\ \&\ J = 3v = 33$. The derivative of final output variable over the derivative of v is $\frac{dJ}{dv} = 3$ when v-value is shifted from $v = 11 \rightarrow v = 11.001$.
- Calculating $\frac{dJ}{da} = \frac{dJ}{dv}\frac{dv}{da}$, nudging a with 0.001, $a = 5 \rightarrow 5.001$, $v = 11 \rightarrow 11.001$, $\&\ J = 33 \rightarrow 33.003$, and J's derivative over a's derivative is $\frac{dJ}{da} = \frac{dJ}{dv}\frac{dv}{da} = 3$. Nudging a affects v and J (a→v→J), so the amount nudging an is the product of how much v changes and how much CF changes.
- Using the nudging method (u = 0.01), calculate $\frac{dJ}{du}$ for $u = 6u = 6.001$, $v = 11.001$, $\&\ J = 3J = 33.003$. Nudging u with 001, the impact CF is $\frac{dJ}{du} = \frac{dJ}{dv}\frac{dv}{da} = 3$.
- Determine $\frac{dJ}{db}$ by increasing b-value: $b = 3 \rightarrow 3.001$, $u = 6 \rightarrow 6.002$, $v = 11 \rightarrow 11.002$, $\&\ J = 33 \rightarrow 33.006$. CF's impact on b-value nudging with 001 is $\frac{dJ}{db} = \frac{dJ}{du}\frac{du}{db} = 3 \cdot 2 = 6$.
- Calculate $\frac{dJ}{dc}$ by increasing c-value: $c = 2 \rightarrow 2.001$, $u = 6 \rightarrow 6,003$, $v = 11 \rightarrow 11.00$, $\&\ J = 33 \rightarrow 33.009$. The impact of CF on c-value nudging by 001 is $\frac{dJ}{dc} = \frac{dJ}{du}\frac{du}{dc} = 3 \cdot 3 = 9$. Therefore, CF changed 9.

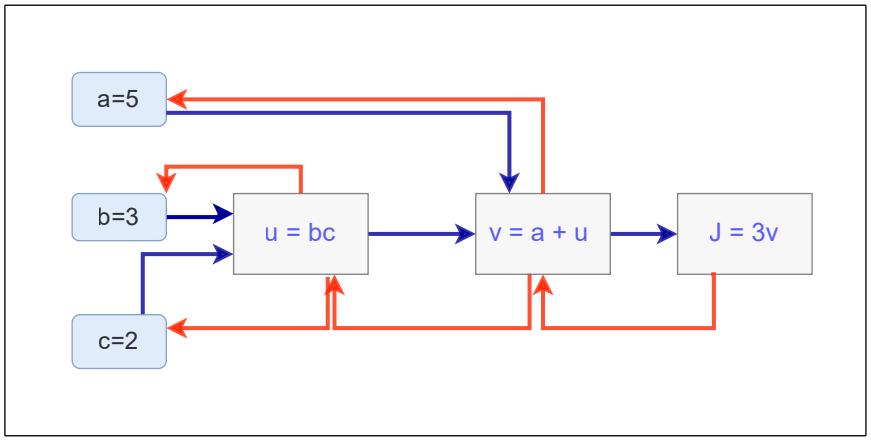

Figure A.1.: Example of computation graph.

A.3. Linear Activation Function

Also known as identity AF. The output neuron is linearly identical to the input. Figure A.2 shows that the neural network output is linear, regardless of data complexity, using the equation $y = f(x) = x$. Thus, neural network output is proportional to the input.

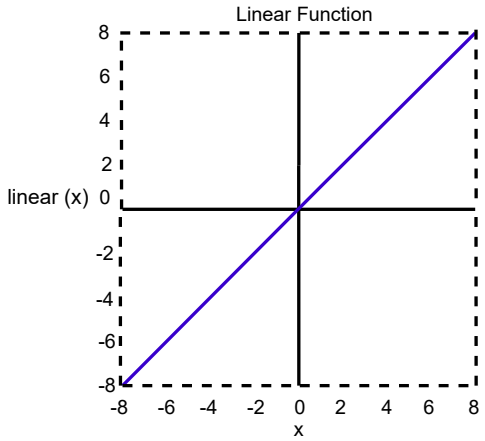

Figure A.2.: Linear Activation Function

A.3.1. Why Non-linear Activation Functions

When using linear AF, NNs only outputs a linear function of the input, regardless of hidden layer count or parameter changes. See figure A.3. The equations for NNs are $z^{[1]} = W^{[1]}x + b^{[1]}$ and $a^{[1]} = z^{[1]}$. The output layer equations are $z^{[2]} = W^{[2]}x + b^{[2]}$ and $a^{[2]} = z^{[2]} = W^{[2]}a^{[2]} + b^{[1]}$. For the equation $a^{[2]} = W^{[']}x + b^{[']}$. Output equals input for the network, so the network is linear.

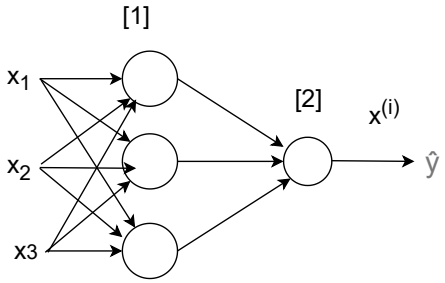

Figure A.3.: Why we need Non-linearity

A.4. Example of convolution in CNN

The convolution filter extracts tiles from the input feature map (an image) by iteratively sliding over each tile, one pixel at a time (assuming no other feature, such as padding or stride, is added). The CNNs first processes the features of the input image by multiplying the filter matrix one element at a time for each filter-tile pair. This makes the convolved feature output (output feature map). This is performed on a 5x5 input matrix to generate a 3x3 feature map for each filter-tile pair. This feature map is then put through additional analysis and processing. To create a convolved feature output, this is done for every filter-tile pair in the input matrix, as shown in figure A.4.

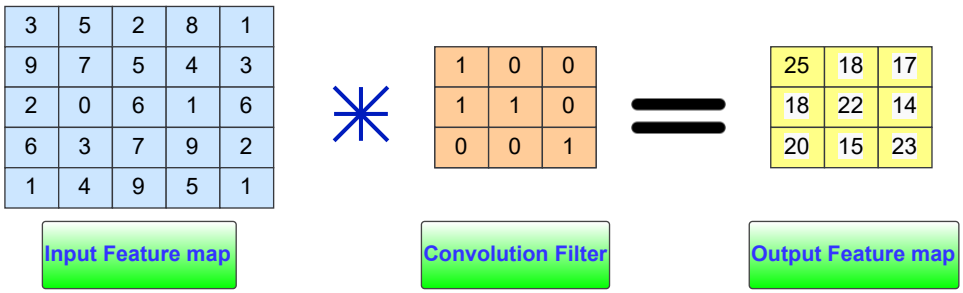

Figure A.4.: Convolution in CNNs

In order to extract crucial features, such as textures, edges, and forms, the CNNs learns autonomously the ideal value for the convolution filter matrices. Adding more filters during training allows for a greater number of features to be extracted from the input but also extends the total training time. So, it's important to acquire as many functions as you can. Only a few filters are needed to extract features for use in categorization images. The number of filters and training time must be balanced for accurate and effective image classification.

A.5. Example of max pooling

In this thesis is used max pooling. CNNs use max pooling, which includes choosing the highest value from each tile. This method reduces input data dimensionality while keeping key properties. Max pooling creates translation-invariant representations, making it suitable for image classification. Max pooling captures the most essential characteristics by sliding the filter over the picture, like convolution. For each tile, the new output feature map uses just the maximum value. These procedures have a stride and a maximum pooling filter size of 2x2 pixels. Figure A.5 shows that a stride of 2 places the following tile 2 pixels forward. The output is a 2x2 feature map from a 4x4 input feature map with a 2x2 max pooling filter.

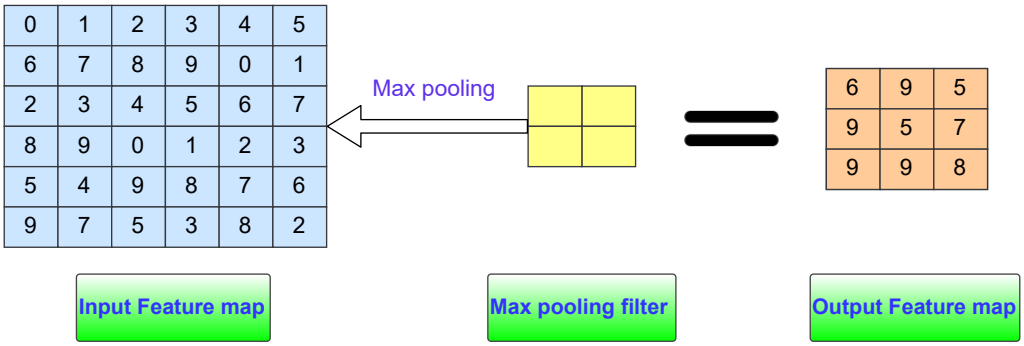

Figure A.5.: Max pooling example in CNNs

List of Figures

List of Tables

Listings

Acronyms

AF Activation Function. 11, 13–15, 21, 22, 38, 39

AI Artificial Intelligence. 1, 3–6, 11, 41

ANN Artificial Neural Network. 6, 11, 15, 41

CE Cross-Entropy. 8, 9

CF Cost Function. 9, 10, 37

CNN Convolutional Neural Network. iv, 1, 2, 5, 13, 15–19, 21, 27, 28, 31, 35, 39–41

D Dimensional. 13, 21

DL Deep Learning. 1, 3, 6, 7, 20, 41

GPU Graphics Processing Unit. 27

HNN Hybrid Neural Network. 5, 15

KNN Konvolutionäre Neuronale Netzwerke. v

LF Loss Function. 8, 11, 13, 27, 36

LR Logistic Regression. 7, 10, 12

M1 Initial Model. iv, v, 1, 2, 27–29, 31–33, 35

M2 Improved Model. iv, v, 1, 2, 24, 27–29, 33, 35

ML Machine Learning. 1, 3–6, 11, 13, 41

MNIST Modified National Institute of Standards and Technology. 1, 20, 35, 41

MSE Mean Square Error. 8, 30

NLF Non-Linear Function. 13

NN Neural Network. iv, v, 1–3, 5, 10–13, 15, 17, 24–26, 39, 41

ReLU Rectified Linear Unit Function. 14, 21

RL Reinforcement Learning. 6

RMSE Root Mean Square Error. 8

RNN Recurrent Neural Network. 5, 15

SD Structured Data. 6

SL Supervised Learning. 5–7

SSL Semi-Supervised Learning. 6

TUW Vienna University of Technology. 27

USD Unstructured Data. 6

USL Unsupervised Learning. 6

Bibliography

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[3] E. B. Hunt. *Artificial intelligence*. Academic Press, 2014.

[4] A. Pannu. "Artificial Intelligence and its Application in Different Areas". In: *Artificial Intelligence* 4.10 (2015).

[5] J. Alzubi, A. Nayyar, and A. Kumar. "Machine learning from theory to algorithms: an overview". In: *Journal of physics: conference series*. Vol. 1142. IOP Publishing. 2018, p. 012012.

[6] A. L. Samuel. "Some studies in machine learning using the game of checkers". In: *IBM Journal of Research and Development* 44.1.2 (2000), pp. 206–226. DOI: 10.1147/rd.441.0206.

[7] M. Mohammed, M. B. Khan, and E. B. M. Bashier. *Machine learning: algorithms and applications*. Crc Press, 2016.

[8] N. Sandhya and K. Charanjeet. "A review on machine learning techniques". In: *International Journal on Recent and Innovation Trends in Computing and Communication* 4.3 (2016), pp. 451–458.

[9] R. E. Baker, J.-M. Pena, J. Jayamohan, and A. Jérusalem. "Mechanistic models versus machine learning, a fight worth fighting for the biological community?" In: *Biology letters* 14.5 (2018), p. 20170660.

[10] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell. "Machine learning: A historical and methodological analysis". In: *AI Magazine* 4.3 (1983), pp. 69–69.

[11] Z.-H. Zhou. *Machine learning*. Springer Nature, 2021.

[12] L. R. Ramiro Vargas Amir Mosavi. *Deep Learning: A Review, Advances in Intelligent Systems and Computing*. 2017.

[13] F. Chollet. *Deep Learning with Python*. Manning Publications Co., 2018, pp. 1–240.

[14] Q. Dong, X. Zhu, and S. Gong. "Single-label multi-class image classification by deep logistic regression". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 3486–3493.

[15] W. Cheng and E. Hüllermeier. "Combining instance-based learning and logistic regression for multilabel classification". In: *Machine Learning* 76 (2009), pp. 211–225.

[16] R. Yu, Y. Wang, Z. Zou, and L. Wang. "Convolutional neural networks with refined loss functions for the real-time crash risk analysis". In: *Transportation research part C: emerging technologies* 119 (2020), p. 102740.

[17] A. Noor, Y. Zhao, R. Khan, L. Wu, and F. Y. Abdalla. "Median filters combined with denoising convolutional neural network for Gaussian and impulse noises". In: *Multimedia Tools and Applications* 79 (2020), pp. 18553–18568.

[18] U. Ruby and V. Yendapalli. "Binary cross entropy with deep learning technique for image classification". In: *Int. J. Adv. Trends Comput. Sci. Eng* 9.10 (2020).

[19] H. Zhao, O. Gallo, I. Frosio, and J. Kautz. "Loss functions for neural networks for image processing". In: *arXiv preprint arXiv:1511.08861* (2015).

[20] P. Murugan. "Implementation of deep convolutional neural network in multi-class categorical image classification". In: *arXiv preprint arXiv:1801.01397* (2018).

[21] V. Nasteski. "An overview of the supervised machine learning methods". In: *Horizons. b* 4 (2017), pp. 51–62.

[22] S. Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[23] C. C. Aggarwal et al. "Neural networks and deep learning". In: *Springer* 10.978 (2018), p. 3.

[24] B. Yegnanarayana. *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

[25] A. Nandy, M. Biswas, A. Nandy, and M. Biswas. "Neural Network Basics". In: *Neural Networks in Unity: C# Programming for Windows 10* (2018), pp. 1–26.

[26] I. A. Basheer and M. Hajmeer. "Artificial neural networks: fundamentals, computing, design, and application". In: *Journal of microbiological methods* 43.1 (2000), pp. 3–31.

[27] R. A. Cohen, H. Choi, and I. V. Bajić. "Lightweight compression of neural network feature tensors for collaborative intelligence". In: *2020 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE. 2020, pp. 1–6.

[28] H.-x. WANG, J.-q. ZHOU, C.-h. GU, and H. LIN. "Design of activation function in CNN for image classification". In: *Journal of Zhejiang University (Engineering Science)* 53.7 (2019), pp. 1363–1373.

[29] B. Ding, H. Qian, and J. Zhou. "Activation functions and their characteristics in deep neural networks". In: *2018 Chinese control and decision conference (CCDC)*. IEEE. 2018, pp. 1836–1841.

[30] G. Lin and W. Shen. "Research on convolutional neural network based on improved Relu piecewise activation function". In: *Procedia computer science* 131 (2018), pp. 977–984.

[31] M. Jogin, M. Madhulika, G. Divya, R. Meghana, S. Apoorva, et al. "Feature extraction using convolution neural networks (CNN) and deep learning". In: *2018 3rd IEEE international conference on recent trends in electronics, information & communication technology (RTEICT)*. IEEE. 2018, pp. 2319–2323.

[32] M. L. Minsky and S. A. Papert. *Perceptrons: expanded edition*. 1988.

[33] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou. "A survey of convolutional neural networks: analysis, applications, and prospects". In: *IEEE transactions on neural networks and learning systems* (2021).

[34] S. Albawi, T. A. Mohammed, and S. Al-Zawi. "Understanding of a convolutional neural network". In: *2017 international conference on engineering and technology (ICET)*. Ieee. 2017, pp. 1–6.

[35] S. Sharma, S. Sharma, and A. Athaiya. "Activation functions in neural networks". In: *Towards Data Sci* 6.12 (2017), pp. 310–316.

[36] V. Christlein, L. Spranger, M. Seuret, A. Nicolaou, P. Král, and A. Maier. "Deep generalized max pooling". In: *2019 International conference on document analysis and recognition (ICDAR)*. IEEE. 2019, pp. 1090–1096.

[37] D. M. Pelt and J. A. Sethian. "A mixed-scale dense convolutional neural network for image analysis". In: *Proceedings of the National Academy of Sciences* 115.2 (2018), pp. 254–259.

[38] N. Sharma, V. Jain, and A. Mishra. "An analysis of convolutional neural networks for image classification". In: *Procedia computer science* 132 (2018), pp. 377–384.

[39] Y. Sun, B. Xue, M. Zhang, and G. G. Yen. "Evolving deep convolutional neural networks for image classification". In: *IEEE Transactions on Evolutionary Computation* 24.2 (2019), pp. 394–407.

[40] A. A. M. Al-Saffar, H. Tao, and M. A. Talab. "Review of deep convolution neural network in image classification". In: *2017 International conference on radar, antenna, microwave, electronics, and telecommunications (ICRAMET)*. IEEE. 2017, pp. 26–31.

[41] B. B. Traore, B. Kamsu-Foguem, and F. Tangara. "Deep convolution neural network for image recognition". In: *Ecological informatics* 48 (2018), pp. 257–268.

[42] W. Wang, Y. Yang, X. Wang, W. Wang, and J. Li. "Development of convolutional neural network and its application in image classification: a survey". In: *Optical Engineering* 58.4 (2019), pp. 040901–040901.