UNIVERSITY M'HAMED BOUGARA, BOUMERDES
INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERING

BACHELOR THESIS

# Artificial neural networks design exploration toward efficient image classification

*Authors:*
Melisa KHACEF
Dihia LACENE

*Supervisor:*
Dr. Ammar ABDELKARIM

*A thesis submitted in fulfillment of the requirements for the degree of bachelor degree*

*in the electrical and electronic engineering*

March 1, 2025

UNIVERSITY M'HAMED BOUGARA, BOUMERDES INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERING

# *Abstract*

Electronic

**Artificial neural networks design exploration toward efficient image classification**

by Melisa KHACEF
Dihia LACENE

This bachelor thesis explores the impact of two different neural network topologies, namely Fully-Connected (FCNN) and convolutional (CNN) architectures, on image classification. However, most applications such as smart devices or autonomous vehicles require an embedded implementation of neural networks.
The study focuses on evaluating the effect of hyperparameters, specifically the number of hidden layers, number of neurons, and activation functions (such as ReLU, Tanh, and sigmoid), as well as the impact of using softmax on the output layers. The evaluation utilizes two widely-used open source datasets, MNIST and CIFAR10.

Incorporating evidence from extensive simulations, the study varies the number of layers, and number of hidden layers, and explores different combinations of activation functions with and without softmax. Additionally, each configuration is tested with three different random seeds to ensure robustness. The results consistently demonstrate that CNN outperforms FCNN in terms of training accuracy for both datasets. Furthermore, the simulations emphasize the significance of architectural choices and activation function combinations in achieving optimal performance.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **CNN** | **C**onvolutional **N**eural **N**etwork |
| **MLP** | Multi Layer Perceptron |
| **FC** | Fully Connected |
| **AI** | Aritificial Intelligence |
| **FCNN** | Fully Connected Neural Network |
| **ANN** | Artificial Neural Network |

# Chapter 1

# Context and motivation

## 1.1 Introduction

Artificial neural networks have become a popular tool for solving complex problems in various domains. It has gained significant attention due to its popularity and widespread use in the computer vision community. The utilization of image classification in our daily lives has become increasingly prevalent, permeating various domains and transforming the way we interact with technology. From social media and healthcare to transportation and e-commerce, AI-powered image classification is enhancing convenience, efficiency, and safety, offering us a glimpse into the potential of AI-driven advancements in the future.(Gandomi and Haider, 2015; Russel, Norvig, et al., 2013)

## 1.2 AI in our daily life

Artificial intelligence has manifested as a central element of modern life, owing to its adaptability and versatility. Particularly, embedded artificial intelligence has attracted considerable attention, such as in image recognition. According to (De Fauw et al., 2018) and (Rajpurkar et al., 2017) Recent research confirms that embedded systems enhance the performance, functionality, and reliability of AI systems. Recently, several researchers have reported the use of AI-based tools in solving image classification problems in healthcare, based on training with X-ray images, CT scans, histopathology images, etc. Deep learning is an extremely powerful tool for learning complex, cognitive problems. This Bsc thesis focuses on the implementation of neural networks in image recognition.

### 1.2.1 AI in image classifiaction

Image classification, powered by Artificial Intelligence (AI) and machine learning algorithms, has found extensive utilization in our daily lives, transforming various aspects of how we interact with technology and the world around us. The ability of AI systems to accurately analyze and categorize images has opened up new possibilities for enhancing convenience, efficiency, and safety in numerous domains. From social media platforms to healthcare, transportation to e-commerce, the applications of image classification have become pervasive. By leveraging vast amounts of data and sophisticated algorithms, AI systems can classify images into specific categories, enabling a wide range of practical use cases. In social media, as shown in (Alam et al., 2020) image classification algorithms play a crucial role in content moderation, automatically identifying and flagging inappropriate or sensitive content, ensuring a safer online environment for users. They also power image recognition features that automatically tag and categorize photos, making it easier for users to search and organize their visual content. In the healthcare sector, providing to (Esteva et al., 2017) image classification has revolutionized medical imaging, aiding in diagnosing and treating various conditions. AI models can analyze medical images such as X-rays, MRI scans, and histopathological slides, assisting medical professionals in detecting abnormalities, identifying diseases, and providing more accurate and timely diagnoses. Also from Lin et al., 2017In the healthcare domain, edge AI has been used to improve patient monitoring and diagnosis. Zhang et al. (2020) proposed an edge-based intelligent healthcare system that enables real-time tracking and health analysis for patients. From (Chen et al., 2015)The automotive industry has also benefited from image classification techniques, particularly in the development of autonomous vehicles. Cameras and sensors mounted on self-driving cars capture and analyze real-time images of the surrounding environment. Image classification algorithms enable the vehicle to identify and classify objects such as pedestrians, traffic signs, and other vehicles, facilitating decision-making processes and ensuring safer navigation. Based on this research paper (Guo, 2018)-commerce platforms utilize image classification to enhance product search and recommendation systems. By analyzing images of products, AI algorithms can accurately categorize and tag items, enabling users to quickly find desired products and improving the overall shopping experience. Additionally, visual search capabilities allow users to take pictures of items they want to purchase and find similar products online. Security systems heavily rely on image classification for surveillance and threat detection. AI-powered cameras can analyze live video feeds and automatically identify suspicious activities, unauthorized access, or potential threats. This helps enhance public safety in various settings, including airports, banks, and public spaces. The above has been studied in (Ratha, Connell, and Bolle, 2001)

As image classification algorithms continue to advance, we can expect further integration into our daily lives. From personalized virtual assistants that understand and respond to visual cues to augmented reality applications that overlay information in the real world, the possibilities are vast.

## 1.3 Objectives

The overall objective of this thesis is to study different topologies of artificial neural networks and explore the impact of the hyperparameters and topologies Through a combination of empirical research, theoretical analysis, and practical implementation, this study seeks to achieve the following objectives:

1. To explore the biological inspiration behind the artificial neural network
2. To develop and compare neural network models using different activation functions.
3. To investigate the impact of hyperparameters on both topologies fully connected and convolutional networks.
4. To compare the performance of fully connected and convolutional networks on the MNIST and CIFAR-10 datasets.
5. To analyze the role of softmax in neural network training.

# Chapter 2

# Methods

## 2.1 Introduction

Artificial Neural Networks (ANNs) have revolutionized the field of machine learning and artificial intelligence, enabling computers to learn and perform tasks in a way that mimics the human brain. ANNs are computational models inspired by the structure and function of biological neural networks found in the human brain. Artificial neural networks are used fro a variety of tasks, including image and speech recognition, natural language processing, pattern recognition, recommendation systems, and even in activities that have traditionally been considered as reserved to humans, like painting.

## 2.2 Origin and history

Artificial neurons have a history dating back to mid-20th century in 1943 when Warren McCulloch and Walter Pitts published a paper proposing a model of artificial neurons that mimicked the behavior of real neurons in the brain.

Researchers including Frank Rosenblatt, Bernard Widrow, M. E. A. El-Masry, and Marvin Minsky developed more advanced artificial neurons, such as the perceptron and the adaptive linear neuron (ADALINE).

In 1974, Paul Werbos introduced the backpropagation algorithm, which enabled more complex neural networks with multiple layers of artificial neurons to be constructed and trained.

Nowadays, artificial neural networks are used in a wide variety of applications from natural language processing to healthcare. Deep learning neural networks are a subset of artificial neural networks that attempt to mimic the structure and function of the human brain.

## 2.3 Neural model

### 2.3.1 Biological inspiration

A biological neuron is an electrically excitable cell that plays a critical role in the functioning of the nervous system. It is made up of three main components: a cell body, an axon, and dendrites. The cell body contains the nucleus and other organelles that are responsible for maintaining the cell's metabolism. Dendrites branch out from the cell body to receive information from other neurons and relay it back to the cell body. The axon is a long, slender projection that carries electrical signals away from the cell body towards other neurons or target cells. At the end of the axon, there is a specialized junction or synapse where the electrical signal is transmitted to other cells using chemical messengers called neurotransmitters. The Fig. 2.1 shows a biological neuron .

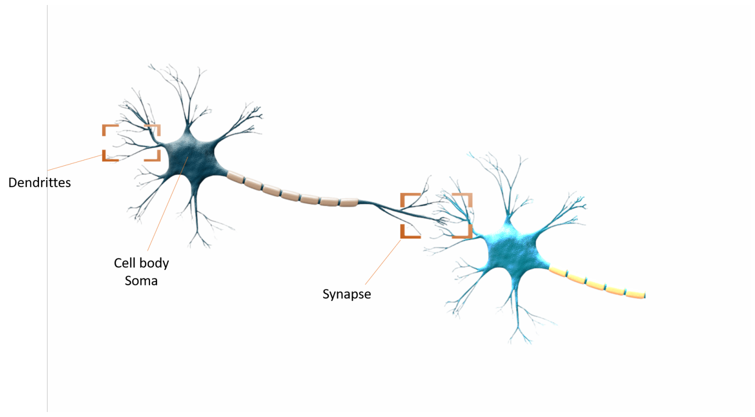


FIGURE 2.1: Biological neuron

### 2.3.2 Perceptron

A perceptron from (Rosenblatt, 1958) is a type of neural network algorithm that is used for supervised learning tasks, such as classification problems. The perceptron algorithm was first proposed by Frank Rosenblatt in 1958, and it consists of a single layer of neurons that take input signals and output a single binary value.

The perceptron algorithm works by taking a set of input signals, multiplying each input signal by a weight, and then summing up the weighted signals, as shown in Fig. 2.2 This sum is then passed through an activation function, such as a step function, which outputs a binary value of either 0 or 1. The perceptron algorithm adjusts the weights of the input signals based on the error between the predicted output and the actual output.

The perceptron algorithm has been used in a variety of applications, including image and speech recognition, and it is often used as a building block for more complex algorithms, such as multilayer perceptrons.

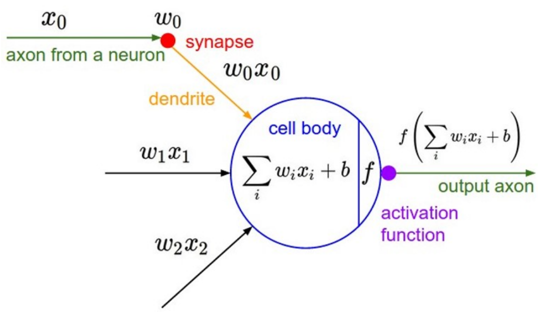


FIGURE 2.2: Perceptron

### 2.3.3 Multi layer perceptron

while Perceptrons are limited to linearly separable problems, multi-layer perceptrons (MLPs) can handle non-linear patterns using multiple layers of nodes with non-linear activation functions . One classic example of a non-linearly separable problem is the XOR problem illustrated in Fig. 2.3, which involves combining two inputs in a way that cannot be separated by a single hyperplane. MLPs are capable of solving the XOR problem by using multiple layers of nodes with non-linear activation functions. A multi layer perceptron is a feedforward neural network that

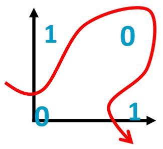


FIGURE 2.3: The XOR problem

consists of at least three layers of nodes, including an input layer, one or more hidden layers with nonlinear activation functions, and an output layer.

The Multi layer Perceptron learns by adjusting the weights of the connections between the nodes during the training process. . The goal of training an Multi layer perceptron is to minimize the difference between the predicted outputs of the network and the true outputs for a given set of training data. A multi layer perceptron is shown in Fig.2.4

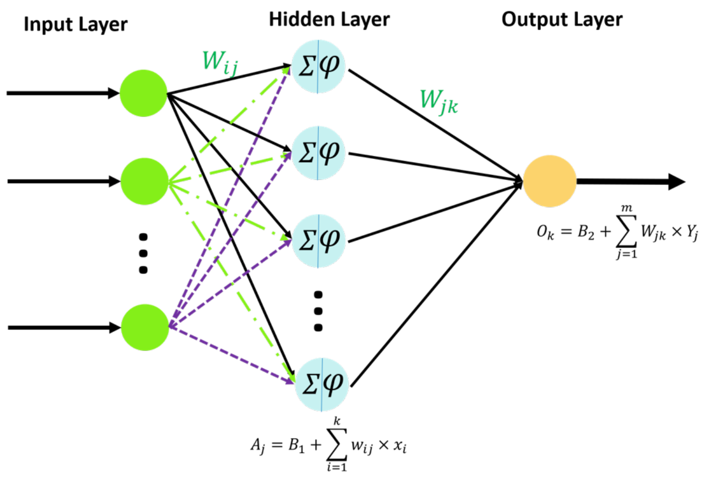

Input Layer
Hidden Layer
Output Layer
$W_{ij}$
$W_{jk}$
$\Sigma\varphi$
$O_k = B_2 + \sum_{j=1}^{m} W_{jk} \times Y_j$
$A_j = B_1 + \sum_{i=1}^{k} w_{ij} \times x_i$

FIGURE 2.4: Multi Layer Perception

### 2.3.4 Activation functions

Activation functions are a crucial component of artificial neural networks, including multi-layer perceptrons. They introduce non-linearity and allow neural networks to model complex relationships in data. Activation functions are applied to the output of each neuron in a network, transforming the weighted sum of inputs into a desired output or activation level. There are many different types of activation functions used in neural networks, each with their own strengths and weaknesses. Some common activation functions include the sigmoid function, as mentioned earlier, as well as the hyperbolic tangent function and the rectified linear unit (ReLU) function.

The choice of activation function depends on the specific problem being solved and the architecture of the network. Some functions are better suited to classification tasks, while others may be better for regression tasks. Additionally, some functions may perform better for shallow networks with few hidden layers, while others may be better suited for deep neural networks with many layers. The perceptron moder in Fig.2.5 shows the different activation functions of the neuron .

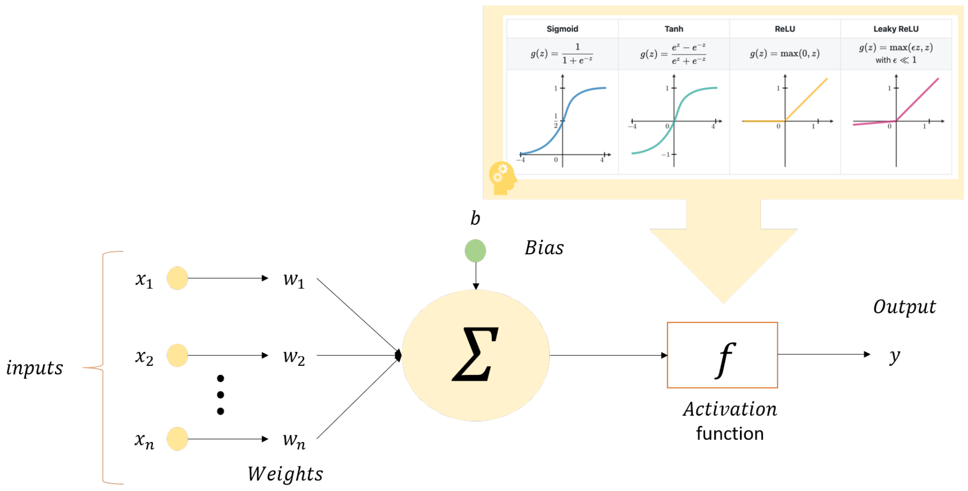


FIGURE 2.5: Perceptron model

## Sigmoid

One commonly used activation function is the sigmoid function, which maps the input to a value between 0 and 1. It is useful in binary classification tasks, where it can provide a probability-like output. However, the sigmoid function suffers from vanishing gradients as the inputs move away from the origin, which can impede the learning process.

The equation for the sigmoid function is:

$$f(x) = \frac{1}{1 + e^{-x}}$$

## ReLU

Another popular activation function is the rectified linear unit (ReLU), which returns the input as the output if it is positive and zero otherwise. ReLU has become widely adopted due to its simplicity and computational efficiency. It helps overcome the vanishing gradient problem and accelerates the training of deep neural networks.

$$RELU(x) = \begin{cases} 0 \; if \;\; x < 0 \\ x \; if \;\; x >= 0 \end{cases}$$

## Tanh

Hyperbolic tangent is also like logistic sigmoid but better. , wih the same S-shape but symmetric around the origin .It ranges from (-1 to 1).

$$f(x) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$

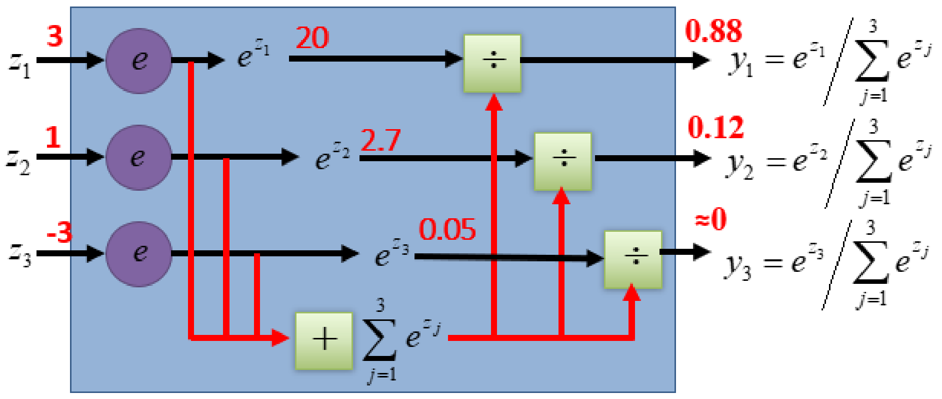
Tanh can introduce non-linearity and capture complex relationships in the data. Compared to the sigmoid function, tanh has the advantage of being zero-centered, which helps with gradient-based optimization algorithms.However, one limitation of the tanh function is that it can suffer from the "vanishing gradient" problem, especially for inputs far from zero. This means that the gradient becomes very small, which can slow down the learning process in deep neural networks.

#### Softmax

The softmax function operates by exponentiating each element of the input vector and then normalizing the resulting values to ensure they sum up to 1. This normalization process makes the output values represent probabilities. The larger the original input value, the larger its corresponding output value after softmax. The softmax function is explained in the Fig.2.6

*Probability*:
- $1 > y_i > 0$
- $\sum_i y_i = 1$

The softmax function is often used as the final activation function in the output layer of a neural network for multi-class classification tasks. It helps in obtaining class probabilities that can be used to make predictions and evaluate the model's confidence in its predictions.



$z_1$
3
$z_2$
1
$z_3$
-3
e
e
e
$e^{z_1}$
20
$e^{z_2}$
2.7
$e^{z_3}$
0.05
÷
÷
÷
+
$\sum_{j=1}^{3} e^{z_j}$
0.88
$y_1 = e^{z_1} \Big/ \sum_{j=1}^{3} e^{z_j}$
0.12
$y_2 = e^{z_2} \Big/ \sum_{j=1}^{3} e^{z_j}$
≈0
$y_3 = e^{z_3} \Big/ \sum_{j=1}^{3} e^{z_j}$

FIGURE 2.6: Softmax

The choice of activation function depends on the specific task and the properties of the data. Different activation functions can affect the network's ability to learn complex patterns, the speed of convergence during training, and the overall performance of the neural network. Experimentation and fine-tuning are often required to determine the most suitable activation function for a particular problem. Here is a summary of the most used activation function in Fig.

## 2.4 Neural topologies

### 2.4.1 Fully Connected Layers

Fully connected layers are a fundamental component in various neural network architectures. They consist of neurons that are connected to every neuron in the previous layer, forming a fully connected graph structure. Each neuron computes an

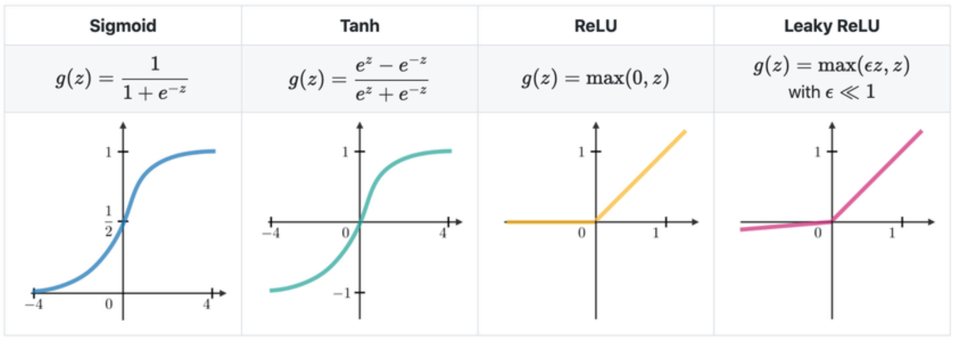


FIGURE 2.7: Activation functions

output based on a weighted sum of the inputs it receives from the previous layer, which is then passed through an activation function. As the number of neurons in a fully connected layer determines the dimensionality of its output, the output of one fully connected layer usually serves as the input to the next layer, creating a sequential arrangement of layers in a neural network.

Fully connected layers as in Fig.2.8 can detect complex patterns and relationships in data by adjusting their weights during the training process, making them powerful tools for tasks such as image classification and natural language processing. However, fully connected layers are computationally expensive and prone to overfitting due to having a high number of parameters. As the number of layers in the network increases, the computational cost grows significantly. Regularization techniques and optimization algorithms can be employed to reduce the risk of overfitting while maintaining the model's performance.

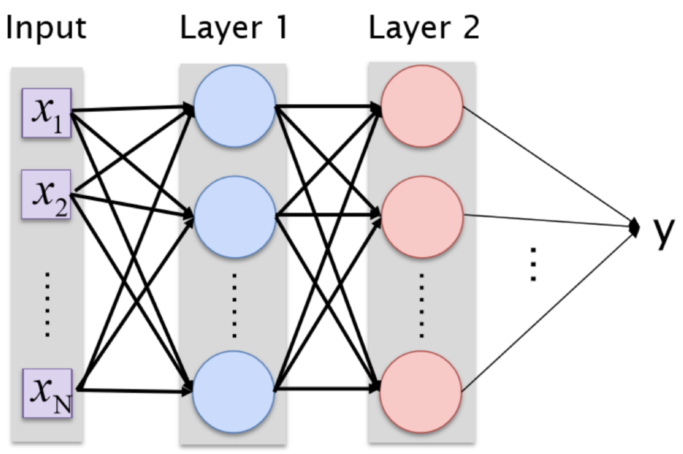


FIGURE 2.8: Fully connected

### 2.4.2 Convolutional and pulling layers

#### Convolution

CNN's Convolutional Neural Networks (CNNs) are Deep Learning algorithms that take in an image and assign importance to various aspects. They can also distinguish one object from another based on their learning. Unlike other classification systems, ConvNets do not require pre-processing. They can learn complex characteristics and filters using only a few training sessions. as shown in Fig.2.9

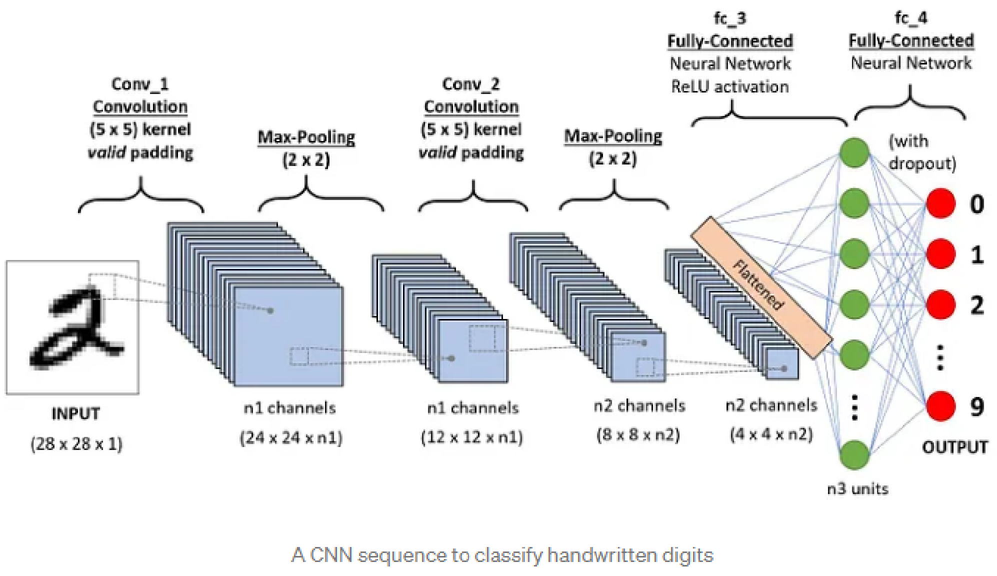


FIGURE 2.9: Convolutional sequence

CNN has multiple layers namely the convolutional layer, pooling layer, and fully connected layer. The main objective of the convolutional layer is to obtain the features of an image by sliding a smaller matrix (kernel or filter) over the entire image and generating the feature maps.The pooling layer is used to retain the most important aspect by reducing the feature maps. A fully connected layer interconnects every neuron in the layer to the neurons from the previous and next layer, to take the matrix inputs from the previous layers and flatten them to pass on to the output layer, which will make the prediction, as said in (Kadam, Adamuthe, and Patil, 2020) CNNs inspired by time-delay neural networks in which the weights are shared in temporal dimension for a reduction in computation.
IT was the first successful deep-learning architecture proved by Liu et al. (2017) To be the best option for machine learning applications.
The architecture was influenced by how the Visual Cortex is organized and is similar to the connectivity network of neurons in the human brain. Only in this constrained area of the visual field, known as the Receptive Field, do individual neurons react to stimuli. The entire visual field is covered by a series of such fields that overlap.

A representation of a convolutional layer is represented below in Fig.2.10

Pooling The Pooling layer, like the Convolutional Layer, is in charge of shrinking the Convolved Feature's spatial size. Through dimensionality reduction, the amount of computing power needed to process the data will be reduced. Additionally, it helps to extract dominating characteristics that are rotational and positional invariant, retaining the effectiveness of the model training process..

Max Pooling and Average Pooling in Fig.2.11are the two different types of pooling. The maximum value from the area of the image that the Kernel has covered is returned by Max Pooling. The average of all the values from the area of the image covered by the Kernel is what is returned by average pooling, on the other hand.

Additionally, Max Pooling functions as a noise suppressant. It also does denoising and dimensionality reduction in addition to completely discarding the noisy activations. Average Pooling, on the other hand, merely carries out dimensionality reduction as a noise-suppressing strategy. Therefore, we can conclude that Max Pooling outperforms Average Pooling significantly.

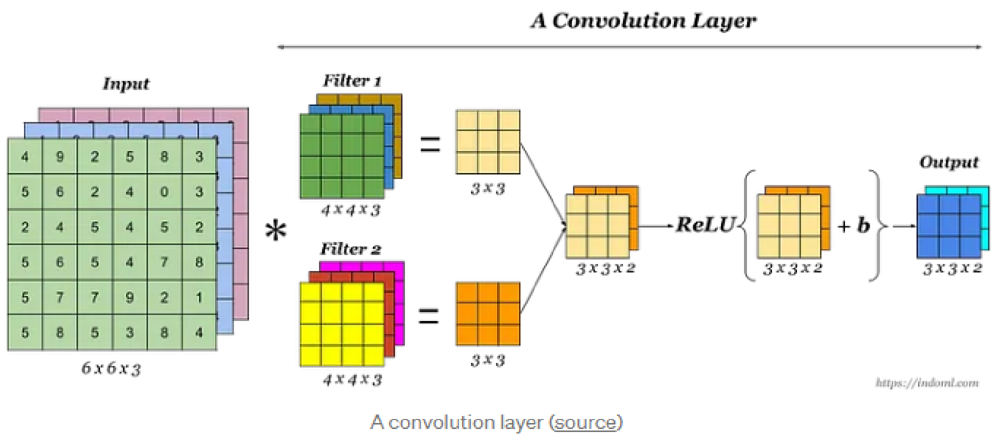


FIGURE 2.10: Convolutional layer

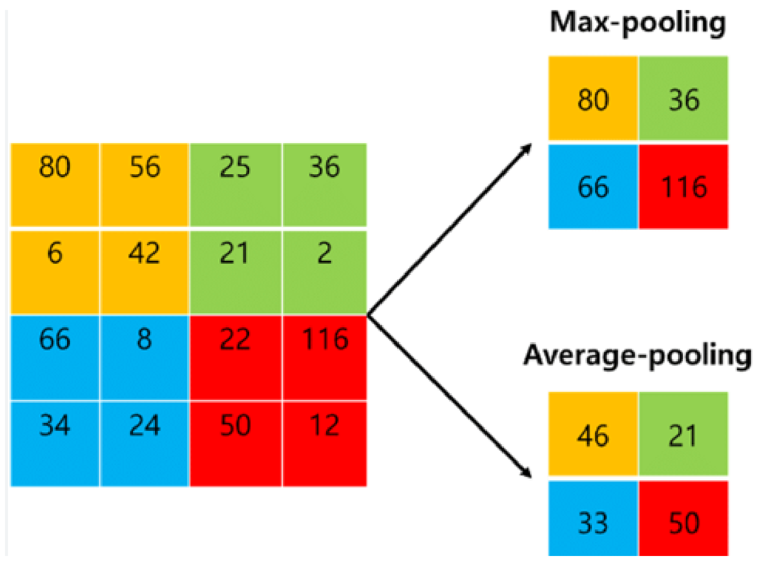


FIGURE 2.11: Types of pooling

## 2.5 Backpropagation

### 2.5.1 origin

It was first proposed by Paul Werbos in 1974 and further refined by researchers such as Rumelhart, Hinton, and LeCun. From(LeCun et al., 1998) Backpropagation uses the chain rule of calculus to calculate gradients required for updating the weights of a neural network. The algorithm gained widespread popularity in 1986 when Rumelhart, Hinton, and Williams demonstrated its effectiveness for training multi-layer neural networks. Backpropagation has since been extended and refined in numerous ways and is a key tool in the field of machine learning. LeCun and colleagues introduced the concept of convolutional neural networks, which use back-propagation to train multiple layers of filters for image recognition.

### 2.5.2 definition

Backpropagation defined by (LeCun et al., 1998) is a widely used algorithm in artificial neural networks for training models to minimize the difference between predicted and actual outputs. It is a form of supervised learning where the algorithm adjusts the weights of neurons in the network based on the calculated error.

Referring to (Rumelhart, Hinton, and Williams, 1986), backpropagation works by propagating the error back through the network, layer by layer, and adjusting the weights in each layer to reduce the overall error. This process involves computing the gradients of the error with respect to the network's weights and using these gradients to update the weights using an optimization algorithm, such as stochastic gradient descent.

### 2.5.3 Training

The algorithm involves a chain of steps to train a specific neural network by propagating the error back through the network and adjusting the weights.

It is an efficient application of the Leibniz chain rule (1673) from (Binder et al., 1997)

Split data

We start by splitting the data into 3 subsets. First, we have to define the exact number of examples that we want to be in each split of the training/validation sets
Number of training examples: 54000
Number of validation examples: 6000
Number of testing examples: 10000

The training data is the biggest (in -size) subset of the original dataset, which is used for picking weights to train or fit the machine learning model. Firstly, the training data is fed to the ML algorithms, which lets them learn how to make predictions for the given task.

The MNIST dataset comes with a training and test set, but not a validation set. We want to use a validation set to check how well our model performs on unseen data. We should only be measuring our performance over the test set once, after all, training is done. Furthermore, we create a validation set, taking 10 percent of the training set. If the validation set is taken from the test set, then the test set is not the same as everyone else's and the results cannot be compared against each other.

Once we train the model with the training dataset, it's time to test the model with the test dataset. This dataset evaluates the performance of the model and ensures that the model can generalize well with the new or unseen dataset. The test dataset is another subset of the original data, which is independent of the training dataset. It should be large enough to give meaningful predictions.

#### Backpropagation algorithm

At the end of each forward pass through a network, the algorithm performs a backward pass in which it adjusts the model's parameters (weights and biases). The backpropagation algorithm is probably the most fundamental building block in a neural network. It was first introduced in the 1960s and almost 30 years later (1989) popularized by Rumelhart, Hinton, and Williams in a paper called "Learning representations by back-propagating errors" According to this paper, (Rumelhart, Hinton, and Williams, 1986). The algorithm aims to repeatedly adjust the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector.

It aims to minimize the cost function by adjusting the network's weights and biases. The level of adjustment is determined by the gradients of the cost function with respect to those parameters.
**Advantages**

1. No need to tune the parameters except for the number of inputs.
2. Highly adaptable and efficient
3. Does not require any prior knowledge about the network.
4. Fast and easy to program.
5. No need to learn any special functions.
6. They efficiently compute the gradient of the loss function with respect to the network weights. It enables the use of gradient methods, like gradient descent.
7. The concept of momentum in backpropagation states that previous weight changes must influence the present direction of movement in weight space.

### Error calculation

y(x,w) corresponds to the predicted output of the network of weights w (a vector), with input data x (a vector).

$y^m$is the known output for the training data $x^m$.

Error over the training set for a given weight vector is provided in Fig.2.12.

$$E = \frac{1}{2} \sum_{\substack{\text{on the training}\\ \text{samples}}} (y(\text{x,w}) - y^m)^2$$

FIGURE 2.12: Error calculation

The purpose is to find the weight that will minimize this error.

### Descent rule

Taking into account a fully connected network with sigmoid as an activation function such as the gradient of the training error is computed as a function of the weights in the Fig.2.13 and Fig.2.14

### Delta rule

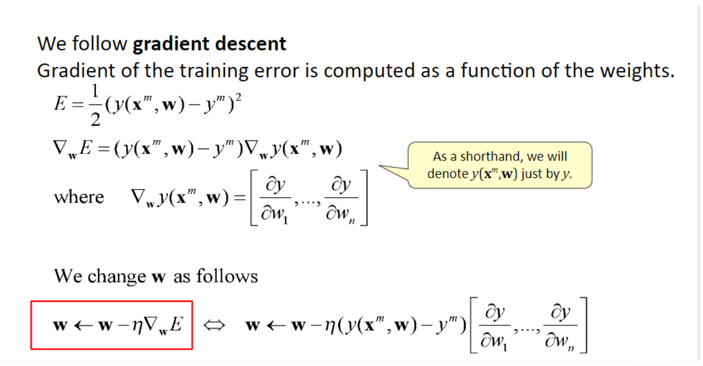
In general, for a hidden unit j we have the delta rule is in Fig.2.16

After each forward prop, we have to :

Compute $\delta p$ for the output layer:

$$\delta p = y_p(1 - y_p)(y_p - y_m)$$

Compute $\delta j$ for all preceding layers by **the backpropagation rule.**



We follow **gradient descent**

Gradient of the training error is computed as a function of the weights.

$$E = \frac{1}{2}(y(\mathbf{x}^m, \mathbf{w}) - y^m)^2$$

$$\nabla_{\mathbf{w}} E = (y(\mathbf{x}^m, \mathbf{w}) - y^m)\nabla_{\mathbf{w}} y(\mathbf{x}^m, \mathbf{w})$$

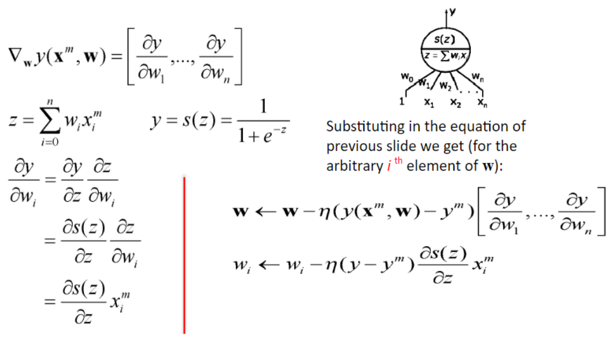
where $\nabla_{\mathbf{w}} y(\mathbf{x}^m, \mathbf{w}) = \left[\frac{\partial y}{\partial w_1}, \dots, \frac{\partial y}{\partial w_n}\right]$

As a shorthand, we will denote $y(\mathbf{x}^m, \mathbf{w})$ just by $y$.

We change $\mathbf{w}$ as follows

$$\boxed{\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} E} \Leftrightarrow \mathbf{w} \leftarrow \mathbf{w} - \eta(y(\mathbf{x}^m, \mathbf{w}) - y^m)\left[\frac{\partial y}{\partial w_1}, \dots, \frac{\partial y}{\partial w_n}\right]$$

FIGURE 2.13: Gradient descent



$$\nabla_{\mathbf{w}} y(\mathbf{x}^m, \mathbf{w}) = \left[\frac{\partial y}{\partial w_1}, \dots, \frac{\partial y}{\partial w_n}\right]$$

$$z = \sum_{i=0}^{n} w_i x_i^m \qquad y = s(z) = \frac{1}{1+e^{-z}}$$

$$\frac{\partial y}{\partial w_i} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i} = \frac{\partial s(z)}{\partial z}\frac{\partial z}{\partial w_i} = \frac{\partial s(z)}{\partial z} x_i^m$$

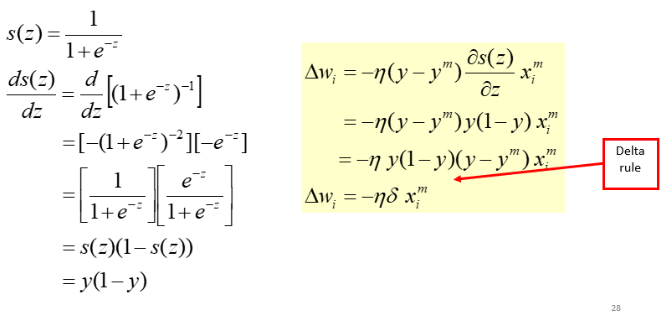
y
s(z)
z = Σ w_i x_i
w_0
w_1
w_2
w_n
1
x_1
x_2
x_n

Substituting in the equation of previous slide we get (for the arbitrary $i^{th}$ element of $\mathbf{w}$):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta(y(\mathbf{x}^m, \mathbf{w}) - y^m)\left[\frac{\partial y}{\partial w_1}, \dots, \frac{\partial y}{\partial w_n}\right]$$

$$w_i \leftarrow w_i - \eta(y - y^m)\frac{\partial s(z)}{\partial z} x_i^m$$

FIGURE 2.14: Backprop and descent rule



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

$$\frac{ds(z)}{dz} = \frac{d}{dz}\left[(1+e^{-z})^{-1}\right] = [-(1+e^{-z})^{-2}][-e^{-z}] = \left[\frac{1}{1+e^{-z}}\right]\left[\frac{e^{-z}}{1+e^{-z}}\right] = s(z)(1-s(z)) = y(1-y)$$

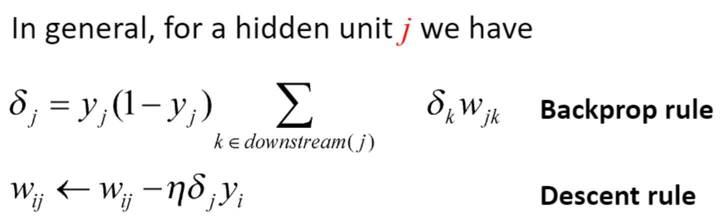
$$\Delta w_i = -\eta(y - y^m)\frac{\partial s(z)}{\partial z} x_i^m = -\eta(y - y^m)y(1-y)\, x_i^m = -\eta\, y(1-y)(y - y^m)\, x_i^m$$

$$\Delta w_i = -\eta \delta\, x_i^m$$

Delta rule

28

FIGURE 2.15: Derivative of sigmoid



In general, for a hidden unit $j$ we have

$$\delta_j = y_j(1 - y_j) \sum_{k \in downstream(j)} \delta_k w_{jk} \qquad \textbf{Backprop rule}$$

$$w_{ij} \leftarrow w_{ij} - \eta \delta_j y_i \qquad \textbf{Descent rule}$$

FIGURE 2.16: Generalized delta rule

Compute weight change by **descent rule** (repeat for all weights)

- The process of backpropagation in a MLP is shown below in Fig. 2.17 taken from (Khacef, Abderrahmane, and Miramond, 2018a).

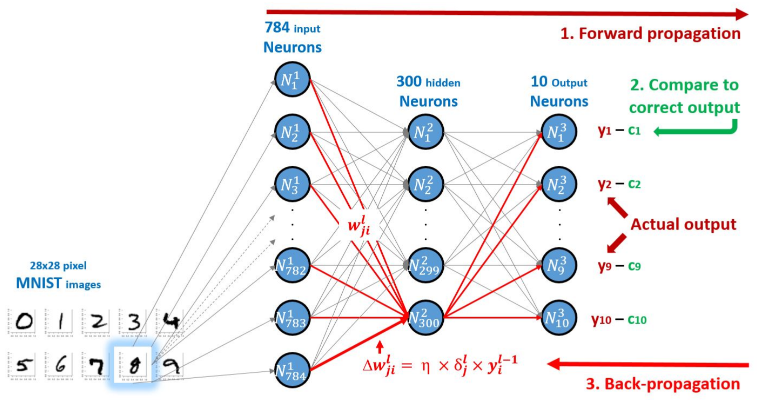


FIGURE 2.17: Gradient back-propagation learning algorithm

## Learning rate

The learning rate is a hyper-parameter that controls the weights of our neural network with respect to the loss gradient. It defines how quickly the neural network updates the concepts it has learned.

A desirable learning rate is low enough that the network converges to something useful, but high enough that it can be trained in a reasonable amount of time. For our experiment, we chose our learning rate = 0.01.

## Optimizer

Local minimum as in Fig.2.18 is a point where the function reaches a relatively low value within a specific neighborhood. The presence of local minima can pose challenges in optimization problems because algorithms may get stuck in these suboptimal points, preventing them from reaching the global minimum. To avoid local minima, we use optimizers like momentum.

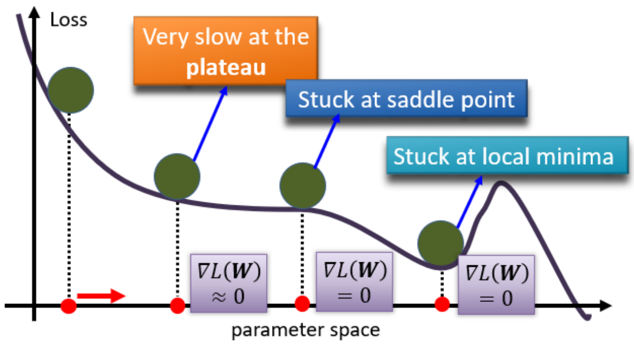

Loss
Very slow at the plateau
Stuck at saddle point
Stuck at local minima
$\nabla L(\boldsymbol{W}) \approx 0$
$\nabla L(\boldsymbol{W}) = 0$
$\nabla L(\boldsymbol{W}) = 0$
parameter space

FIGURE 2.18: Local minima

Momentum, based on (Qian, 1999) is a technique used to improve the convergence of optimization algorithms. Inspired by the laws of physics, momentum adds a velocity component to the traditional gradient descent algorithm. Instead of solely relying on the local gradient information, momentum considers the previous updates and incorporates them into the current iteration. The process is illustrated below in Fig.**??**

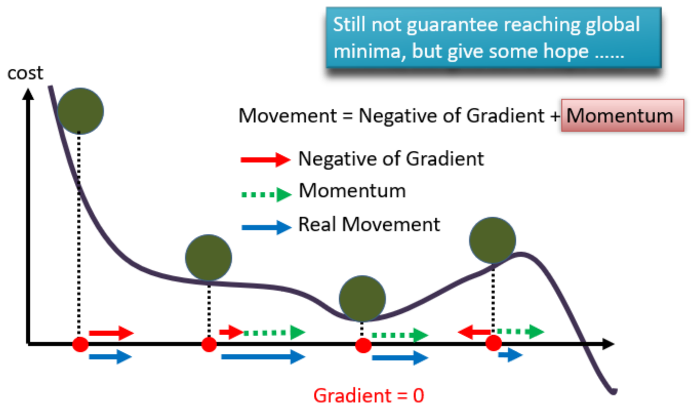

Still not guarantee reaching global minima, but give some hope ......
cost
Movement = Negative of Gradient + Momentum
Negative of Gradient
Momentum
Real Movement
Gradient = 0

FIGURE 2.19: Momentum

## 2.6 Hardware complexity

### 2.6.1 The number of neurons

In the context of neural networks, estimating neurons refers to determining the optimal number of neurons to use in each network layer. This is an important step in designing and training neural networks, as the number of neurons can significantly impact the network's performance. Few neurons can result in underfitting, where the model cannot capture the complexity of the data. At the same time, too many neurons can result in overfitting, where the model fits the training data too closely and performs poorly on new data. Various methods estimate the optimal number of neurons, including trial and error, cross-validation, and more advanced techniques such as pruning. The above has been taken from (Hagan and Menhaj, 1994) The number of neurons in a neural network architecture is often referred to as the "neuron count" or "size" of the network. We can find the number of neurons in an FC neural network simply by adding the number of neurons of each hidden layer and the output layer. We generally neglect the neurons of the input layer because those are not subjected to an activation function. Their function is just to bring the initial data.

### 2.6.2 The number of synapses

The number of synapses in a neural network depends on its architecture and the number of parameters (weights) used in the network. Each parameter (weight) in the network corresponds to a synapse.

In a typical feedforward neural network, the number of synapses can be calculated by counting the number of connections between neurons in each layer. Assuming a fully connected architecture, where each neuron is connected to every neuron in the subsequent layer, the number of synapses can be approximated using the following formula:

$$\textit{Number of synapses} \approx \sum_{i=1}^{L-1} (\textit{Number of neurons in layer } i) \times (\textit{Number of neurons in layer } (i+1))$$

where **L** represents the total number of layers in the neural network. Each term in the summation represents the product of the number of neurons in a layer and the number of neurons in the subsequent layer.

It's important to note that this formula assumes a single-layer neural network and additional terms need to be added for each subsequent layer.

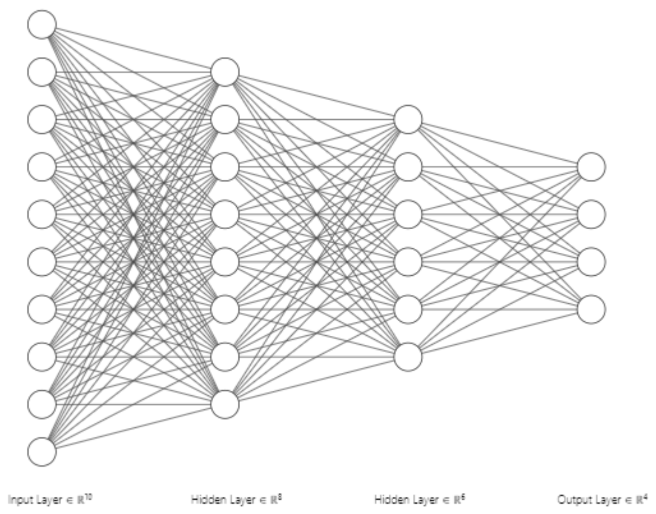
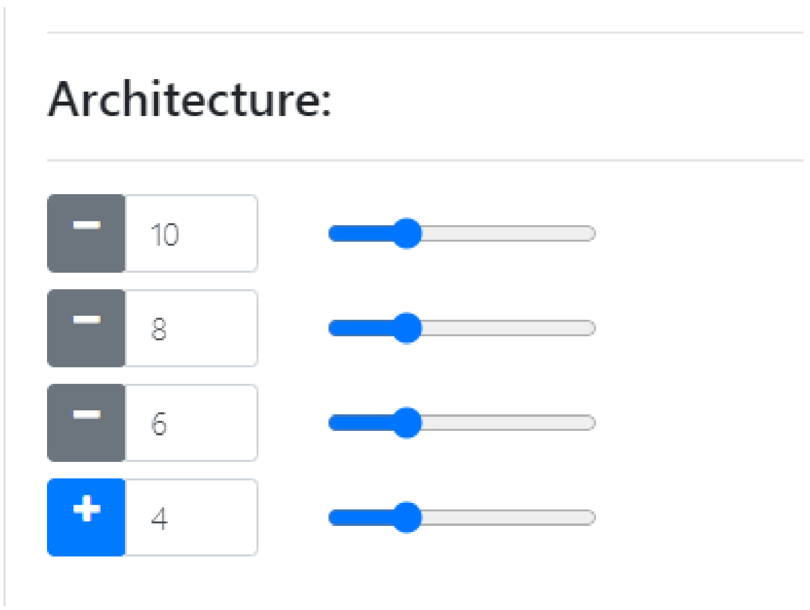


FIGURE 2.20: Example for counting the number of synapses



Architecture:

- 10
- 8
- 6
- 4

FIGURE 2.21: the architecture of the network for counting the number of synapses

In the example of Fig.2.20 with an architecture shown in Fig.2.21, and from the formula given before, the number of weights or synapses of the neural network is :

$$\text{number of weights} = (10 \times 8) + (8 \times 6) + (6 \times 4) = 152$$

For more complex neural network architectures, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), the calculation of the number of synapses becomes more involved due to their specialized structures. The number of synapses will depend on the specific design choices made in constructing the network.

Overall, the exact number of synapses in a neural network can vary significantly depending on its architecture, depth, and the number of parameters (weights) used in the network.

# Chapter 3

# Experiments and Results

## 3.1 Introduction

In this chapter, we present the results of our experiments aimed at evaluating the performance and effectiveness of our proposed approach. We outline the datasets used in our experiments, including the MNIST and Cifar-10 datasets, and describe our experimental methodology. Our primary objective is to evaluate the performance of our approach in terms of accuracy, speed, and robustness. We also investigate the impact of various hyperparameters on the overall performance of our model. We conclude the chapter by interpreting and discussing the results of our experiments, highlighting both the strengths and limitations of our approach and providing direction for future research.

## 3.2 Datasets

### 3.2.1 MNIST

A well-known dataset of handwritten numbers called MNIST is represented in Fig.3.1. was created by two researchers: Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. It is frequently used as a benchmark for machine learning image classification applications. The 70,000 handwritten digits from 0 to 9 in grayscale pictures, each measuring 28 by 28 pixels, make up the MNIST dataset. The dataset is divided into two parts: 60,000 photos for machine learning model training and 10,000 images for accuracy testing.

Each picture in the MNIST collection is connected with the right digit it represents since the images are labeled. Because of this, the MNIST dataset is very beneficial for jobs like digit recognition as in Fig.3.2, which requires locating the digits in a picture.

Academics frequently utilize the MNIST dataset to assess the effectiveness of many machine-learning techniques, notably in computer vision. The dataset is a desirable option for assessing the precision and external validity of various models due to the high-quality labels, numerous samples, and well-defined test sets.

The MNIST dataset continues to be crucial in the development of machine learning algorithms and serves as a common reference point for academics in the field, despite its antiquity and limited size in comparison to more current datasets.

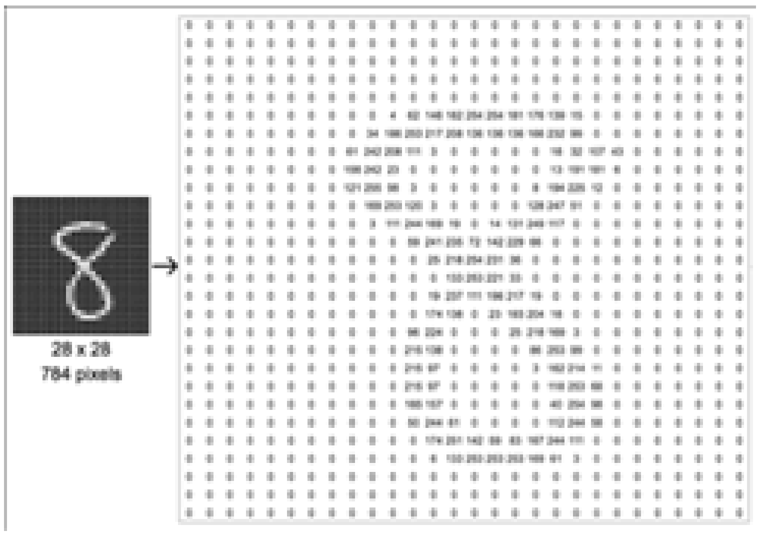


FIGURE 3.1: MNIST Dataset

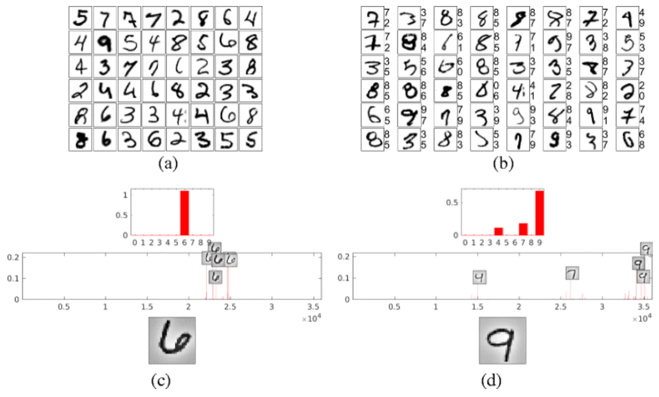


FIGURE 3.2: Results for the MNIST dataset

### 3.2.2 Cifar10

CIFAR-10 from (Krizhevsky, Hinton, et al., 2009) is a well-known dataset of small color images that are often used as a benchmark for image classification tasks in machine learning. It was created by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The CIFAR-10 dataset consists of 60,000 32 x 32 pixel color images of 10 different objects, each belonging to one of 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck).

The images in the CIFAR-10 dataset are labeled, which means that each image is associated with the correct class label. This makes the CIFAR-10 dataset particularly useful for tasks such as object recognition, which involves identifying the objects in a given image.

Researchers often use the CIFAR-10 dataset shown in Fig.3.3 to evaluate the performance of different machine learning algorithms, particularly in the field of computer vision. The high-quality labels, large number of examples, and well-defined test set make the dataset an attractive choice for evaluating the accuracy and generalization of different models.

Despite its relatively small size compared to more recent datasets, the CIFAR-10 dataset continues to play an important role in the development of machine learning algorithms and serves as a standard reference point for researchers in the field.

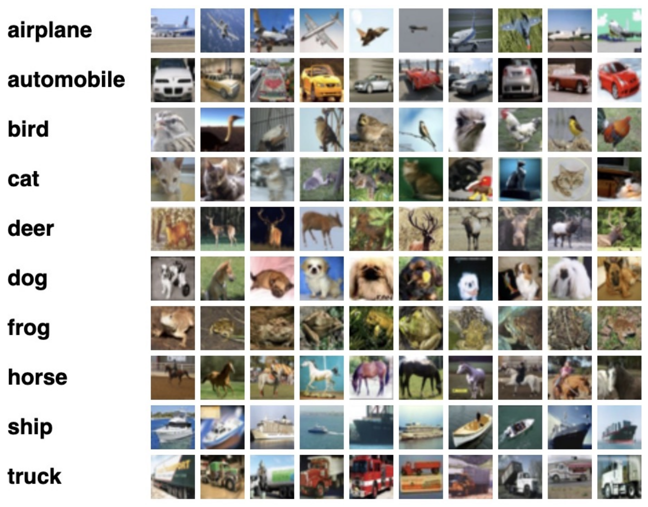


FIGURE 3.3: the Cifar10 dataset

## 3.3 Effects of Layer Depth on Fully Connected Networks

The following graph in Fig.**??**represents the average results of a simulation of 3 codes of different artificial neural networks with various numbers of hidden layers without changing the number of neurons in each layer. For our experiment, we choose to compare the training accuracy on MNIST classification between 1,2, and 3 hidden layers with 200 neurons in each. In the graph above, we can clearly see that the 3 curves converge to approximately 98 % . We also notice that the FCNN with 3 hidden layers is the one that converges the first, followed by the one with 1 hidden layer. The last one is the one with 2 hidden layers. such that:
The results are resumed in the table.3.2

TABLE 3.1: The effect of increasing the number of hidden layers on the acuracy

| **Neurons** | **Efficiency (%)** |
|---|---|
| 100 | 85 |
| 200 | 88 |
| 300 | 91 |
| 400 | 92 |
| 500 | 94 |

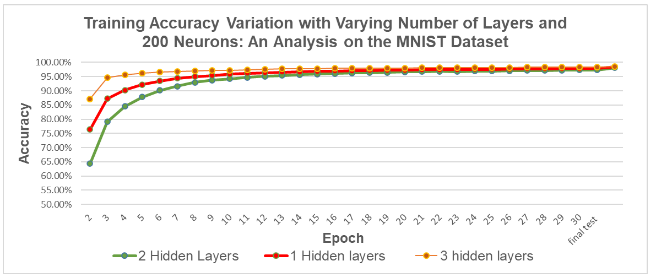


FIGURE 3.4: Variations with varying number of layers

Based on (LeCun, Bengio, and Hinton, 2015)and from what we noticed during the simulations : In general, increasing the number of hidden layers in a fully connected neural network for image classification on the MNIST dataset can have both positive and negative effects:
**Advantages** :

- Capacity for Learning Complex Features: Adding more hidden layers increases the capacity of the neural network to learn complex and hierarchical representations of the input data. This can enable the network to capture intricate patterns and features present in the MNIST images.

**Disadvantages** :

- Risk of Overfitting: As the number of hidden layers increases, the model becomes more capable of memorizing the training data, leading to a higher risk of overfitting. It occurs when the model becomes too specialized to the training set and fails to generalize well.
- Vanishing: Deeper networks are more prone to vanishing or exploding gradient problems during training. The gradients can become extremely small or large, making it difficult for the network to update the weights effectively.
- Increased Computational Complexity: As the number of hidden layers increases, the computational complexity of the network also grows. Deeper networks require more parameters to be trained, which can increase training time and resource requirements.

**Conclusion** : To achieve optimal performance, finding the right balance in the number of hidden layers and other hyperparameters is crucial. Researchers often employ techniques such as cross-validation and grid search to determine the optimal network architecture and hyperparameters for specific tasks.

## 3.4 Exploring Neuron Variations in Two Hidden Layers Neural Networks

The graph in Fig. 3.5.presented illustrates the average results obtained from a simulation involving three artificial neural networks with varying numbers of neurons while keeping the number of layers constant. Specifically, the experiment focuses on comparing the training accuracy achieved on the MNIST classification task using ANN architectures with two hidden layers and different neuron counts, namely 100, 200, and 300. From the graph, we found that all three curves demonstrate a convergence of approximately 98% accuracy. such that:

TABLE 3.2: The effect of increasing the number of neurons on the acuuracyy

| Neurons | Efficiency (%) |
|---|---|
| 100 | 85 |
| 200 | 88 |
| 300 | 91 |
| 400 | 92 |
| 500 | 94 |

We also notice that the FCNN with 100 neurons is the one that converges first, followed by the one with 200 neurons. The last one is the one with 300 neurons. This suggests that as the number of neurons increases, the convergence time tends to slightly lengthen.

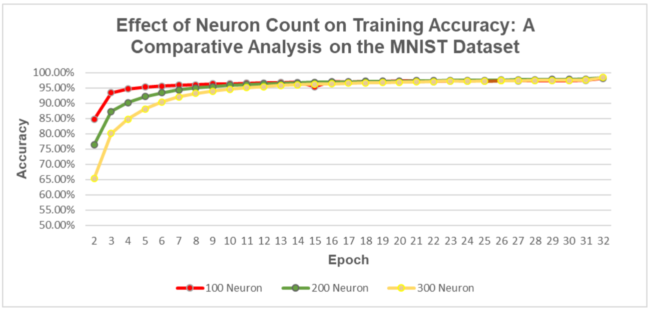


FIGURE 3.5: Effect of Neuron count on training accuracy

**Advantages**:

- Improved Model Flexibility: Increasing the number of neurons in a hidden layer can enhance the model's capacity to learn complex patterns and improve its ability to capture intricate relationships within the data. This flexibility allows the neural network to achieve higher accuracy. (Bengio, LeCun, et al., 2007)

- Potential for Higher Accuracy: Increasing the number of neurons allows the neural network to represent more intricate decision boundaries, potentially resulting in improved training accuracy. (He et al., 2016)

**Disadvantages**:

- Increased Model Complexity: A higher number of neurons increases the complexity of the neural network, making it more prone to overfitting, especially when the training dataset is limited. Balancing model complexity is essential to prevent overfitting and maintain generalization. (Srivastava et al., 2014)

- Higher Computational Requirements: Increasing the number of neurons leads to a larger model, requiring more computational resources and potentially longer training times. This can limit the scalability of the model, particularly in resource-constrained environments. (Simonyan and Zisserman, 2014)

## 3.5 Investigating the impact of the increased number of neurons and synapses on accuracy in MLP

For the purpose of exploring the effect of increasing the number of neurons in a fully connected neural network implementation, after doing the simulation for all one, two, and three hidden layers, we selected 2 hidden layers with multiple numbers of neurons for each type. we then calculate the number of neurons and synapses of each: 100, 200,300,400,500,600, and 700 neurons in each layer. the results are shown in table3.3

TABLE 3.3: Impact of number of neurons and synapses on the accuracy

| **Number of Neurons in Hidden Layers** | **Total Number of Neurons** | **Weights** | **Efficiency (%)** |
|---|---|---|---|
| 100 | 210 | 109400 | 97.10% |
| 200 | 410 | 198800 | 97.38% |
| 300 | 810 | 388200 | 97.80% |
| 400 | 1010 | 478000 | 97.68% |
| 500 | 1001 | 647000 | 98.01% |
| 600 | 1210 | 836400 | 98.20% |
| 700 | 1410 | 1045800 | 98.39% |

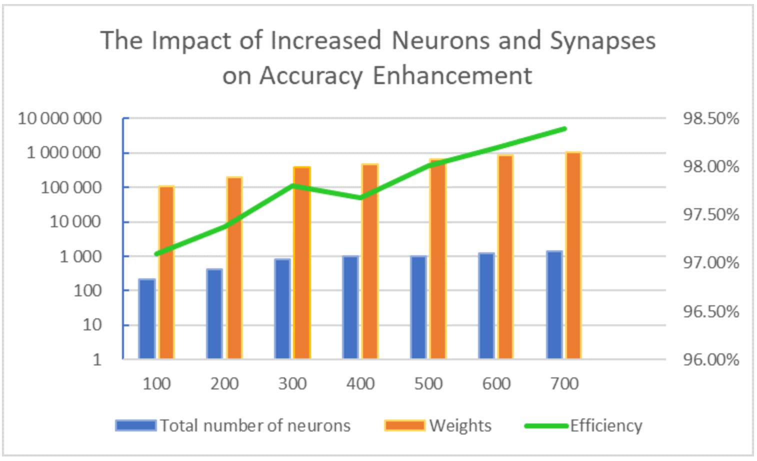


FIGURE 3.6: The Impact of Increased Neurons and Synapses on Accuracy Enhancement

We clearly see that increasing the number of neurons increases the accuracy. However, it is not the best choice for our neural networks because, from (Khacef, Abderrahmane, and Miramond, 2018b), today's most applications such as smart devices or autonomous vehicles require an embedded implementation of neural networks. Their implementation in CPU/GPU remains too expensive, mostly in energy consumption, due to the non-adaptation of the hardware to the computation model, which becomes a limit to their use.(Khacef, Abderrahmane, and Miramond, 2018b)
**Conclusion** The results obtained from our simulation on the MNIST classification task demonstrate that increasing the number of neurons in two hidden layers leads to improved training accuracy. Specifically, we observed a positive correlation between the number of neurons and training accuracy, with the accuracy converging to approximately 98%. These findings emphasize the significance of neuron count

in determining the performance of artificial neural networks (ANNs) and provide valuable insights for optimizing the architecture of ANNs to achieve higher accuracy.

## 3.6 Advancing Insights into the Influence of Softmax on Neural Network Output with Homogeneous Activation Functions

The results are resumed in table.3.3 The following graph in Fig .3.7compares an FCNN with a sigmoid function at its output layer with one with sigmoid at all its layers. The curves have approximately the same shape, The sigmoid one converges the first to 96.09% while the sigmoid converges slower but better, such that its accuracy is 97.15%.

TABLE 3.4: Impact of softmax at the output layer

| Softmax | Train | Efficiency |
|---|---|---|
| Without | 1 | 96.99% |
| Without | 2 | 96.97 |
| Without | 3 | 97% |
| With | 4 | 97.63% |
| With | 5 | 96% |
| With | 6 | 97.66% |

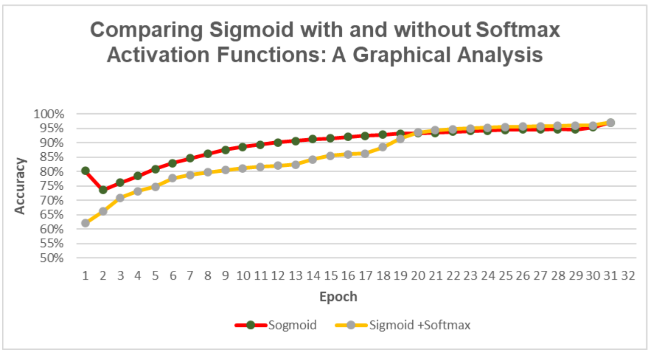


FIGURE 3.7: Comparing Sigmoid with and without Softmax

From various results of research as (LeCun et al., 1998) and (Géron, 2022) with (Bishop and Nasrabadi, 2006), the impact of softmax on the performance of a fully connected neural network for MNIST classification can be summarized as follows :

- Probability Interpretation: The softmax function ensures that the output of the network represents probabilities by normalizing the outputs across all classes.
- Facilitating Decision-Making: By converting the network's output into probabilities, softmax enables us to make informed decisions. We can select the class with the highest probability as the predicted class for the input image. This decision-making process becomes more straightforward and intuitive with the help of softmax.

- Handling Multiclass Classification: Softmax is well-suited for multiclass classification problems, such as the MNIST dataset, where each input image can belong to one of multiple classes (0 to 9 in the case of MNIST). Softmax assigns a probability to each class, allowing us to estimate the likelihood of the input image belonging to each class individually.

**Conclusion** :

softmax is a vital component of the fully connected neural network for MNIST classification. It converts the network's output into probabilities, facilitating decision-making and enabling training with cross-entropy loss. Softmax ensures that the network produces meaningful predictions and enables the interpretation of the network's confidence in its classifications.

## 3.7 Impact of CNN on MNIST dataset

For the above graph in Fig.3.8, we added convolution to our previous FCNN with 3 hidden layers and 200 neurons in each. It is clear that the blue curve of the CNN is faster than the one without convolution. such that it converges faster and reaches **99.19%** of accuracy while the red one which represents the FCNN reaches only **97.79%**.The average of each 3 simulation have been summarized in table.**??**

TABLE 3.5: The impact of CNN on the accuracy of MNIST classification.

| Train | final test |
|---|---|
| MLP Average | 97% |
| CNN +MLP Average | 99.19% |

- Note that the percentages are the average of three different simulations with different random seeds each time

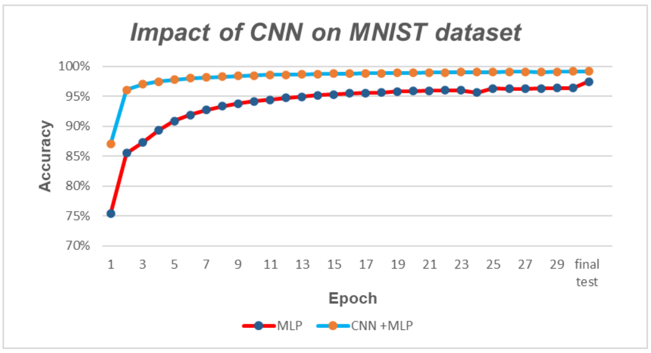


FIGURE 3.8: Impact of CNN on MNIST dataset

According to (LeCun et al., 1998) the effect of convolution in MNIST classification may not be as significant compared to more complex datasets is primarily due to the simplicity and uniformity of the MNIST dataset. The reasons are:

- Simple and Distinct Features: MNIST digits have simple and distinct features, such as lines and curves, which can be easily captured by fully connected networks.
- Limited Spatial Variability: MNIST images contain digits that are centered and occupy a significant portion of the image space. There is limited spatial variability in terms of digit positions, orientations, and deformations.

While fully connected networks can achieve excellent accuracy on MNIST, it's worth noting that convolutional layers can still provide benefits. They can help extract local features, capture spatial relationships, and reduce the number of parameters in the network.
**Conclusion** :
In conclusion, while the effect of convolution in MNIST classification may not be as significant as in more complex datasets, convolutional layers can still enhance performance by leveraging local feature extraction and spatial relationships.

## 3.8 Impact of CNN on Cifar10 Dataset

In the presented graph in Fig.3.9, we conducted experiments on the Cifar10 open-source dataset to compare the performance of a fully connected artificial neural network (FCNN) with and without the addition of convolution. Our study focused on a 3 hidden layers architecture with 200 neurons.

The graph clearly illustrates the significant advantage of incorporating convolution, as depicted by the orange curve representing the convolutional neural network (CNN). The CNN exhibits faster convergence and achieves an impressive accuracy of **76.08%**, surpassing the performance of the FCNN, represented by the purple curve, which reaches a maximum accuracy of only **44.57%**. We summarized the resulats in table.3.6

These findings highlight the effectiveness of convolutional layers in enhancing the performance of artificial neural networks for image classification tasks. The CNN's ability to capture local spatial dependencies and extract meaningful features from the input data contributes to its superior accuracy compared to the FCNN.

TABLE 3.6: The impact of CNN on the accuracy of CIFAR10 classification.

| Epoch | final test |
|---|---|
| CNN + MLP | 76.08% |
| MLP | 44.57% |

- Note that the percentages are the average of three different simulations with different random seeds each time

**Conclusion** :
our analysis of the Cifar10 dataset, which consists of RGB-based pixel images, highlights the importance of combining convolutional neural networks (CNN) and fully connected neural networks (FCNN) for achieving high accuracy in image classification tasks. The experiments conducted with 3 hidden layers and 200 neurons architecture demonstrate that incorporating convolutional layers significantly improves the performance compared to using only fully connected layers. According to the

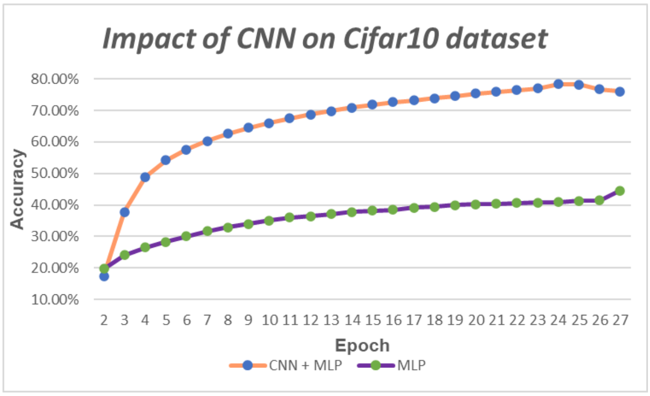


FIGURE 3.9: Impact of CNN on Cifar10 dataset

study, (Krizhevsky, Sutskever, and Hinton, 2017) CNNs have much fewer connections and parameters and so they are easier to train. Thus, The CNN's ability to capture spatial dependencies and extract meaningful features from the RGB-based pixel data allows it to achieve a higher accuracy of **76.08%**.

## 3.9 Conclusion

Achieving 100% accuracy in training a neural network is often not feasible . There are several reasons for that like limited Training Data, because Neural networks learn from data, and the training process aims to generalize patterns from the available training data. If the training dataset is limited or does not encompass the full complexity of the problem, the network may not achieve 100 %accuracy. It's important to note that the goal of training a neural network is to achieve the best possible performance on unseen data rather than aiming for perfect accuracy on the training set.

The choice of hyperparameters is particularly crucial when designing neural networks for embedded systems. In such resource-constrained environments, as studied in the paper (Khacef, Abderrahmane, and Miramond, 2018a) their implementation in CPU/GPU remains too expensive, principally in energy consumption, due to the non-adaptation of the hardware to the computation model, which becomes a limit to their use.

Efficient use of hardware is essential. Therefore, careful selection of hyperparameters such as the number of hidden layers, neurons, and activation functions is paramount to finding the right balance between model complexity and computational efficiency. Optimal hyperparameter configurations allow neural networks to be effectively deployed in embedded systems, overcoming limitations caused by limited resources. The goal is to make neural networks widely deployable in a variety of real-world applications, even in resource-constrained scenarios.

# Bibliography

Alam, Firoj et al. (2020). “Deep learning benchmarks and datasets for social media image classification for disaster response”. In: *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, pp. 151–158.

Bengio, Yoshua, Yann LeCun, et al. (2007). “Scaling learning algorithms towards AI”. In: *Large-scale kernel machines* 34.5, pp. 1–41.

Binder, John et al. (1997). “Adaptive probabilistic networks with hidden variables”. In: *Machine Learning* 29, pp. 213–244.

Bishop, Christopher M and Nasser M Nasrabadi (2006). *Pattern recognition and machine learning*. Vol. 4. 4. Springer.

Chen, Chenyi et al. (2015). “Deepdriving: Learning affordance for direct perception in autonomous driving”. In: *Proceedings of the IEEE international conference on computer vision*, pp. 2722–2730.

De Fauw, Jeffrey et al. (2018). “Clinically applicable deep learning for diagnosis and referral in retinal disease”. In: *Nature medicine* 24.9, pp. 1342–1350.

Esteva, Andre et al. (2017). “Dermatologist-level classification of skin cancer with deep neural networks”. In: *nature* 542.7639, pp. 115–118.

Gandomi, Amir and Murtaza Haider (2015). “Beyond the hype: Big data concepts, methods, and analytics”. In: *International journal of information management* 35.2, pp. 137–144.

Géron, Aurélien (2022). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O’Reilly Media, Inc."

Guo, Yike (2018). *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM.

Hagan, Martin T and Mohammad B Menhaj (1994). “Training feedforward networks with the Marquardt algorithm”. In: *IEEE transactions on Neural Networks* 5.6, pp. 989–993.

He, Kaiming et al. (2016). “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Kadam, Shivam S, Amol C Adamuthe, and Ashwini B Patil (2020). “CNN model for image classification on MNIST and fashion-MNIST dataset”. In: *Journal of scientific research* 64.2, pp. 374–384.

Khacef, Lyes, Nassim Abderrahmane, and Benoît Miramond (2018a). “Confronting machine-learning with neuroscience for neuromorphic architectures design”. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8.

Khacef, Lyes, Nassim Abderrahmane, and Benoît Miramond (2018b). “Confronting machine-learning with neuroscience for neuromorphic architectures design”. In: *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. DOI: 10.1109/IJCNN.2018.8489241.

Krizhevsky, Alex, Geoffrey Hinton, et al. (2009). “Learning multiple layers of features from tiny images”. In.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2017). “Imagenet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60.6, pp. 84–90.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). “Deep learning”. In: *nature* 521.7553, pp. 436–444.

LeCun, Yann et al. (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Lin, Jie et al. (2017). “A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications”. In: *IEEE internet of things journal* 4.5, pp. 1125–1142.

Qian, Ning (1999). “On the momentum term in gradient descent learning algorithms”. In: *Neural networks* 12.1, pp. 145–151.

Rajpurkar, Pranav et al. (2017). “Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning”. In: *arXiv preprint arXiv:1711.05225*.

Ratha, Nalini K., Jonathan H. Connell, and Ruud M. Bolle (2001). “Enhancing security and privacy in biometrics-based authentication systems”. In: *IBM systems Journal* 40.3, pp. 614–634.

Rosenblatt, Frank (1958). “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6, p. 386.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). “Learning representations by back-propagating errors”. In: *nature* 323.6088, pp. 533–536.

Russel, Stuart, Peter Norvig, et al. (2013). *Artificial intelligence: a modern approach*. Vol. 256. Pearson Education Limited London.

Simonyan, Karen and Andrew Zisserman (2014). “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556*.

Srivastava, Nitish et al. (2014). “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1, pp. 1929–1958.