# APPLICATION OF DEEP LEARNING FOR CLASSIFICATION OF TWO-DIMENSIONAL RADIOGRAPHS

A thesis submitted in partial fulfilment of
the requirements for the degree of
**Master of Engineering**
in
Electronics & Tele-Communication Engineering

by

**Indranil Bera**
bearing Class Roll Number: 002010702017 and Registration Number:
154086 of 2020-2021 under the guidance of

**Prof. Amit Konar**

to the

**Department of Electronics & Tele-Communication Engineering**
under
**Faculty of Engineering & Technology**
**Jadavpur University**
**Kolkata - 700032, West Bengal, India**
in
**June 2022**

# APPLICATION OF DEEP LEARNING FOR CLASSIFICATION OF TWO-DIMENSIONAL RADIOGRAPHS

A thesis submitted in partial fulfilment of
the requirements for the degree of
**Master of Engineering**
in
Electronics & Tele-Communication Engineering

by

**Indranil Bera**
bearing Class Roll Number: 002010702017 and
Registration Number: 154086 of 2020-2021 under the guidance of

**Prof. Amit Konar**

to the

**Department of Electronics & Tele-Communication Engineering**
under
**Faculty of Engineering & Technology**
**Jadavpur University**
**Kolkata - 700032, West Bengal, India**
in
**June 2022**

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**JADAVPUR UNIVERSITY**

RECOMMENDATION

DATE : __ / __ / 2022

I hereby recommend that the thesis prepared under my direct supervision by Mr. **INDRANIL BERA** (University Registration No.: 154086 of 2020-2021) entitled "**APPLICATION OF DEEP LEARNING FOR CLASSIFICATION OF TWO-DIMENSIONAL RADIOGRAPHS**" be accepted in partial fulfilment of the requirement for the award of the degree of Master of Engineering in Electronics & Tele-communication Engineering of Jadavpur University, Kolkata.

**Prof. Amit Konar**
Supervisor of the Thesis
Dept. of Electronics & Telecommunication Engineering
Jadavpur University

**Prof. Ananda Shankar Chowdhury**
Professor and Head of the Department
Dept. of Electronics & Telecommunication Engineering
Jadavpur University

**Prof. Chandan Mazumdar**
Dean of Faculty of Engineering and Technology
Jadavpur University

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**JADAVPUR UNIVERSITY**

## CERTIFICATE OF APPROVAL*

The foregoing thesis is hereby approved as a creditable study of Engineering subject and presented in a manner satisfactory to warrant its acceptance as a pre-requisite to obtain the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but approve the thesis only got the purpose for which it is submitted.

Committee on final examination for the evaluation of the thesis.

……………………………………

……………………………………

……………………………………
Signature of Examiners

*Only in the case the thesis is approved

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**JADAVPUR UNIVERSITY**

## DECLARATION OF ORIGINALITY AND COMPLIANCE OF THESIS

I hereby declare that this thesis entitled **"APPLICATION OF DEEP LEARNING FOR CLASSIFICATION OF TWO-DIMENSIONAL RADIOGRAPHS"** contains literature survey and original research work by the undersigned candidate, as part of his Degree of Master of Engineering in Electronics & Tele-communication Engineering. All information have been obtained and presented in accordance with academic rules and ethical conduct. It is hereby declared that, as required by these rules and conduct, all materials and results that are not original to this work have been properly cited and referenced.

Candidate Name: **Indranil Bera**

Registration Number: **154086 of 2020-2021**

Examination Roll Number: **M4ETC22017**

Thesis Title: **APPLICATION OF DEEP LEARNING FOR CLASSIFICATION OF TWO-DIMENSIONAL RADIOGRAPHS**

Date:

……………………………………

Signature of Candidate

DEDICATION

To the relentless advancement of
Science & Technology
and the teachers who make it possible.

## Table of Contents

# Acknowledgement

I would like to express my earnest gratitude and sincere thanks to my thesis supervisor Prof. Amit Konar, Department of Electronics and Tele-communication Engineering, Jadavpur university, for giving me opportunity to work under him and inspiring me to explore the interesting fields of Artificial Intelligence and Robotics. I am indebted to him for his patient, guidance, critical and constructive views and untiring support that shaped my work. The past year has been a remarkable experience in terms of gaining knowledge and skill that I hope to carry on and develop further.

I am extremely grateful to Prof. Ananda Shankar Chowdhury who has acted as Head of the Department of Electronics and Tele-communication Engineering, Jadavpur University during my Master of Engineering course for their valuable guidance and support. I convey my regards and respect to all Professors and associates in the Department of Electronics and Tele-communication Engineering for their help and support. I am thankful to the Department of Electronics and Tele-communication Engineering for the assistance with materials, equipment and subjects for conducting extensive experiments in the laboratories.

I would like to extend my sincere respect and thanks to all seniors who have provided constant support and guidance throughout my work especially. I consider myself fortunate to have worked with such friendly and motivating seniors who helped me in cultivating my knowledge and improving my skills.

Finally, I am immensely indebted to my parents, family and friends for their continuous support and encouragement that made me believe in myself and provided the strength to work hard.

Indranil Bera
Graduate Student
Dept. of Electronics & Tele-communication Engineering
Jadavpur University

# Abstract

Artificial intelligence (AI) has been a growing trend lately. One of the tasks which can be achieved by AI is computer vision, which is the ability for computers to process and analyse images, aiming to mimic human vision. One of the main tasks of computer vision is image classification, which is the process of labelling images into "classes". For example, if there are images of multiple objects, and those images need to be categorized into "classes", for instance "car", "plane", "ship", or "house", that is image classification.

One common way to execute image classification is through convolutional neural networks, a technique implementing deep learning, which is a subset of machine learning, which is in turn a subset of AI.

The objective of this thesis was to study the application of deep learning in image classification using convolutional neural networks. The Python programming language with the TensorFlow framework and Google Colaboratory hardware were used for the thesis. Models were chosen from available ones online and adjusted by the author. After the research, an accurate and performant model was developed and there is still room for further optimization.

Keywords: image classification, convolutional neural network, deep learning, deep learning with python

## List of figures:

## Table of abbreviations:

| | |
|---|---|
| AI | Artificial Intelligence |
| ML | Machine Learning |
| RL | Reinforcement Learning |
| CART | Classification And Regression Trees |
| DL | Deep Learning |
| ANN | Artificial Neural Network |
| CNN | Convolutional Neural Network |
| DNN | Deep Neural Network |
| TF | TensorFlow |
| TFLite | TensorFlow Lite |
| mAP | mean Average Precision |
| CRNN | Convolutional Recurrent Neural Network |
| ConvNet | Convolutional Neural Network |
| SVM | Support Vector Machine |
| CV | Computer Vision |
| LDA | Linear Discriminant Analysis |
| QDA | Quadratic Discriminant Analysis |
| LR | Learning Rate |
| Sklearn | Scikit-learn |
| PCA | Principal Component Analysis |
| RLFM | Regression Based Latent Factors |
| SVD | Singular Value Decomposition |
| ReLU | Rectified Linear Unit |
| GAN | Generative Adversarial Network |
| ELU | Exponential Linear Unit |
| CMMs | Convolutional Markov Models |
| MAP | Maximum A Posteriori |
| MLE | Maximum Likelihood Estimation |

# Chapter 1

# Artificial Intelligence & Machine Learning

## 1.1 Introduction

It is hard to find nowadays someone that didn‘t hear in some way or another about Artificial Intelligence and doesn‘t have an idea about what it is. So, how it is related with Machine Learning?

As Bernard (2016, p.1) puts it - Artificial Intelligence is the broader concept of machines being able to carry out tasks in a way that it would be considered smart. And, Machine Learning is a current application of AI based around the idea that machines should be able to have access to data and learn for themselves.

In this chapter, the concept of Machine Learning, how it actually works, and the algorithms implemented in this project, will be explained extensively.

## 1.2 Types of ML

According to what goal to be achieved by using ML, it can be classified into four major types as follows:

### 1.2.1 Supervised learning

Supervised learning earned its name because data scientists acts as a guide to teach the algorithm what conclusions it should come up with. It is similar to the way a student learns basic arithmetic from a teacher. This type of learning requires labeled data with the correct answers to be expected from the algorithm‘s output. For classification and regression problems Supervised learning proved itself to be accurate and fast (cited in Castle 2017, p.1).

**Classification:** It consists of predicting the categorical output value where the data can be separated into specific classes. Classification has different use cases, such as: determining the weather, if an email is a spam or not or types of animals after being trained on a properly labeled dataset of images with the species and some identifying characteristics (cited in Sanjeevi 2017, p.2).

**Regression:** It‘s a type of problem where the prediction of a continuous-response value such as stock and housing prices is needed (cited in Sanjeevi 2017, p.3).
So, the way it works is modeling relationships and dependencies between the target prediction output and the input features such that it is possible to predict the output values for new data based on those relationships which it learned from the previous datasets (cited in Fumo 2017, p.2).

### 1.2.2 Unsupervised Learning

Conversely, unsupervised learning is more closely aligned with what it is called true artificial intelligence by some experts - the concept that a machine can learn to identify complex processes and patterns without supervision from humans. This approach is particularly useful in cases where the experts doesn‘t know what to look for in the data and the data itself does not include Targets. Under the many use cases of unsupervised machine learning it‘s worth mentioning k-means clustering, principal and independent component analysis, and association rules. (cited in Castle 2017, p.2)

**K-means clustering:** It‘s a type of a problem where similar things are grouped together. It shares the same concept with classification but in this case, there are no labels provided and the system will understand from the data itself and cluster it. A use case for this would be clustering news, articles depending on their genre, content. (cited in Trevino 2016)
Despite This type of machine learning opens the doors to solving problems that human normally would not tackle, it‘s not used as widely as the supervised learning due to its complexity and difficulty to implement. (cited in Castle 2017, p.2)

### 1.2.3 Semi-supervised Learning

Until now, the data provided is all labeled with the desired output or not labeled at all. Semi-supervised machine learning is a combination of the two. In many practical situations, the cost to label is quite high and in case of large datasets the task become tedious and very much time consuming. In addition, providing too much labeled data, can force human biases on the model. Even though the unlabelled data is unknown for the network, this data brings useful information about the target group parameters. Which leads to the conclusion, that by including unlabelled data the accuracy of the model can be improved while also saving time and money building it.

For example, semi-supervised machine learning could be used in webpage classification, voice recognition or genetic sequencing. In those cases, data scientists can access large volumes of unlabelled data, and the task of labeling all of it would take an overwhelming time. (cited in Castle 2018, p.1-2)

Using the information acquired until now a comparison between these three types of machine learning can be set for the same use case, for example classification:

**Supervised classification:** The algorithm will classify the types of the webpages according to the labels provided from the beginning. (cited in Castle 2018, p.2)

**Unsupervised clustering:** The algorithm will look for patterns and characteristics that help placing webpages into groups. (cited in Castle 2018, p.2)

**Semi unsupervised classification:** The algorithm will identify the different groups of webpages based on the labeled data and will use the unlabeled data to define the boundaries of those webpage types and to look for other types that might not be listed in the labeled data. (cited in Castle 2018, p.2)

### 1.2.4 Reinforcement Learning

Reinforcement Learning is the third main Machine Learning type along with Supervised and Unsupervised Learning. It consists of five important components which are: the agent, environment, state, action and reward. The goal of RL is to maximize the reward and minimize the risk by exploiting its interaction with the environment. The RL algorithm (called the agent) will periodically improve by exploring the environment going through the different possible states. To maximize the performance, the ideal behavior will be automatically determined by the agents. A feedback (the reward) is what allows the agent to improve its behavior. (cited in Fumo 2017, p.4)

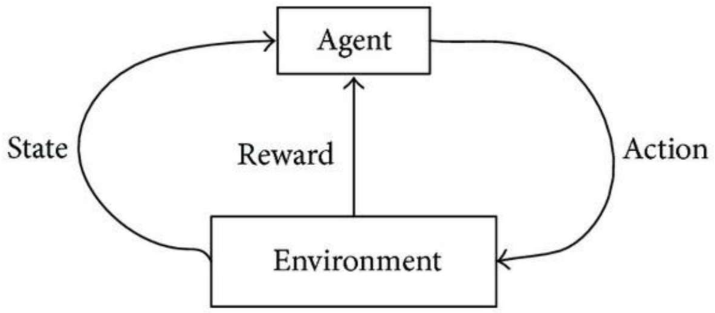


**Figure 1.1: Reinforcement Learning Components (Fumo 2017, p.4)**

To obtain agents with good results, reinforcement machine learning goes through five main steps. Fumo (cited in 2017, p.5) describe them in his article as follows:

- The agent examines constantly the input state.
- The agent performs an action according to the function responsible for decision making.
- The agent will receive reinforcement (reward) after performing its action.
- Information about the reward state will be stored.

In Reinforcement Learning there are two types of tasks: episodic and continuous:
**Episodic task:** The task in this case is defined by a starting and an ending point or also called a terminal state. This creates an episode: a list of states, actions, rewards, and new states. Video games are a typical example of this type of tasks. (cited in Simonini 2018, p.7)

**Continuous task:** Opposite to the first type, this one has no terminal state and as its name indicates, continue forever. In this case the agent has to learn how to choose the best actions and simultaneously interacts with the environment. Automated stock trading is a typical use case of this type of tasks. The agent keeps doing actions and receiving feedback until it‘s decided to be stopped, since there is no starting point and terminal state. (cited in Simonini 2018, p.8) One of the most used algorithms for Reinforcement Learning is **Monte Carlo** which is based on collecting the rewards at the end of the episode and then calculating the maximum expected future reward. A second popular algorithm is **Temporal Difference Learning** that uses a different approach from the first one which is estimating the rewards at each step. (cited in Simonini 2018, p.8)

## 1.3 Techniques

Since the beginning of the AI implementation, many techniques were used, and many others are emerging until this day. In this subchapter, three different techniques will be discussed and ordered by their introducing date to the public.

### 1.3.1 SVM

Support Vector Machine (SVM) is a supervised machine learning technique which tackles mainly regression and classification challenges. In case of classification, each data item is plotted as points in n-dimensional space (where **n** represents the number of available features) with the value of each feature being the value of a particular coordinate. Afterwards, by classifying the different classes, a hyper-plane will be plotted to separate them.

As Ray (2017, p.3) clarify it - Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

There are several scenarios that can be stumbled on while trying to apply SVM and it can deal with them perfectly to identify the right hype-plane.

- A, B and C represents three hyper-planes. The one which segregates the two classes better will be selected. As the figure shows, B is the appropriate choice. (cited in Ray 2017, p.4)

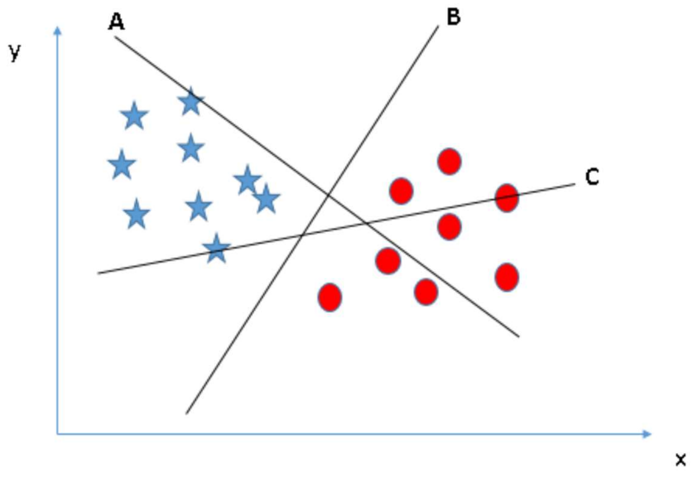


Figure 1.2: First classification scenario with SVM (Ray 2017, p.4)

- In this case all three hyper-planes are segregating the classes well. To decide which one from the three is the right one, the distances between the nearest data point and the hyper-plane should be maximized. This distance is called Margin. Another reason for choosing the hyper-plane with the higher margin is robustness, otherwise a misclassification has a high chance to occur when choosing a hyper-plane with a low margin. (cited in Ray 2017, p.4-5)

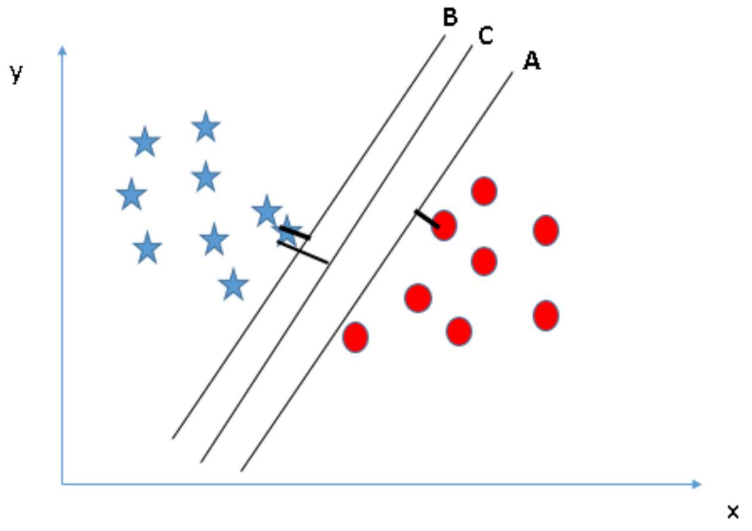


Figure 1.3: Second classification scenario with SVM (Ray 2017, p.4)

In this scenario, applying the same logic as the previous scenario won't give a correct classification since B has the higher margin and as the below figure demonstrate, A should be the right choice. Here, the SVM technique will be aware of the situation and won't prioritize the margin maximization over classifying correctly the two classes. (cited in Ray 2017, p.5)

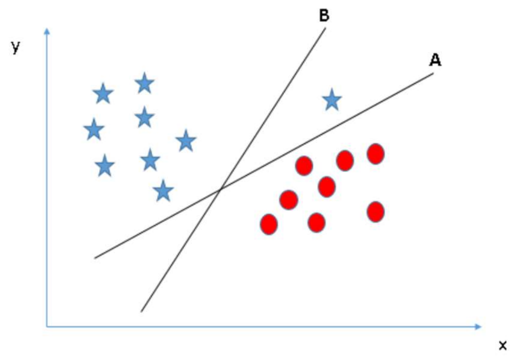


Figure 1.4: Third classification scenario with SVM (Ray 2017, p.5)

- In this case segregating the two classes is not possible since one of star class lies in the territory of the other class as an outlier. Luckily SVM robustness will prevent choosing the wrong hyper-plane by ignoring any possible outliers (cited in Ray 2017, p.5).

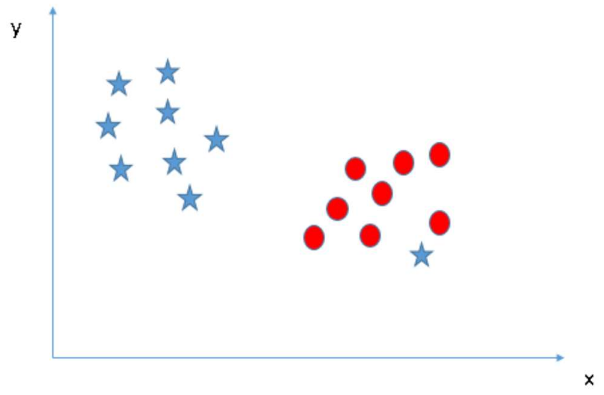


Figure 1.5: Fourth classification scenario with SVM (Ray 2017, p.5)

- Here the two classes can‘t be directly separated with a linear hyper-plane, that‘s why SVM introduces a new additional feature which is: $z = x^2 + y^2$ to properly separate the two classes. (cited in Ray 2017, p.6)

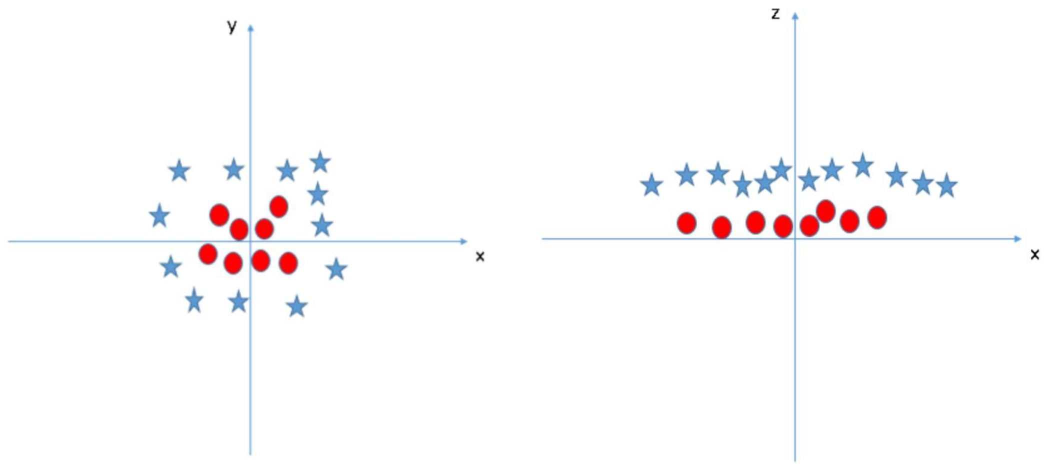


Figure 1.6: Fifth classification scenario with SVM (Ray 2017, p.6-7)

Having a linear hyper-plane between these two classes is an easy task for SVM. But should the additional feature be added manually as done in the last scenario to have a hyper-plane? It is done automatically by an SVM technique called **kernel trick**. The kernels are functions that take data which is not linearly separable in a low dimensional space and transform it in a higher dimensional space where it can be linearly separable. It is mostly useful in non-linear separation problem. Said otherwise, based on the labels or defined outputs, it will do some extremely complex data transformations to figure out the process to separate the data. (cited in Ray 2017, p.7)

The Scikit-learn developers (2017a, p.1) lists several advantages and disadvantages of SVM. They are as follows:

- The advantages of support vector machines are:
  - Effective in high dimensional spaces.
  - Still effective in cases where number of dimensions is greater than the number of samples.
  - Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
  - Versatile: different kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.
- The disadvantages of support vector machines include:
  - If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
  - SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

### 1.3.2 Random Forest

To better understand the Random Forest technique, an explanation of what a decision tree is needed.

#### 1.3.2.1 Decision Tree

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions (Gupta 2017a, p.1). Random Forest is widely used to solve classification and regression problems in machine learning and it‘s also a commonly used tool in data mining to achieve a particular goal by deriving strategies (cited in Gupta 2017a, p.1).

The algorithm is represented as an upside-down drawn tree. In the figure below, the tree splits every time there is a condition (internal node) which are represented with the bold text in black. The outputted decision is called a branch (edge). In case a branch reached its limit and can‘t be divided anymore, it is identified as a decision (leaf). As shown in the next figure, the leaves are in red and green and represent whether a passenger from the titanic died or survived. (cited in Gupta 2017a, p.2)

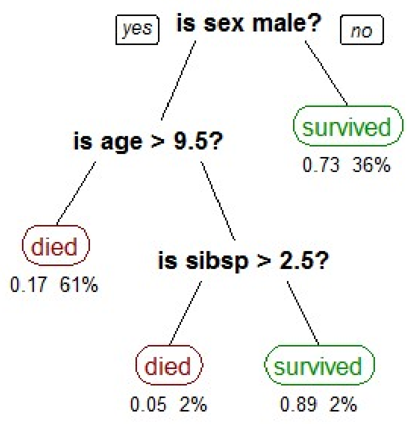


Figure 1.7: Titanic decision tree (cited in Gupta 2017a, p.2)

The main reason why this algorithm is widely used is its simplicity and how the feature importance and the relations in the tree can be easily represented. The above figure represents an example of a **classification tree** since its objective is predicting and classifying data of the titanic passengers into different classes (died or survived). When in the other hand, a **regression tree** predicts an output depending on a continuous progressing data. Decision Tree is mostly referred to as **CART** (**C**lassification **A**nd **R**egression **T**rees). (cited in Gupta 2017a, p.2) Finally, Gupta lists in his post (2017a, p.5) the different advantages and disadvantages of CART:

- Advantages of CART
  - Simple to understand, interpret, visualize.
  - Decision Trees implicitly perform variable screening or feature selection
  - Can handle both numerical and categorical data. Can also handle multioutput problems.
  - It requires relatively little effort from users for data preparation.
  - Nonlinear relationships between parameters do not affect tree performance.
- Disadvantages of CART
  - Decision-tree learners can create over-complex trees that do not generalize the data well, which will result in
  - overfitting.
  - Decision trees can be unstable because small variations in the data might result in completely different tree being generated. This is called variance, which needs to be lowered by methods like bagging and boosting.
  - Greedy algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training

- multiple trees, where the features and samples are randomly sampled with replacement.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the data set prior to fitting with the decision tree.

#### 1.3.2.2 Random Forest

This technique falls under the supervised machine learning category. As the name indicates, a forest will be created from a group of decision trees and then will be randomized. Different methods can be used to train the random forest, and the mostly used one is Bootstrap Aggregation (Bagging) method. (cited in Donges 2018, p.2)
As Brownlee describes it in his article (2016a, p.3) the Bagging method is a very powerful ensemble method and goes further with his explanation - an ensemble method is a technique that combines the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model. As the Random Forest consists of multiple Decision Trees, it‘s used also for the same purposes: Regression and Classification. Not only that, it shares also almost the same hyperparameters as a decision tree. Fortunately, a decision tree doesn‘t have to be combined with a bagging classifier and the classifier-class of Random Forest can be used here. Furthermore, Regression problems can be solved with using Random Forest regressor. During the process of training, the Forest randomize the model. It also seeks the best feature among a random subset of features while creating the nodes of the tree instead of looking for the most important feature. This behavior will optimize the model since the randomness will result in a wide diversity.
This randomness can be also added to other aspects of the training, such as randomizing the thresholds used for each feature. Which oppose to the traditional decision tree method, where the model will look for the most fitting threshold. (cited in Donges 2018, p.3)

#### 1.3.2.3 Feature Importance

Random Forests makes the measurement of the relative importance of each feature on the prediction very easy (cited in Donges 2018, p.4). By using the methods provided by the class **Sklearn** from the machine learning tool **scikit-learn** (cited in scikit-learn developers 2017b, p.1), the features importance can be evaluated and measured to discover any impurity that may exists in the forest. The output of each feature will be scaled in a way that the sum of all importance is equal to 1. (cited in Donges 2018, p.4)
Since some features doesn‘t play a role in the prediction process, they can be dropped and determined by looking through at the feature importance. This step is almost necessary, since

the more features taken under consideration in machine learning the more likely the model will be overfitted. (cited in Donges 2018, p.4)

Despite that Random Forests are based on Decision Trees, there are some differences between them:

If a decision tree is provided by a training dataset with features and labels, it will formulate some set of rules, which will be used to make the predictions.

To better understand the concept, a real-life example such as targeted advertisements, can facilitate explaining it. By collecting the advertisements, a user clicked on in a period of time and features that describe his decision, a model can be trained to predict whether that user will visit a certain advertisement site or not. When the features and labels are fed to a decision tree, it will generate some rules. Then a prediction can be made whether the advertisement will be clicked or not. In the other hand, a model trained with Random Forest will build several decision trees based on random observations and features. Not to mention that decision trees are exposed to overfitting, while random forest mostly avoid this problem by creating random subsets of the features and building smaller trees using these subsets. Finally, those subtrees will be combined (cited in Donges 2018, p.5). However, this technique is a double-edged sword, since it slows down computation in case the forest has a large number of trees. Which leads in some cases such as real-time prediction to avoid implementing Random Forest and look for an alternative. (cited in Donges 2018, p.6-7)

As a conclusion, Random Forest are more suitable for use cases that doesn‘t require big datasets and detection time don‘t play a big role in the application to avoid any possible complications. For more complex tasks it is preferable to implement another approach. (cited in Donges 2018, p.7)

## Deep Learning:

Since Artificial Intelligence came a while ago, it has a wide range of applications and it‘s divided into many branches (cited in Le 2017, p.10; Goodfellow et al. 2016, p.1). Deep Learning is a subset of machine learning, which is in itself a subfield of AI. The figure below is a visual representation of the relationship between AI, ML and DL (cited in Le 2017, p.10).

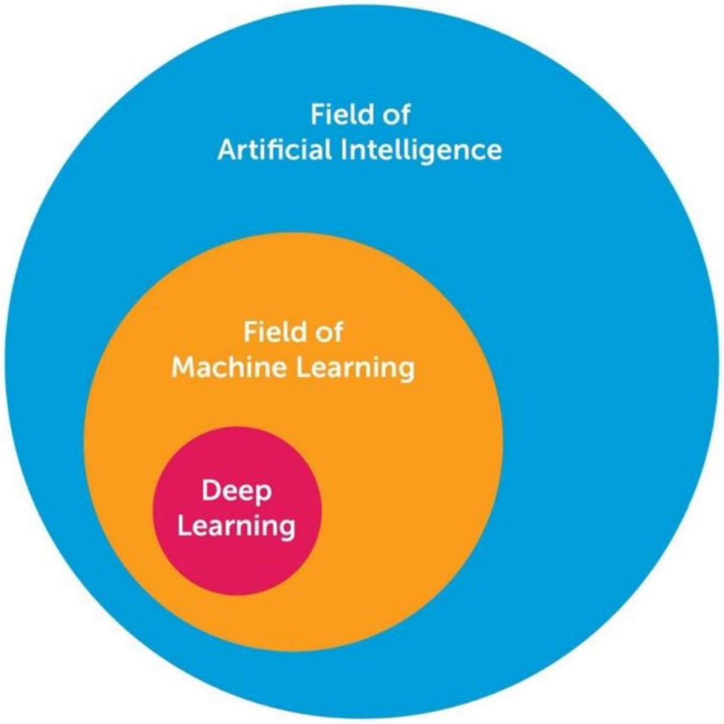


Figure 1.8: Relationship between AI, ML and DL (Le 2017, p.10)

So, what is exactly deep learning and what kind of problems it solves? This question will be answered in depth in a later chapter.

### 1.3.3 Transfer Learning

To train a model following the traditional supervised learning approach to perform a certain task and domain **A**, it is necessary to provide labeled data that matches the same task and domain. The Figure 1.9 is a visual representation of how the data should respect the task to be performed. Briefly explained, a task is the expected behavior from the model after the training process is done and a domain is simply the environment where the data was taken. (cited in Ruder 2017, p.3)

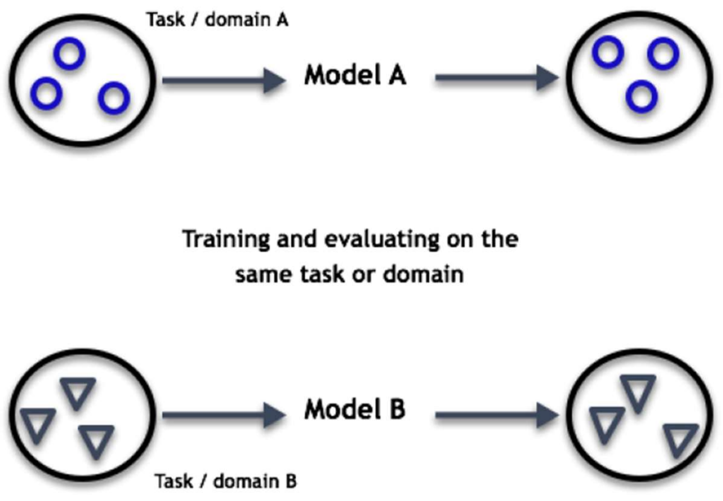


Figure 1.9: The traditional supervised learning setup in ML (Ruder 2017, p.3)

After training the model A on its dataset, it will function correctly and as expected when testing it on data of the same task and domain. In the other hand, and as the Figure 1.9 illustrates, a new model **B** should be trained if a new data for a different task or domain **B** is provided. (cited in Ruder 2017, p.4)

Since many tasks happen to be similar, it only makes sense to try to reuse data for similar situations. In case of using traditional supervised learning, a problem arises, which is deterioration or collapse in performance of the model. For instance, a model that has been trained to detect pedestrians during the day time can theoretically be used to train a model to detect pedestrians during the night time. In practice this will cause inaccuracy since the model is fitted to its dataset and don‘t know how to generalize to the new domain. This issue is more problematic when trying to train a model to detect bicyclists, since the existing model can‘t even be reused because of difference in labels. (cited in Ruder 2017, p.4)

To overcome this problem the concept of Transfer learning can be used by leveraging the already existing data of some related task or domain. It is possible then to store this knowledge gained in solving the source task in the source domain and apply it to the problem of interest as can be seen in Figure 1.10. (cited in Ruder 2017, p.4)

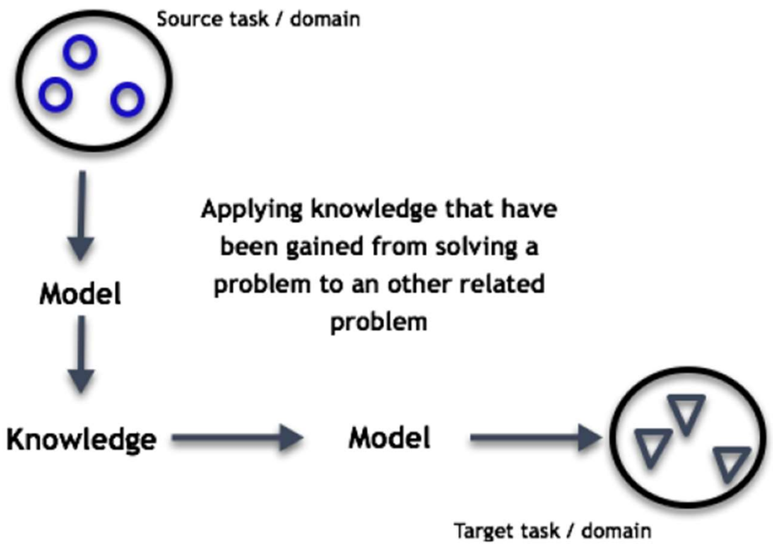

Source task / domain
Applying knowledge that have been gained from solving a problem to an other related problem
Model
Knowledge
Model
Target task / domain

Figure 1.10: The Transfer Learning Setup (Ruder 2017, p.4)

Researches seek often to take advantage of the existing data and trained models as much as possible to transfer it to a new domain for similar tasks. There are different aspects that can be of interest and taken under consideration when the transfer learning is going to be made. For instance, researches can be interested in how objects are shaped in the data, because they have certain shape that will help define the new objects, or in case of speech recognition, the context and the voice patterns are the main focus. (cited in Ruder 2017, p.5)
In the following paragraph the two terms 'domain' and 'task' will be explained in depth with an example given by Ruder in his article (2017, p.9)

Transfer learning involves the concepts of a domain and a task. A domain **D** consists of a feature space $\mathcal{X}$ and a marginal probability distribution **P(X)** over the feature space, where $\mathbf{X} = \mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathcal{X}$. For document classification with a bag-of-words representation, $\mathcal{X}$ is the space of all document representations, $\mathbf{x}_i$ is the ***i*-th** term vector corresponding to some document and **X** is the sample of documents used for training.
Given a domain, $\mathbf{D} = \{\mathcal{X}, \mathbf{P(X)}\}$, a task $\mathcal{T}$ consists of a label space $\mathcal{Y}$ and a conditional probability distribution $\mathbf{P(Y|X)}$ that is typically learned from the training data consisting of pairs $\mathbf{x}_i \in \mathbf{X}$ and $\mathbf{y}_i \in \mathcal{Y}$. In this example, $\mathcal{Y}$ is the set of all labels, i.e., *True*, *False* and $\mathbf{y}_i$ is either *True* or *False*.
Given a source domain $\mathbf{D}_S$, a corresponding source task $\mathcal{T}_S$, as well as a target domain $\mathbf{D}_T$ and a target task $\mathcal{T}_T$, the objective of transfer learning now is to enable researchers to learn the target conditional probability distribution $\mathbf{P}(\mathbf{Y}_T|\mathbf{X}_T)$ in $\mathbf{D}_T$ with the information gained from

$\mathbf{D}_S$ and $\mathcal{T}_S$ where $\mathbf{D}_S \neq \mathbf{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$. In most cases, a limited number of labeled target examples, which is exponentially smaller than the number of labeled source examples are assumed to be available.
Four different scenarios can be deducted considering the structure of **D** and **T** and the inequalities expressed in the example above. Ruder goes further and explains in theory these scenarios using the previous use case. (2017, p. 9-10).

Given source and target domains $\mathbf{D}_S$ and $\mathbf{D}_T$ where $\mathbf{D} = \{\mathcal{X}, \mathrm{P}(\mathrm{X})\}$ and source and target tasks $\mathcal{T}_S$ and $\mathcal{T}_T$ where $\mathcal{T} = \{\mathcal{Y}, \mathrm{P}(\mathrm{Y}|\mathrm{X})\}$ source and target conditions can vary in four ways.

- $\mathcal{X}_S \neq \mathcal{X}_T$ **:** The feature spaces of the source and target domain are different, e.g., the documents are written in two different languages. In the context of natural language processing, this is generally referred to as cross-lingual adaptation.
- $\mathrm{P}(\mathrm{X}_S) \neq \mathrm{P}(\mathrm{X}_T)$ **:** The marginal probability distributions of source and target domain are different, e.g., the documents discuss different topics. This scenario is generally known as domain adaptation.
- $\mathcal{Y}_S \neq \mathcal{Y}_T$ **:** The label spaces between the two tasks are different, e.g., documents need to be assigned different labels in the target task. In practice, this scenario usually occurs with scenario 4, as it is extremely rare for two different tasks to have different label spaces, but exactly the same conditional probability distributions.
- $\mathrm{P}(\mathrm{Y}_S|\mathrm{X}_S) \neq \mathrm{P}(\mathrm{Y}_T|\mathrm{X}_T)$ **:** The conditional probability distributions of the source and target tasks are different, e.g., source and target documents are unbalanced with regard to their classes. This scenario is quite common in practice and approaches such as over-sampling, under-sampling.

After discussing the concepts relevant for transfer learning in theory and the scenarios in which it is applied, the practical side of transfer learning will be covered in the next section by going through different applications that illustrate some of its potential.

#### 1.3.3.1 Applications of Transfer Learning

**Learning from simulations:** When considering training a network to perform a certain task, the most important factor is the dataset. In most cases, to get good results a large number of data must be provided, but in some scenarios the number of samples is just enormous. This is due to its critical need to be as accurate as possible and this can result to incapacity to provide the needed amount of data because of the tedious work behind it and the expenses it will cost. That is why researcheres approached this challenge in a creative way and leveraged transfer learning and simulations to come up with a solution. The reason behind using simulation is quite simple. Engines can run seamlessly and without the need of constant human interaction. This technique can provide a huge amount of data in a short period of time that can be later on transferred to the desired domain. Once concern may arise considering this approach, which is using simulations can never be as good as depending on real-world raw data. Nonetheless, with the advancements of today technology, this gap is getting smaller. Finally, learning from simulations is a typical example of the second scenario mentioned in the previous section. This technique is often used to get data for self-driven cars and to mimic robotics behavior. Such use cases are much more sensitive to errors than the others. That‘s why they need as much data as possible. Thus, using simulations to solve this problem. (cited in Ruder 2017, p.11-12)

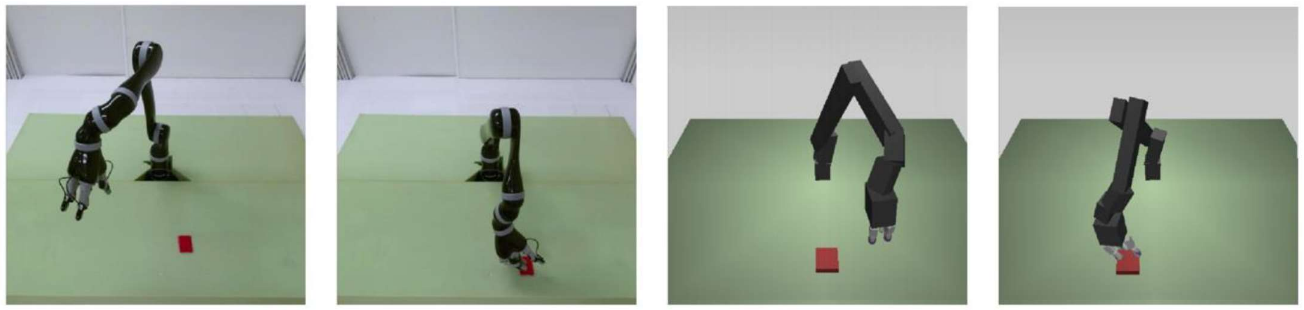


**Figure 1.11: Robot and simulation environments (Ruder 2017, p.12)**

**Transfer learning in Computer Vision:** Solving problems in the Computer Vision field was a complex task since it revolves mainly around detecting features that represent the data provided. This procedure was done manually until Deep Learning methods were introduced. Features are now detected automatically using Deep Learning during the training process. That is why only raw images are needed as an input to train a model. This concept proved its success and effectiveness in the Computer Vision world in many scenarios and it changed also the main challenge when solving a problem from figuring out the features manually to choosing the right architectures that fits the appropriate task in hand and its data. Finally, Deep Learning introduced the concept of layers and each layer is responsible for learning certain type of features. This hierarchical feature representation is the strongest asset of Deep Learning architectures since it makes them reusable and best suitable for transfer learning. (cited in Hulstaert 2018, p.7-8)

## 1.4 Recapitulation

In summary, the first section of this chapter capsuled a variety of information about artificial intelligence and machine learning, from the basics such as the connection between the two and their purposes, to more detailed information such as different types and techniques used in machine learning.

The second part of this chapter handled deep learning and neural networks, in particular convolutional neural networks in which this project revolves around. The following two figures are a visual representation of the most points explained in this section.

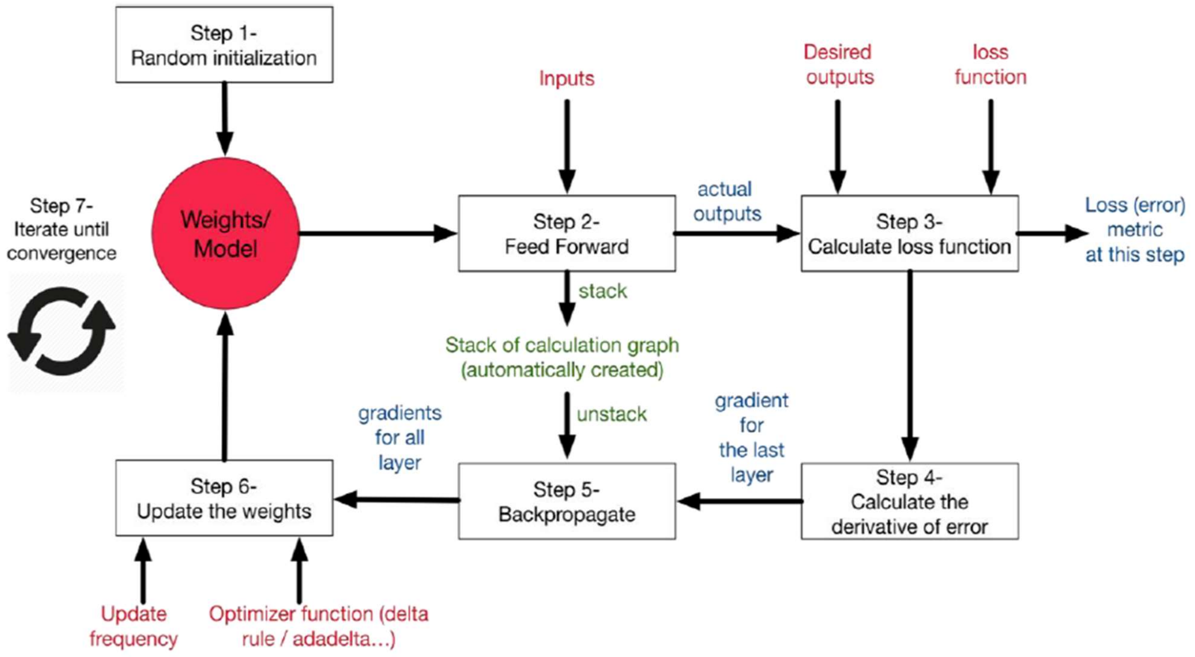


Figure 1.12: Learning process of a Neural Network (Moawad 2018, p.14)

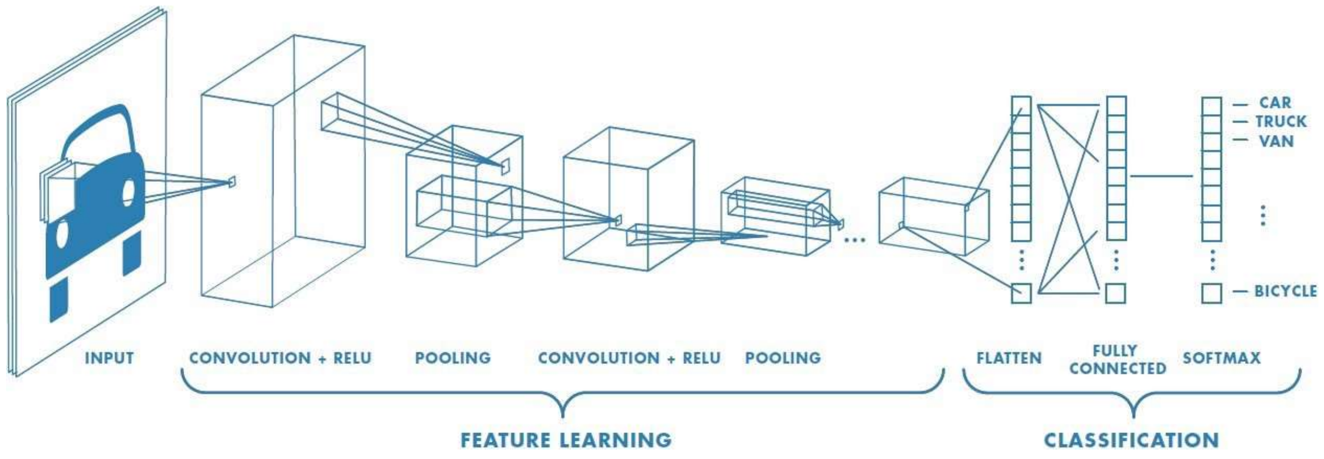


Figure 1.13: Convolutional Neural Network architecture (Mathworks n.d., p.4)

# Chapter 2

*"Very simple. Just keep adding layers until the test error does not improve anymore."*

# Neural Networks: a short introduction

## 2.1 Brain model

The human brain is composed of connected structures that can interact with each other, transmit the information and produce a precise reaction to a particular input. These structures are the neurons, electrically excitable cells connected by synapses and axons. The main function of these kind of cells is to receive a signal in response to a well-defined stimulus and then forward it, after some processing, to the next cells, which are connected to them. The signal is an electrical/chemical process which is generated by any internal or external interaction of the body. The final result of this propagation process is an interpretation of the stimulus or a reaction to it. Recently, this connection concept has been proposed to build a learning model for computers.

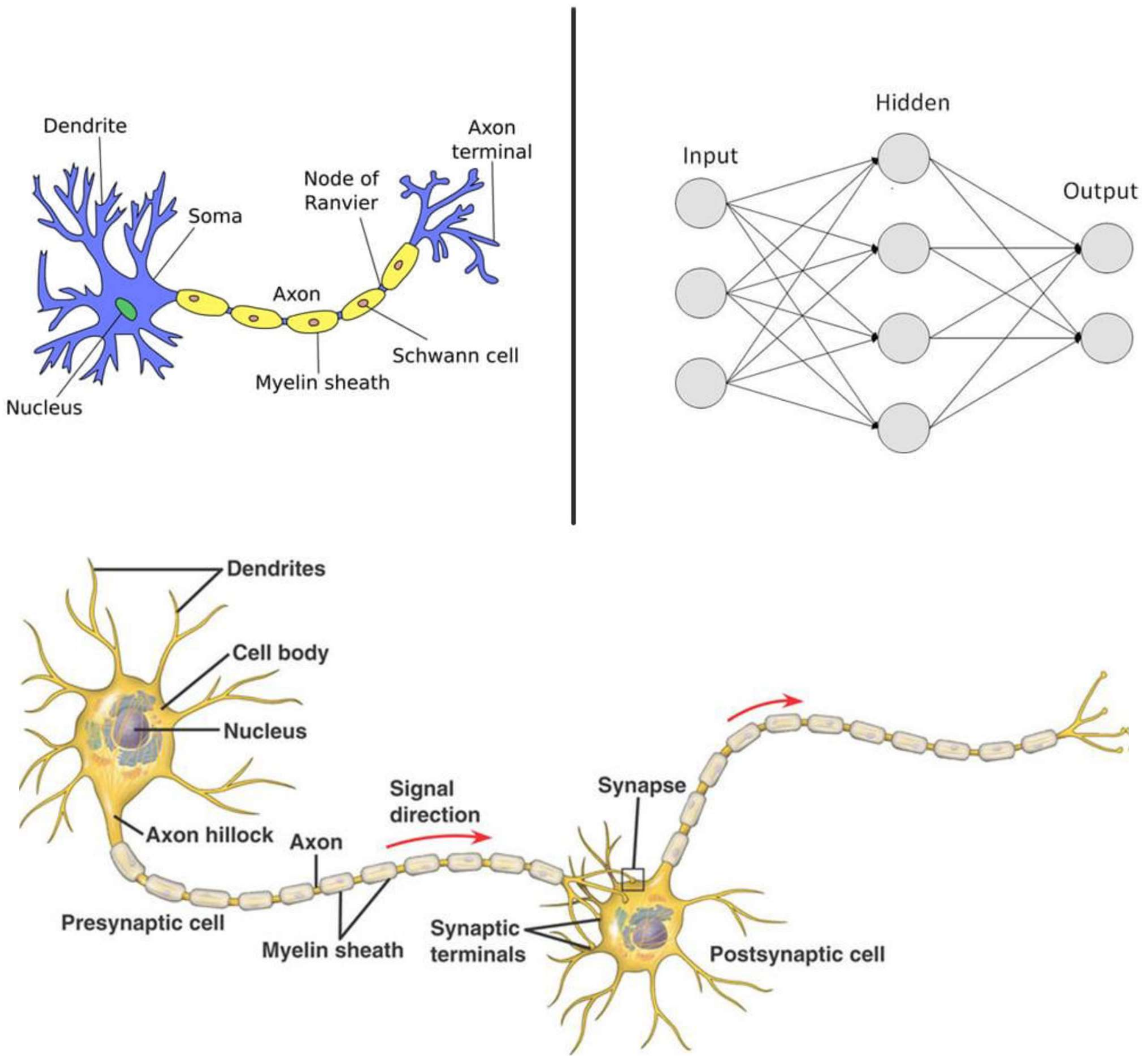


Figure 2.1: Comparison between biological neurons and artificial neurons

## 2.2 Neuron

An artificial neuron is a mathematical function which takes one or more inputs and combines them to create an output. Interconnecting these neurons makes possible to create a network, which is able, after a learning phase where all the variables of the network are set, to generate an output which is coherent with the given task. The visual comparison between biological neuron and artificial neuron is visible in figure 2.1.

Each artificial neuron implements the function:

$$y_k = \phi(\sum_{j=0}^{m} w_{kj} x_j)$$

……………………………..(2.1)

where $y_k$ is the output of the k-th neuron, $x_j$ is the value of the j-th input, $w_{kj}$ is the weight of the edge connecting input j to neuron k and $\varnothing$ is a function called *activation function*. A single unit of these has a limited learning ability, since it can implement a simple linear decision function. Their concatenation, instead, can perform very complex tasks.

The main components of equation 2.1 are:

• **Values:** These are the inputs given to the model

• **Weights:** They control how much each input affects the output value

• **Activation function:** The sum of all the input values multiplied by their corresponding weight is then processed by this non-linear function. The final result is the actual output of the neuron. Since this function defines the type of output obtained, there are many of them, all suitable for different problems;
These are some of the most used activation functions:
• **Step:** The output of this function is a binary value, which depends on a given threshold $\theta$.

$$f(x) = \begin{cases} 0 & x < \theta \\ 1 & x \geq \theta \end{cases}$$

…………………………..(2.2)

• **Sigmoid:** It is a simple non-linear function, whose derivative can be easily computed.

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

………………………(2.3)

The function is "S"-shaped, and the output is between 0 and 1. While the algorithm is in the training phase, the activation function needs to be derived several times, and therefore, the use of sigmoid can lighten the computation;
• **Rectifier:** This non-linear function returns the positive part of the input argument:

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

……………………….(2.4)

It has been shown that deeper networks can be trained more effectively by exploiting this function in their layers

• **Softmax:** this function is a way to convert a k-dimensional vector into another k dimensional vector where each of its value is between 0 and 1, and their sum is 1. The
first objective of this function is to provide a likelihood value for each output value
instead of a hard value.

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

……………………………(2.5)

## 2.3 Neural network

A neural network is a system where several of these neurons are interconnected to form a single structure. The neurons are typically organized into layers, where the "input layer" represent the data to be processed and the output layer is the result of the processing. All the other layers are called "hidden layers"; the number of layers identifies the depth of the network.
Each layer is connected to the next one by a certain number of edges, and the connection pattern affects the behaviour of the entire network. The most common pattern is the "fully connected", where each neuron in the layer i is connected with each neuron at layer i + 1; it is important to notice that each edge has its weight, which is a parameter that will be adjusted during the training phase. A greater number of edges will cause a greater number of parameters
to train and the performance of the entire network can slow down. Usually, there's one more parameter in each neuron, called bias: the idea behind the bias is to add in every linear combination describing the output of the given neuron a value that it does not depend on previous neurons, but it's constant; since the bias value is fixed, the only parameter to adjust is its weight.
By connecting a layer with l neurons and a bias to another layer of n neurons, the parameters concerning this particular part of the network will be (l+1)*n, so the complexity grows easily.

## 2.4 Training

This is the main phase for the creation of the model. It uses a training set, i.e., a set of data where each sample is made by the data in itself and a ground truth value, which is the information that the model will need to predict. The ground truth will be used to understand how much the current prediction done on its relative sample of data is far from the expected value, helping the network to learn from its errors. The phase of training is primarily divided into two distinct moments: forward propagation and backward propagation.

• **Forward propagation:** The input data from the training set are fed to the network; it is then propagated to the end and an output is produced. This output is compared to the ground truth, generating an error defined by a disparity or loss function.

• **Backward propagation:** The error is then propagated in the opposite direction in order to correct the weight values and refine the final accuracy.
This procedure is repeated for the whole training set, trying to adjust the parameters of the model for every data in it.

### 2.4.1 Learning

The learning phase adjusts the weights of the model. At the beginning, the weighs are initialised to an arbitrary value, say 1. Then the dataset is split into training, validation and test set. The actual training comes when the training set is processed by the network, and its output is compared to the ground truth of the data. An analysis on the error generated in this comparison will lead to an adjustment of the weights of every edge to improve its accuracy. The error measured by the loss function, which is a metric that parameterize the distance between the network output and the ground truth value. Some of the most-widely used metrics are:

• **Mean Square Error (MSE):** It is the average of the squared difference between predictions and values obtained. With this function, outliers (observations that are very far from their actual values) are penalized, because their big difference from the truth is squared.

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \quad \text{.....................(2.6)}$$

• **Mean Absolute Error:** It is very similar to the MSE, but the difference between observation and the ground truth depends on the absolute value. This function is slightly more complicated than the MSE while calculating its gradient, and the outliers are not so penalized.

$$MAE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n} \quad \text{......................(2.7)}$$

• **Mean Bias Error:** This function is not so common, and it is very similar to the MAE.

$$MBE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{n} \quad \text{........................(2.8)}$$

There could be both positive and negative errors, the MBE is not very accurate, but it

is useful to describe the trend of the training, if the bias is positive or negative.
Loss functions can, of course, be customized according to the problem analysed: further in this work an example of custom loss function will be used and explained.
There are also some parameters, called *hyperparameters*, that cannot be trained, but are defined at the beginning of the training phase and they can determine how well the entire process will perform.

• **Number of hidden layers:** If there are many hidden layers in the network, then the accuracy can increase, but the computation can be slower. On the other hand, if there are few hidden layers, the model will not have enough parameters to train and get a good approximation of the data, so the training will fail. In other words, it will underfit. Figure 2.2 shows an example of neural network with more than one hidden layer.

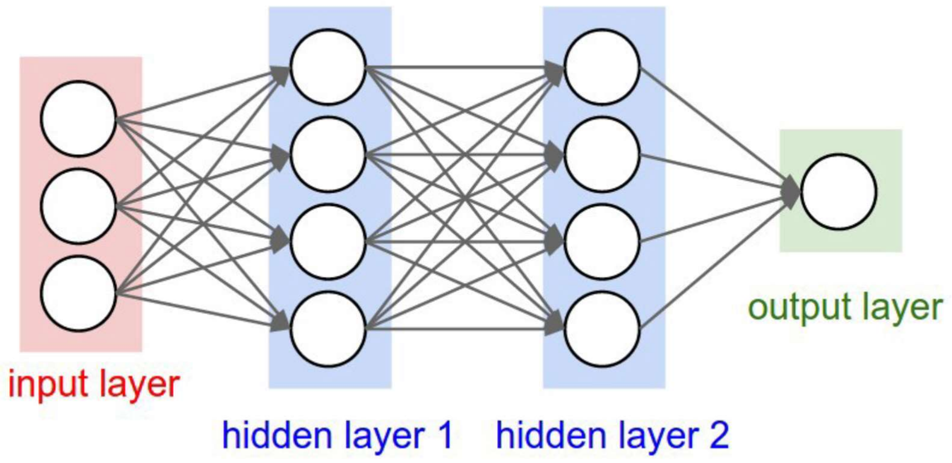


**Figure 2.2: An example of neural network with more than a single hidden layer**

• **Dropout:** It is used to avoid overfitting. It consists in ignoring a fixed number of neurons from each layer while training. This will keep the processing power of the network intact while avoiding overfitting it on a specific training set. In figure 2.3 it is possible to see how dropout works.

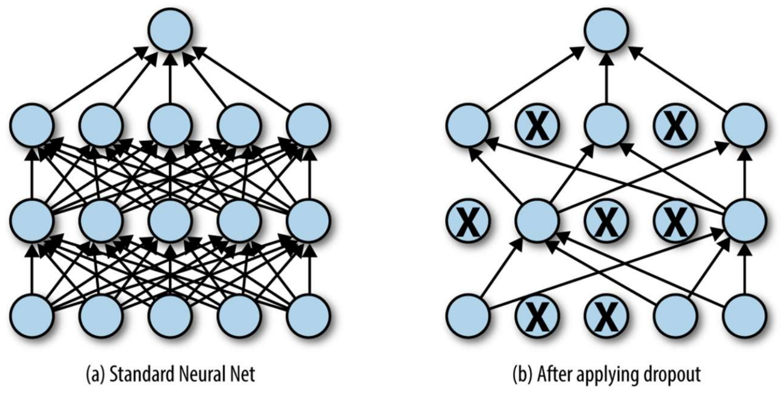


**Figure 2.3: How dropout works**

• **Activation function:** It introduces nonlinearity in the model. In particular, the rectifier is the most popular and the sigmoid is the one for binary predictions. While doing multi-class predictions, the softmax function applied to the output layer is the best choice.

• **Weight initialization:** How the weights on the edges are initialized, normally done accordingly, to the activation function of each layer. The most used distribution is the uniform one.

• **Learning rate:** It defines how quickly the network updates its parameters. If this value is low the entire process is slower, but the overall algorithm arrives at its optimum value smoothly. If the value is larger, the algorithm is faster but it could not converge: this difference is shown in figure 2.4.

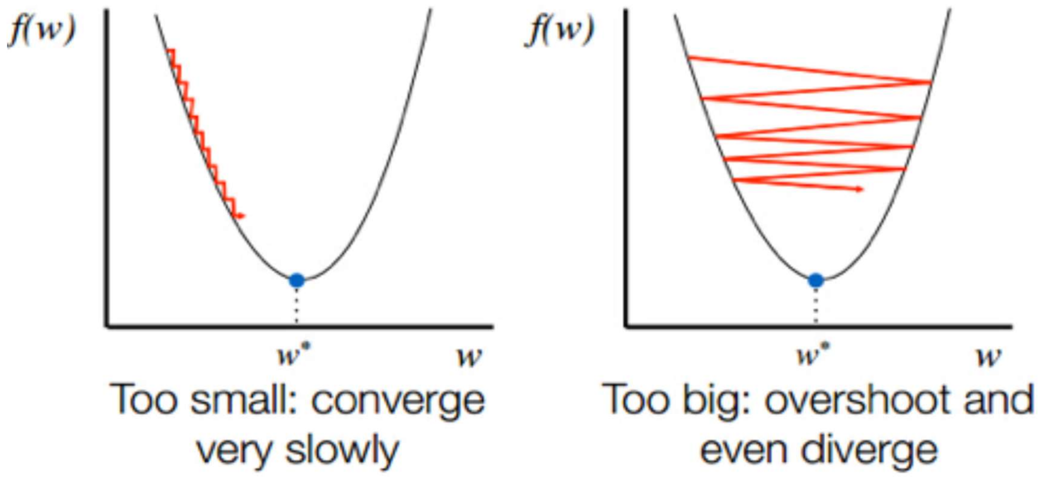

f(w)
w*
w
Too small: converge very slowly
f(w)
w*
w
Too big: overshoot and even diverge

**Figure 2.4: Differences between learning rates**

• **Number of epochs:** An "epoch" occurs when the entire training set is processed through the network. Therefore, the number of epochs sets how many times this entire set is used in the training phase.

• **Batch size:** It sets how many samples feed to the network before changing the parameters. A higher batch size will speed up the training algorithm, because the new parameters are calculated less often, but it will be less accurate.

## 2.4.2 Backpropagation

Several optimisation techniques can be used to adjust the parameters of the network. The first possible way is a brute-force algorithm, which means that every possible combination of values is tried for the edges of the network. Of course, there are limitations to this approach: if the weights are assumed to be values between two given thresholds, let's say $-l$ and $l$, with a step between each possible value of $\varepsilon$, each weight will have to be tried $\sim 2 * l / \varepsilon$ times to cover all possibilities. This amount of calculation is required for every edge and at the end, the resulting weights are those who minimize the loss. On these terms, the brute-force approach appears to be suitable only for small networks, with a few parameters to train, and the final result is not very precise. Talking about a normal fully connected network, with thousands of nodes, the edges will be so much that the brute-force method is infeasible due to the computational power and the overall time to compute the final result.

Mathematics comes to help because it is known that the derivative of a given function in a certain point is the rate at which the function is changing value in that point. It's possible to apply this concept to the loss function, to understand how it behaves as the internal weights change at a given rate.
What is interesting is the rate of which the error changes relative to the changes in the weights. For example, it could happen that if the weights change a little, for e.g., $newW = W + \varepsilon$, the overall sum of the errors can be several times greater than $\varepsilon$. So, this method can be applied also to decrease the error, by decreasing the values of the weights. Therefore, the overall learning process is given below.

1. Check the derivative of the loss function to see how much the error is increasing/decreasing.

2. If the value is positive, it means that if the weights are increased, the error is also increasing. The solution is to decrease the weights.

3. If the value is negative, it means that if the weights are increasing, the error is decreasing. The best thing to do is to increase the weights.

4. If the value is 0, the stable point is reached, and the algorithm is over.

In figure 2.5 it is shown that how the weights change.

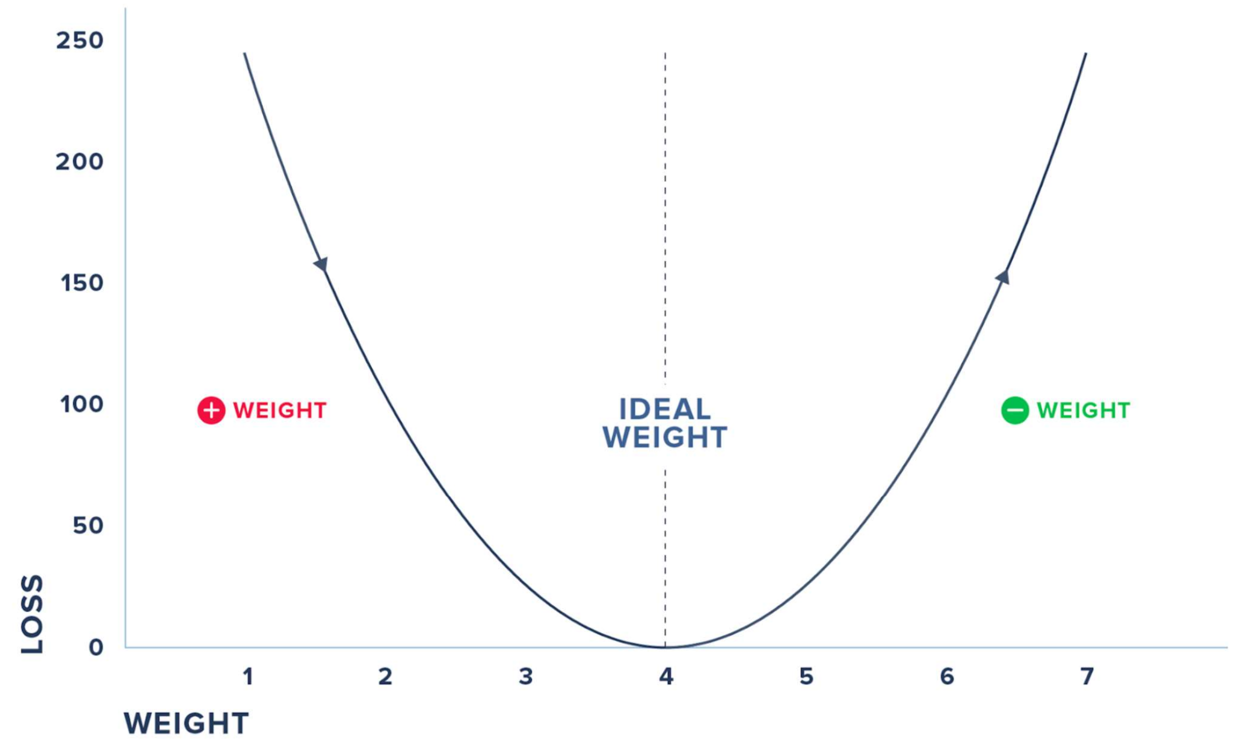


**Figure 2.5: How to change the weights with regards to the sign of the gradient**

So, the goal of the backpropagation algorithm is to change the weights inside the network such that the next results are closer to their target value.
This algorithm is briefly described as follows: it works by looking at the error generated by the feed-forward of a particular input and trying to understand how much a change in a particular edge (connected to the output) will affect the total error. This estimation is done through the partial derivative of the total error concerning the considered weight.

$$\frac{\partial C}{\partial w_i} \quad \text{......................................(2.9)}$$

Explaining how this rate is transferred from the final error to the desired weight is possible through the chain rule:

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial z_j} \cdot \frac{\partial z_j}{\partial a_j} \cdot \frac{\partial a_j}{\partial w_i} \quad \text{................................(2.10)}$$

where $a_j$ is the linear combination of the nodes as input to node j, and $z_j$ is this value fed to the activation function, e.g., the output of the network in neuron j.

The single error of neuron j is defined as $\delta$, in particular:

$$\delta = \frac{\partial C}{\partial z_j} \qquad \text{……………………………………(2.11)}$$

This value is then multiplied by a factor $\eta$, which is the learning rate, and the final quantity has to be subtracted from the weight that is considered. This entire procedure is done for every weight connected to the layer, in this case, the output.
For hidden layers, the process is similar, but need to take into account the fact that now every neuron in this layer contributes to the next one.

$$\frac{\partial C}{\partial z_i} = \frac{\partial C_1}{\partial z_i} + \frac{\partial C_2}{\partial z_i} \qquad \text{……………………………….(2.12)}$$

After having done the calculations for all the neurons in the same layer, the procedure is repeated for every layer in a backwards direction, until the input layer is reached. At this point, all the weights can be updated with their newly found value and other samples are fed to the network as inputs; then the algorithm is called again until the last batch of data is treated.

### 2.4.3 Problems

Some challenges could arise while optimizing the network during the training phase:

• **Ill-conditioning:** This issue appears if the loss function has a steep derivative, i.e., even a small change in the input or weight increases the cost function. One consequence of this problem is that learning turns out to be very slow because the learning rate must be reduced to face a strong curvature. Newton's method is very useful to optimize convex functions, but it cannot be used yet since it must be adapted to the neural network context.

• **Local minima:** Looking for a minimum in a generic function with an iterative procedure can lead the training algorithm to settle on a local minimum, which can be very different from the lowest achievable value. If the values are not so different, falling into a local minimum is not such a great problem for the overall learning result; otherwise, the trained network will have suboptimal accuracies.

• **Flat regions:** Saddle points are zero gradient regions that can be the result of the optimization algorithm. In particular, in high-dimensional spaces, saddle points are more common than local minima. It seems that the standard optimization strategies stop their iterative learning process while facing saddle points.

• **Cliffs:** While on a steep region, the gradient could make a huge step and jump off the entire cliff structure. This is solved by the gradient clipping, that reduces the update step according to the size of the gradient: bigger gradient means a small step, and vice-versa.

• **Long-term dependencies:** They happen when the same operation is repeated through various steps. To make an example, repeating the multiplication with a particular weights matrix can lead to the exploding gradient problem, which generates very steep surfaces, like cliffs.

• **Inexact gradients:** Usually, the optimization algorithms assume that the computed gradient is correct, but in practice, this value is noisy or biased, so the gradient is not perfect. The problem can be avoided by accounting for imperfections in the gradient while designing the algorithm or by choosing a loss function that is easier to approximate.

## 2.5 Validation:

The validation is an optional phase of the learning process of a network, where a new set of data, called validation set, is used to decide when to stop the training phase. The idea is to calculate an estimation of the overall loss of each network with this set that is not part of the training set. The validation set is used to simulate a test set, by calculating a validation loss on the input data. The backpropagation algorithm can find the optimal parameters of a network by adjusting the weights of the edges in the model, but this can lead the network to a situation where the result keeps floating around its optimal value. Validation has been introduced to avoid this loss of precision by keeping under control when the training procedure would generate a good test loss and stopping the training algorithm at its optimum. Usually, if after 10 epochs of training the validation loss has not improved, the learning procedure is interrupted and the network is taken as it is.

The validation, due to its ability to emulate the behaviour of a test set, is also used to understand which one, between different configurations of the model, is the best to fit the data.

## 2.6 Test:

Once various models have been trained, and the best one has been identified through validation, it is possible to assess the performance of the fully specified model.
The test set used is a set of data that have to be extracted from the same distribution as the training and validation set, to ensure consistency. If the model fits both the training set and the test set similarly, it means that no overfitting occurred. In case the test accuracy is lower than the training accuracy the generated model does not generalize the task it has been trained for.

## 2.7 Overfitting/Underfitting:

Statistically speaking, overfitting occurs when the trained model proves to be tailored very closely or in an almost exact way to a particular set of data, i.e., the training data. This implies that further sets can't be reliably predicted by this model.

Underfitting is the opposite of overfitting. In this case, the model is not able to adequately

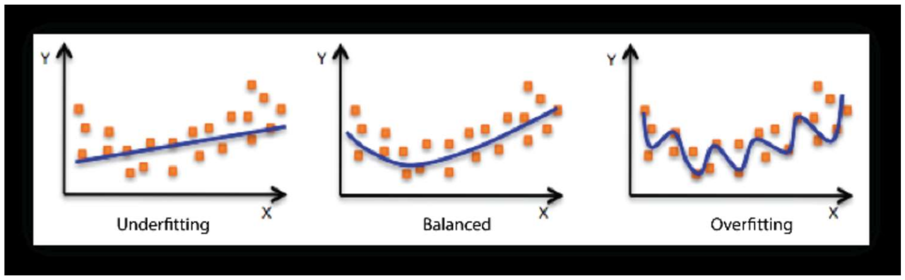


Figure 2.6: Overfitting vs underfitting

capture the inner structure of the data, bringing poor performance even on training data. This problem can be related to the structure of the model itself in figure 2.6 a comparison between overfitting, underfitting and a correct balanced model.

# Chapter 3

# Deep neural networks

A neural network is a data processing architecture which can implement an unknown function. This is made possible by the training phase, in which the network learns how to process data and provide an answer which is as correct as possible with respect to the ground truth values.
The Universal Approximation Theorem states that:

***"A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of*** **$R^n$"**

This theorem essentially says that a network with a single hidden layer can approximate in a good way every single continuous function in the real space of any dimension. In this way, every continuous function can be disassembled in its minimal parts. The weights on the edges will care about how much that particular neuron value counts for the final result. As reported

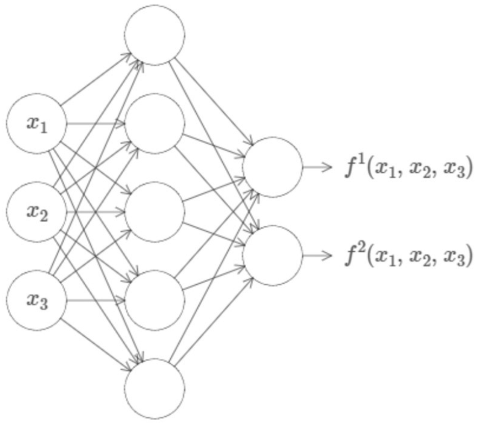


**Figure 3.1: Example of a fully-connected neural network with multiple outputs**

in the previous, weight values are learned during the training phase. These weights will then contribute to the final layer, with one or more output neurons; in figure 3.1, how multiple outputs are organized. But there are two caveats to face while declaring this statement: A neural network with a single hidden layer can only learn *continuous* functions and they are only *approximate* versions of themselves.

If a function is *discontinuous*, it is generally not possible to represent or approximate it with a network of this kind. This is a not so surprising fact because usually, the functions that a network is called to compute are continuous. In this case, the network will try to learn a continuous approximation, which normally is enough to represent it with a good accuracy

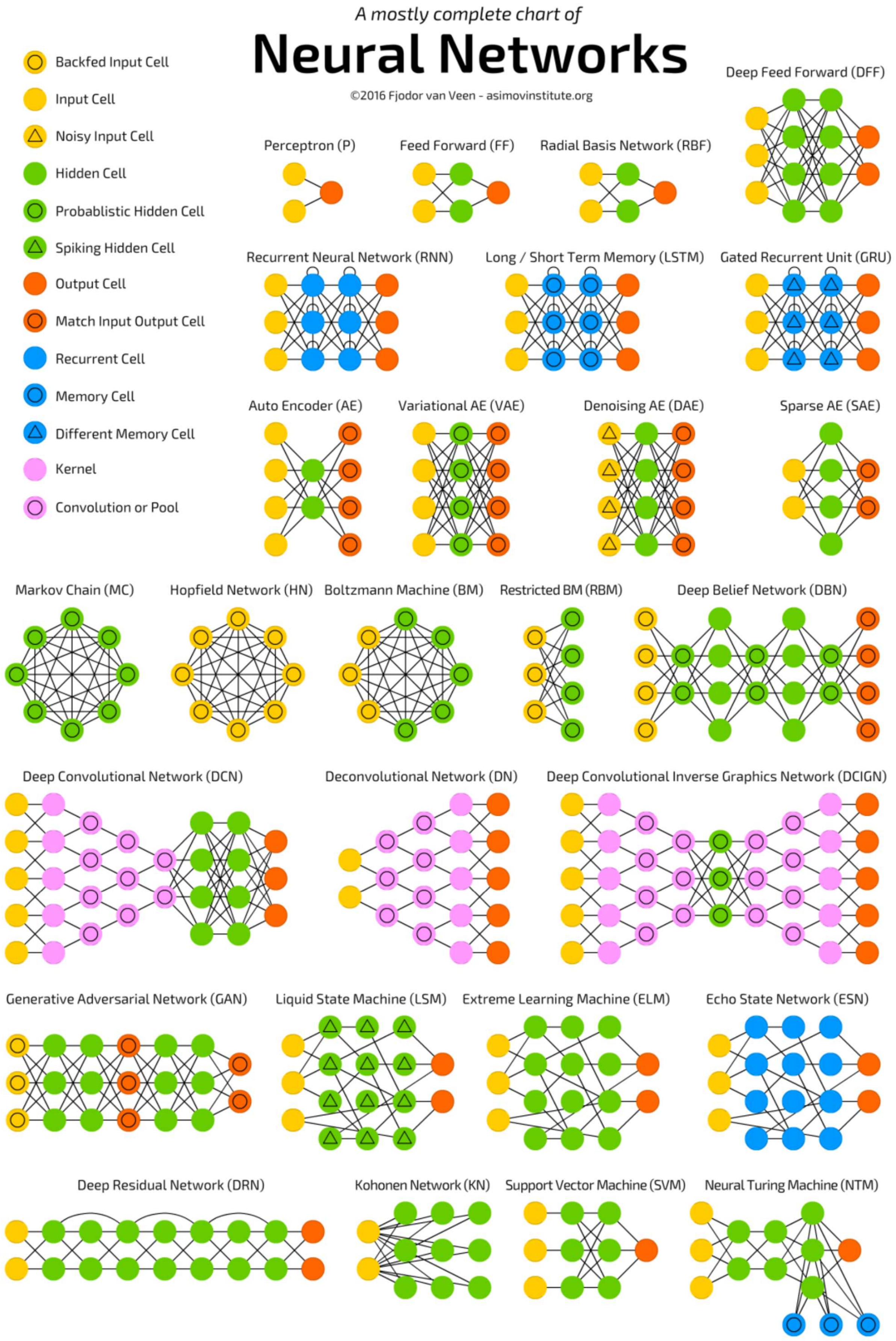

A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Figure 3.2: A mostly complete chart of Neural Networks

Another limitation is the fact that the final result is an approximated representation. The accuracy is directly connected to the number of neurons in the hidden layers. If there are few neurons, the network will not be able to understand the relational structure between input and output; if there are more of them, more nuances can be caught by the layer and the approximation will be better.

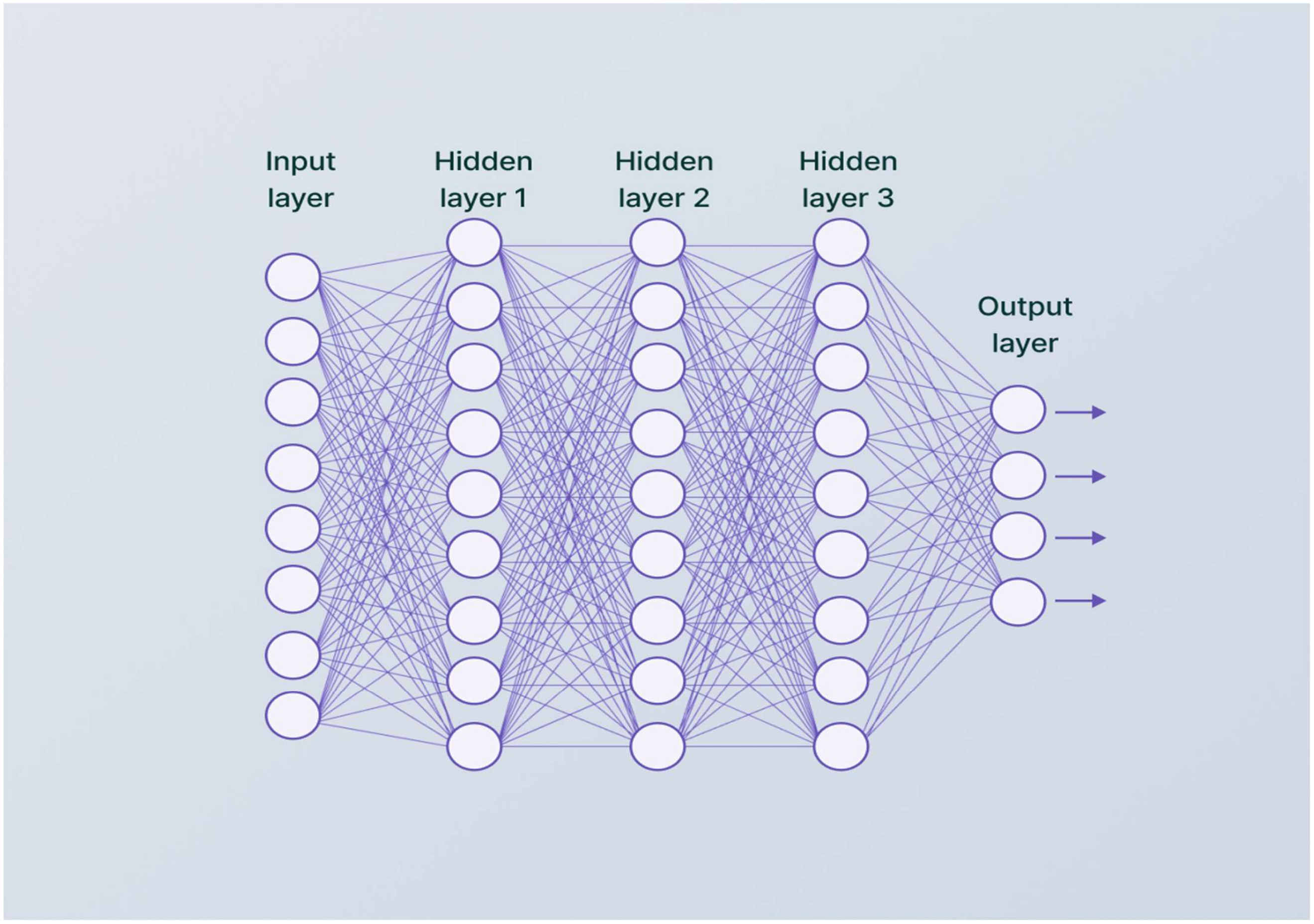


Figure 3.3: An example of a deep fully-connected neural network

This fact leads to a major drawback: the total amount of neurons required can be exponentially large. In particular, the more complex is the needed function, more neurons there will be in the layer. Since every neuron is connected to all the other ones in the following layer, if the number of neurons grows, the total complexity, which is defined by the size of the set of the neurons and the size of the set of the edges, grows in a very steep way.
To overcome these problems, the most adopted strategy is to add one or more hidden layers. This allows the network to have fewer neurons per hidden layer, fastening the feed-forward phase of the algorithm and increasing the total number of edges. This can bring a better approximation of the function that is been considered. The overall complexity of the system

grows, as it can be seen in figure 3.3, but this allows to compute a wider family of functions, both linear and non-linear. In general, a neural network is defined as “deep” if there are multiple hidden layers between the input and output layers.
A deep neural network can generate a compositional model of an object where it is represented as a composition of primitive data, arranged in layers. With more layers, the input can be decomposed in its primitives at a deeper level, regardless of the complexity of the object.

This feature comes to help when harder problems are taken into account: real-life artificial intelligence applications require a deep analysis of every observed data. To make a couple of examples: a pixel of a certain subject in a picture can change its colour values if the photo is taken in different moments of the day, even if the context and the subject itself are the same; a registration of a phrase spoken by a person could be very different from the registration of the same phrase done by another person: there are many factors that can lead to a complete mistake, like the accent and the tone of voice.
Deep neural networks can overcome these problems by considering only the object itself, ignoring every misleading factor, decomposing the computation into nested simple functions, each one belonging to a different layer. The abstraction grows as the input proceed into the network, allowing an easier computation, as it is show in figure 3.4.
Besides their slightly different approach and scope, deep neural networks are very similar to standard neural networks, since the seconds are just a particular case of the firsts.

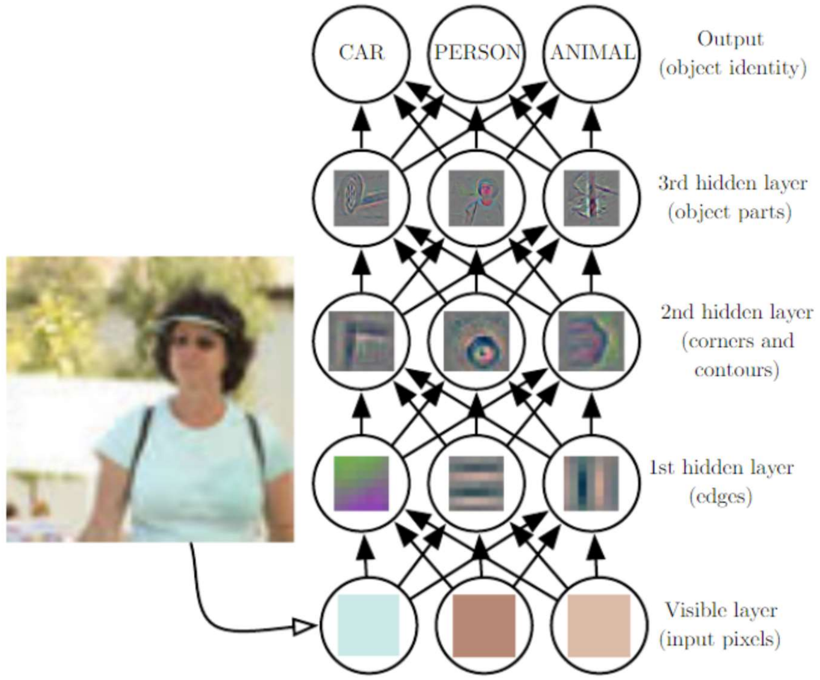


Figure 3.4: Some features captured by a neural network

Many different deep networks have been created to face particular tasks, like face and speech recognition or image classification; these structures have been tested for years and at this point they are capable to fulfil their duty and reached a good level of reliability. In this work, in parallel with the standard implementation of deep neural networks, have been also considered as two alternative architectures: *convolutional neural networks* and *residual neural networks*.

## 3.1 Convolutional neural network

Convolutional neural networks are specialized kind of network thought to process data that has a grid-like topology. An example could be an image dataset, where each piece of data is an image, that is seen as a 2D grid of pixels. As the name says, the particularity of this architecture lays on the convolutional term: it means that at least one layer in the network implements the convolution in place of the standard matrix multiplication.
Mathematically the convolution is an operation on two functions **f** and **g** which produces a third function expressing how the shape of the first is modified by the other one. It is defined as the integral of the product of the two functions with the second one that has been inverted and shifted.

$$(f \circledast g)(t) := \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau \qquad (3.1)$$

$$(f \circledast g)(t) := \sum_{k=-\infty}^{+\infty} f(k)g(t-k) \qquad (3.2)$$

The first term usually is referred to as the input, while the second is the kernel. The output is sometimes called feature map.
While the input is usually a multidimensional array of data, the kernel is usually a multidimensional array of parameters, which will be trained and adjusted by the network.
From figure 3.4 above it can be seen that the output of the convolution between a grid of values and a kernel is just a linear combination of the values involved, which in the case of the kernel they are always the same; those who change are those about the portion of input considered in that particular iteration. Typically, a single convolutional layer is divided into three operations or stages:

**1. Convolution stage:** In this stage, the layer performs various convolutions in parallel and produces a set of linear activations.

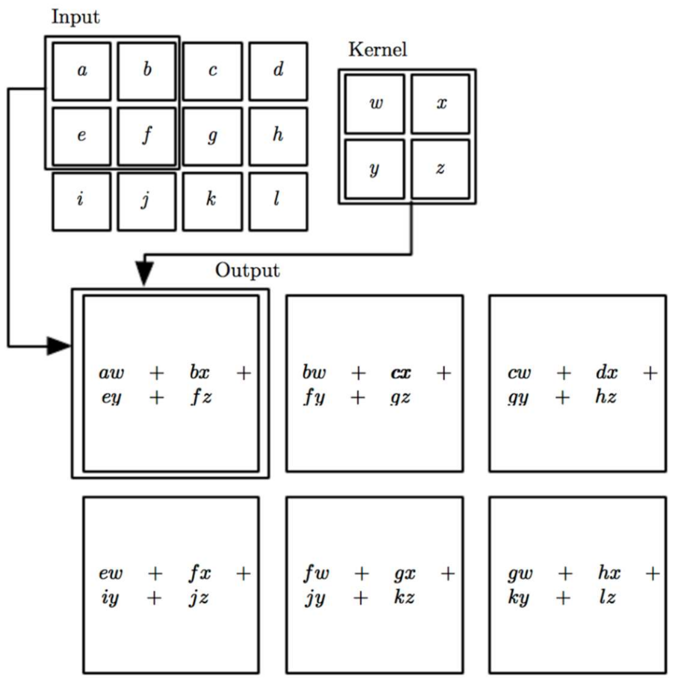


**Figure 3.5: Visual representation of 2-D convolution without kernel flipping (Goodfellow et al. 2016, p.330)**

**2. Detector stage:** In this phase, the output set of the first one is run through a non-linear activation function, such as the ReLu.

**3. Pooling stage:** Here the output of the detector stage is further modified with a pooling operation, which is a function that replaces the output of a net with a summary of the surrounding outputs.

The most common pooling functions used are:

• **Max pooling:** Where the final output is the maximum value in the considered set of values fed to the function. An example is shown in figure 3.6.

• **Min-pooling:** It is same as max-pooling, but the value taken is the smallest one.

• **Average pooling:** Here the final output, as the name suggests, is the average of all the values that the function takes as input. Figure 3.7 is a clear example.

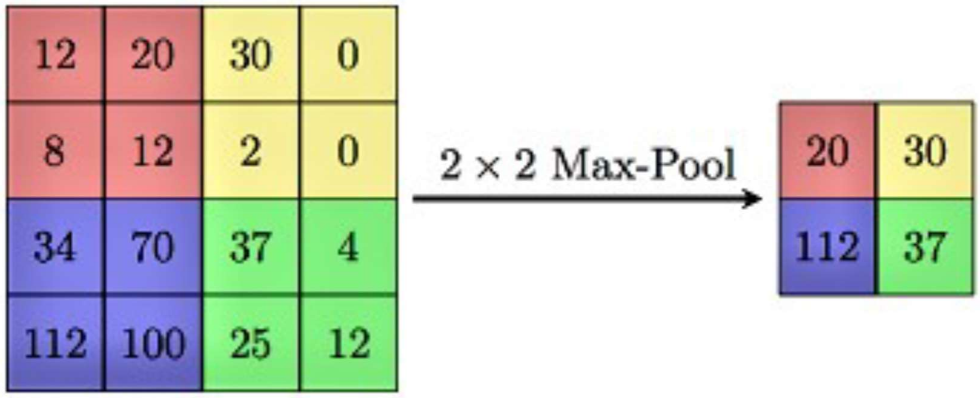


Figure 3.6: How max-pooling works

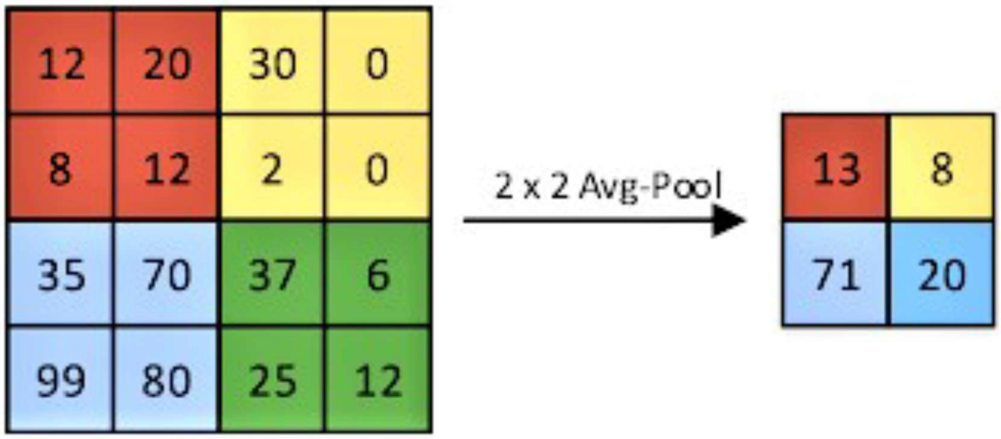


Figure 3.7: How avg-pooling works

Usually, the choice of the pooling method depends on the input data: if the dataset is composed of images, by using the average pooling the final result will be an image with smoothed colours and some information about the edges will be lost; on the other hand, using the max pooling will bring out more information from darker images, and vice-versa for the min pooling.

In every case, pooling helps to make the representation invariant to small changes of the input, so the final result will be more robust.

This is the case of pooling over results of the same convolutions, but this operation is effective even when applied to results coming from separately parametrized convolutions. On the example in figure 3.8 the same input is fed to three different filters, which try to match the input with a rotated version of it. A match in one of these filters will bring out a larger output than the others, so a max-pooling over the outputs of the filters will be able to obtain a large response in any case, even with slightly different inputs.

The size of each layer reduces at each stage, and this is caused by both the convolutional

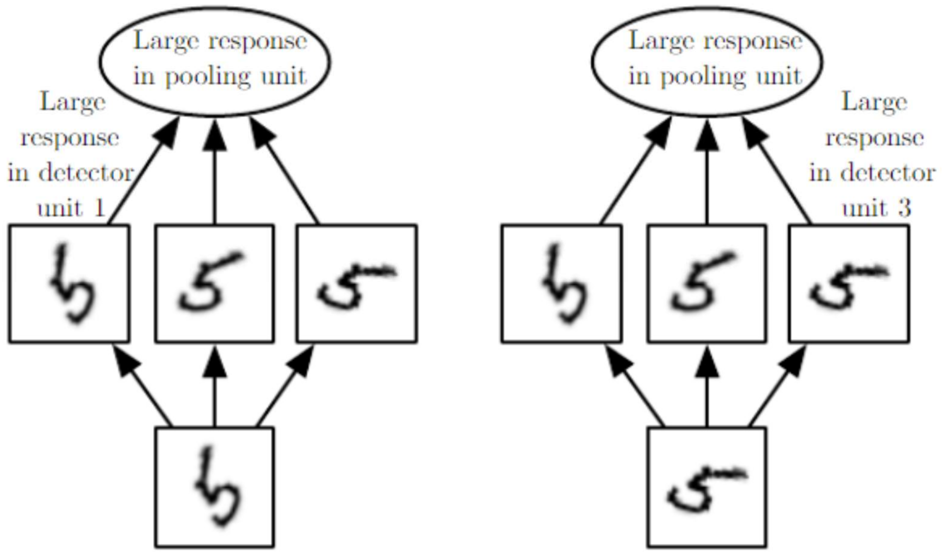

Large response in pooling unit
Large response in detector unit 1
Large response in pooling unit
Large response in detector unit 3

Figure 3.8: Example of learned invariances

stage and the pooling stage. This happens because the filtering operation, which involves a linear combination between the values of the filter and the image, will make the involved pixels collapse into a single one, reducing the dimensionality; the further stage, the pooling with size *q* x *q*, is responsible for the elimination of some unwanted information, reducing the size again, obtaining a final output of size *(N/q)* x *(M/q)*.

For example, an image of 100 x 80 pixels as input, after the first convolutional layer with filters 5 x 5 and pooling of size 2 x 2 will result in a feature map of size 48 x 38.

So, this means that while going on with the computation, the entire input will collapse to a really small feature map e consequently a lot of information will be lost; the number of kernels comes to help in this situation because the further one gets in the network, the more filters there will be per layer. It is a general rule that while shrinking the size of the single feature map, more of them will be generated.
Very far away from the input layer, depending on the initial size, the feature set will become more and more similar to a layer of a fully-connected neural network. This entire structure is graphically described in figure 3.9.

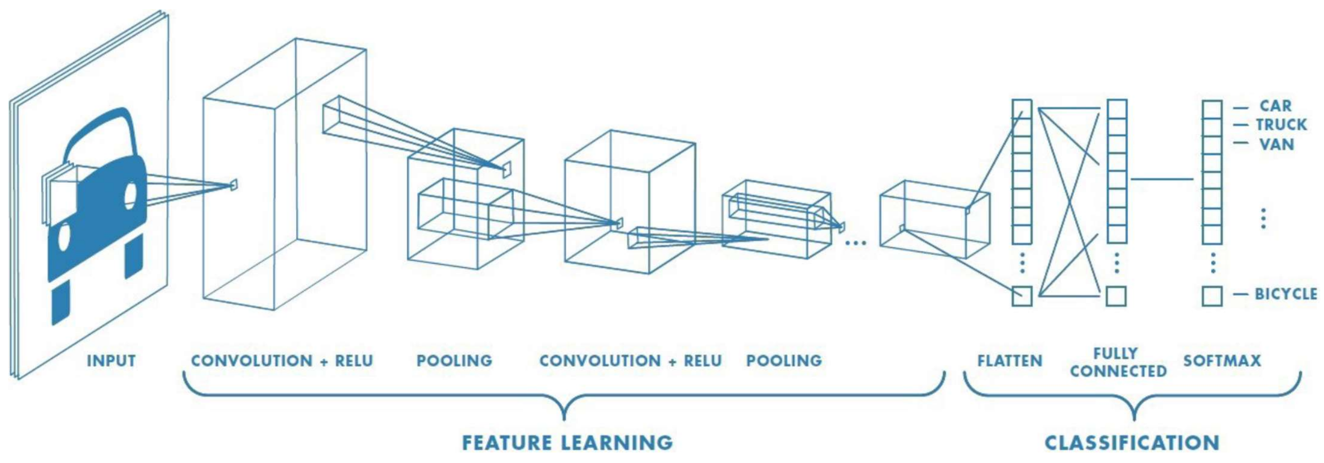


Figure 3.9: Usual structure of a convolutional neural network

Generally speaking, convolutional neural networks decompose the input in various features using the convolutional operation. The features will be more and more abstract as the considered
layer is far away from the input one, and few final fully-connected layers complete the model allowing this structure to perform standard machine learning tasks on grid-like datasets.

## 3.2 Residual neural network:

Residual neural networks were born to face the problem of degradation: if a model is trained to map a certain function and if some layers mapping the identity function are added, the overall accuracy of the network should be not lower than its shallow counterpart. In real cases, it can be shown that adding layers to a network leads to a degradation in performance of the overall network because the error is hard to propagate on the earlier layers; this is due to the exponential reduction of the gradient while travelling backwards the network to adjust the weights.

The idea behind ResNet is to make a network or a set of layers learn not the needed function *H(x)*, but the difference between this function and the identity function, and this quantity *f(x)* is called residual:

$$H(x) = f(x) + x \rightarrow f(x) = H(x) - x \quad \ldots\ldots\ldots\ldots\ldots(3.3)$$

The *x* is simply obtained by a skip connection between the input layer and the needed one. Each one of these structures, where there are few layers in the middle of a shortcut connection between the first and the last, is called Block, and an example of it is in figure 3.10.

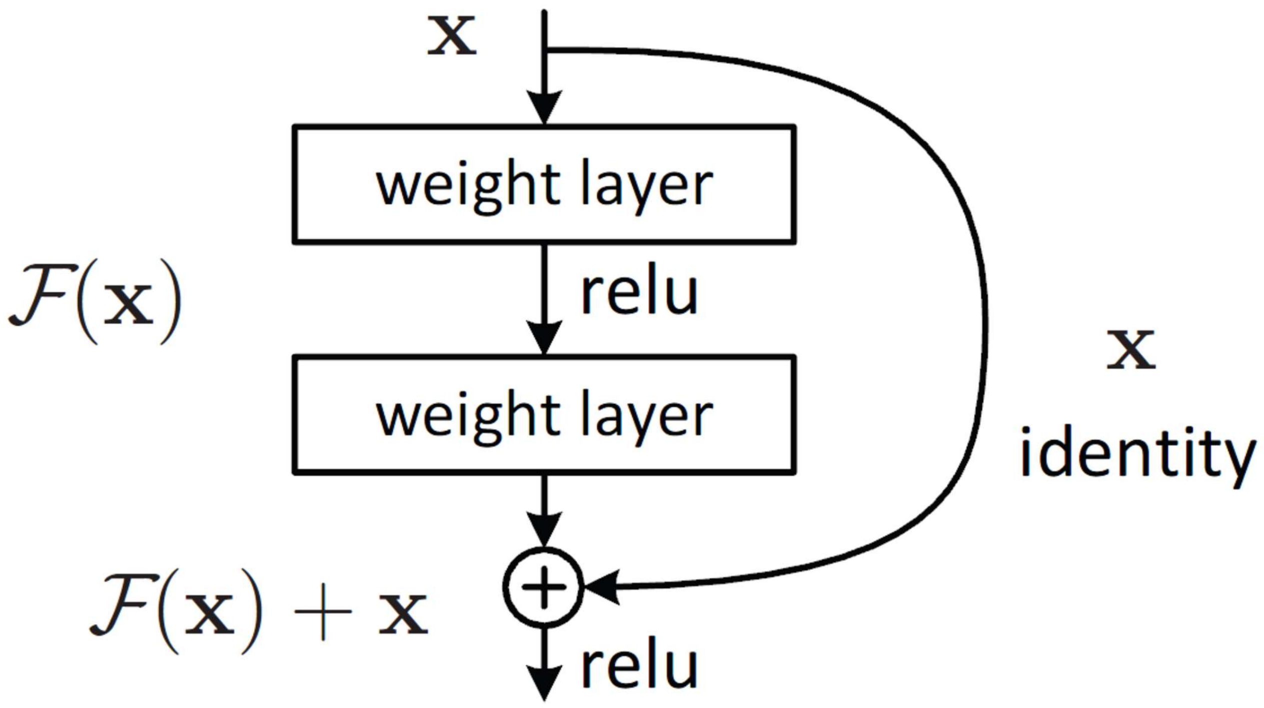


Figure 3.10: Example of a single block of a residual neural network

As can be seen from the above figure, the needed input x is given to a series of layers which can approximate f(x), called residual, and add at the output of this function the input, in an element-wise operation.
Usually, the input and the output of the layers have the same size, to ease the operation of adding, but in the cases in which this do not happen, zero-padding techniques or 1  1 convolutions are used to adapt the dimensions. The learning phases and the backpropagation of the error seen in the fully-connected approach are then applicable here without any problem.
About the model that has been described earlier in this paragraph, adding one or more block instead of the usual layers, will maintain the accuracy of the shallow network without increasing the error: this happens because it is easier to force a function like f(x) described by the layers in the block, to remain simple, because the skip connection will exclude some of its overall complexity by bringing over the function f(x) = x. In this way, ResNet gives the layers a reference point to start mapping the identity function, which is the shortcut connection x, instead of let them create a new identity function for each block.

The idea behind ResNet, (see figure 3.11), was to increase the number of hidden layers (with respect to a fully-connected approach) at a feasible computational cost.

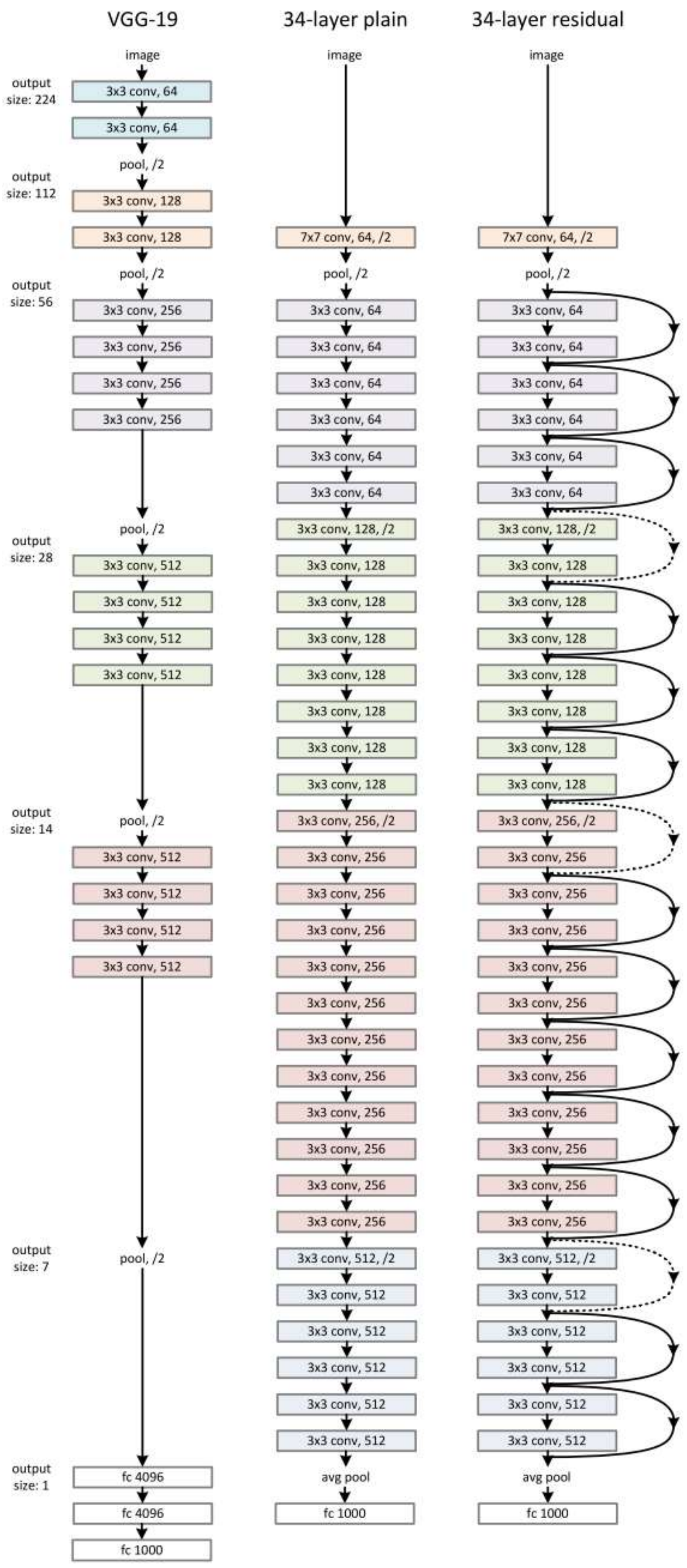


**Figure 3.11: First ResNet architecture**

The neural network with the biggest number of layers is a ResNet with more than a thousand layers, while the biggest number of layers for a fully-connected neural network is still less than thirty.
The thousand-layer model has been proved to be able to generate a training error < 0:1%, while keeping the test error to a good 7:93%.

## 3.3 Autoencoder

There are a lot of different neural network architectures. Some are the best while treating images, others that can be used to learn accurately complex functions more accurately. One of them is the autoencoder, a neural way to compress data. This kind of neural network can be related to both compression and encoding: they were developed to take a vector from Rn and translate it into an *m-dimensional* representation, with m < n.

This neural architecture learns how to extract the most important features of the input and build a smaller, yet complete, representation of it. Then, the inverse process is possible, taking the compressed code and obtaining the original data after having applied a decoding function.

The simplest autoencoder can be modelled by a neural network with just three layers:

1. Input: this layer, as usual, takes the desired data and feeds it into the network;
2. Code: this is the layer of the code, where the reduced representation of the input sample will be shown. Its size is the length of the encoding of the data;
3. Output: in this architecture, the size of the output must be the same as the size of the input, to allow the network to understand if the entire model caught the reduction and the following expansion of the features in a reliable way;

These phases can be grouped into two main operations, visible also in figure 3.12:

- **Encoder:** The first and the second layer, represent an encoding function from $R^n$ to $R^m$.

- **Decoder:** The second and the third and last layers implement a decoding function from $R^m$ to $R^n$, to get the original data back from the coded representation.

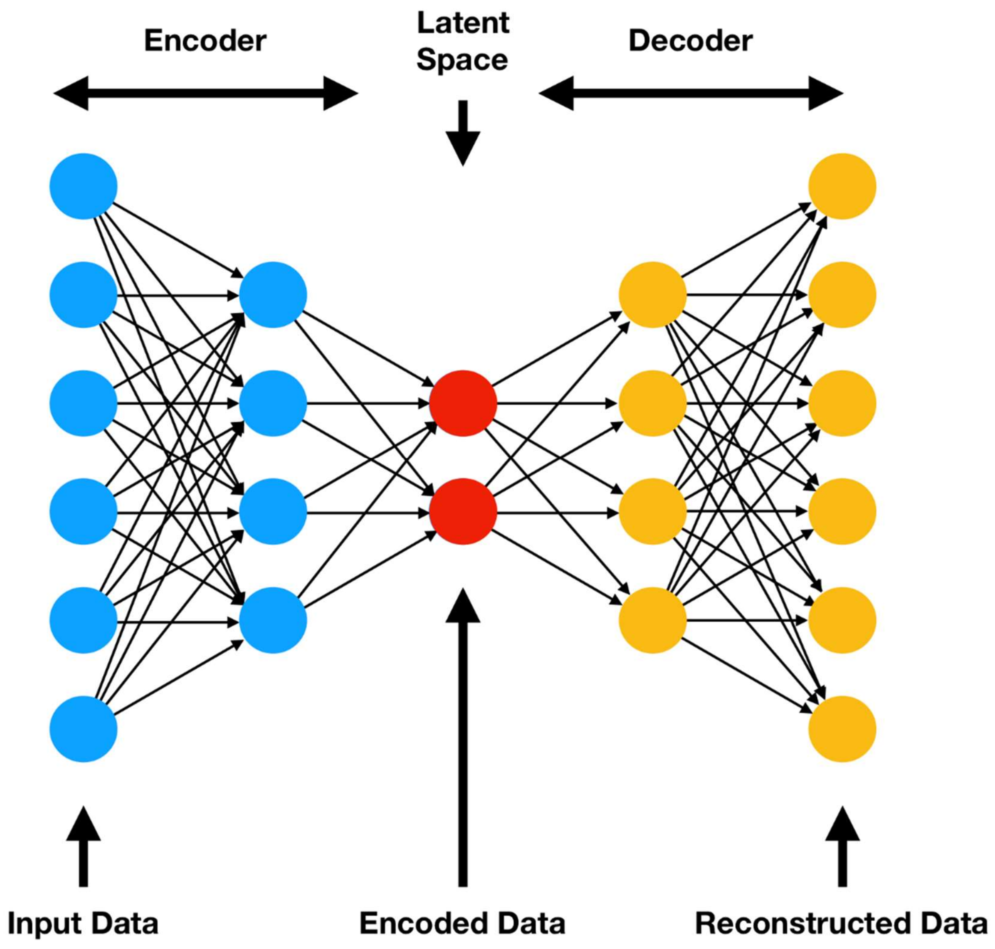


Figure 3.12: The organization of the layers in a simple autoencoder

This means that autoencoders do not need to have a labelled datasets to learn because their loss function describes how much the obtained output is similar to the input that generated it.

## 3.4 Mathematical approach

The two parts of a simple fully-connected autoencoder, the encoding and the decoding parts, are, respectively, two functions $\phi$ and $\psi$, such that

$$\phi : X \rightarrow F \quad \text{.....................................} \quad (3.4)$$

$$\psi : F \rightarrow X \quad \text{.....................................} \quad (3.5)$$

with

$$\phi, \psi = \underset{\phi,\psi}{\arg\min} \; \|X - (\psi \circ \phi)X\|^2 \quad \text{.....................................(3.6)}$$

### 3.4.1 Encoder

The encoder phase takes the input x $\epsilon$ $R^n$ and codes it into x $\epsilon$ $R^m$:

$$z = \sigma(Wx + b) \cdots\cdots\cdots\cdots\cdots (3.7)$$

with W as the weight matrix of the encoder, b as the bias and $\sigma$ as the activation function.

### 3.4.2 Decoder

The decoder takes in input z $\epsilon$ $R^m$ and tries to decode it back to x $\epsilon$ $R^n$:

$$z = \sigma'(W'z + b') \ldots\ldots\ldots\ldots\ldots\ldots (3.8)$$

where W' is the weight matrix of the decoder, b' is the bias, and $\sigma$' as its activation function. Subsequently, the algorithm has to minimize a function of this shape:

$$f(x, x') = \|x - x'\|^2 = \|x - \sigma'(W'(\sigma(Wx + b)) + b')\|^2 \ldots\ldots\ldots\ldots (3.9)$$

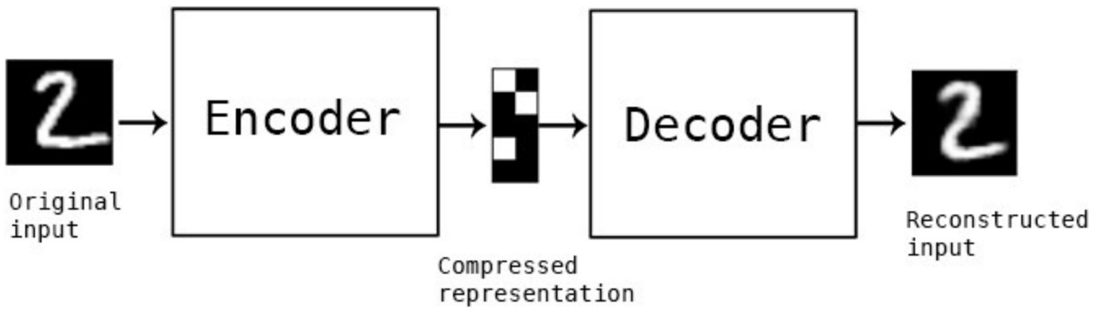


Figure 3.13: A sample application of an autoencoder with MNIST data

## 3.5 Regularized autoencoders

There are autoencoders where the number of neurons in the hidden layer is the same or even bigger than the size of the original data. These borderline cases can lead the model to learn the representation of hidden feature structures, but in the most of the cases, the learned function includes the identity. Various techniques has been developed to avoid this phenomenon, and the architectures resulting are knows as *regularized autoencoders*.

## 3.5.1 Sparse autoencoders

Sparse autoencoders have more neurons in the hidden layers than in the input one. However, these neurons are not all active at the same time, only few of them per iteration can be triggered. This technique can make the model learn all the possible features of the input dataset in the hidden layer, without having all of them involved in the analysis of a single input.

In more detail, this sparsity constraint is a "penalty" which is added to the loss function and depends on the hidden layer; then this sum is what will be minimized in the training phase:

$$f(x, x') + \Omega(z) \quad \text{.............................}(3.10)$$

A neuron is "active" when its output is close to the maximum value allowed by the activation function, and "inactive" in the other case. In the following explanation, the assumed activation function is the *sigmoid*, with maximum output 1 and minimum output 0. Let aj be the activation factor the j-th neuron in the hidden layer and xi the input neuron connected to the considered hidden unit. The average activation over m samples of the hidden unit j is defined as follows:

$$\hat{\rho}_j = \frac{1}{n}\sum_{i=1}^{n}(a_j x_i) \quad \text{.............................}(3.11)$$

where $\hat{\rho}_j$ to be forced to be close to 0 (zero), which means that the most of the neurons are inactive. Therefore, given the "sparsity parameter" $\rho$, which is a constant that defines the percentage of contemporary active neurons for a given input, the method forces $\hat{\rho}_j = \rho$. So, the penalty will penalize $\hat{\rho}_j$ when it deviates significantly from $\rho$; this is done with the Kullback-Leibler divergence:

$$KL(p_x||p_y) = p_x log\frac{p_x}{p_y} + (1 - p_x)log\frac{1 - p_x}{1 - p_y} \quad \text{..........................}(3.12)$$

for generic $p_x$ and $p_y$ probability mass distributions, defined in the same probability space. This quantity is the indication of how much the random variable $p_x$ is related to the random variable $p_y$. It's some sort of asymmetric distance function, since it's close to 0 if the two random variables have similar distributions, and it's positive and greater than 0 if the two distributions are very different; in particular, the more uncorrelated the two distributions are, the bigger will be their divergence value.

In this case, the KL divergence needs to be summed all over the hidden units in layer 2:

$$\sum_{j=1}^{m} KL(\rho||\hat{\rho}_j) = \sum_{j=1}^{m} \rho log\frac{\rho}{\hat{\rho}_j} + (1-\rho)log\frac{1-\rho}{1-\hat{\rho}_j} \quad \text{.........................(3.13)}$$

With this penalty function, if $\hat{\rho}_j = \rho, KL(\rho||\hat{\rho}_j) = 0$ greater than 0. In the example below, $\rho$ has been set equal to 0.2, and the plot $KL(\rho||\hat{\rho}_j)$ of for different values of $\hat{\rho}_j$ is in figure 3.14. As can be seen in figure 3.13, when $\hat{\rho}_j$ = 0.2, the divergence is 0, and the network will learn to activate certain hidden neurons concerning the input.

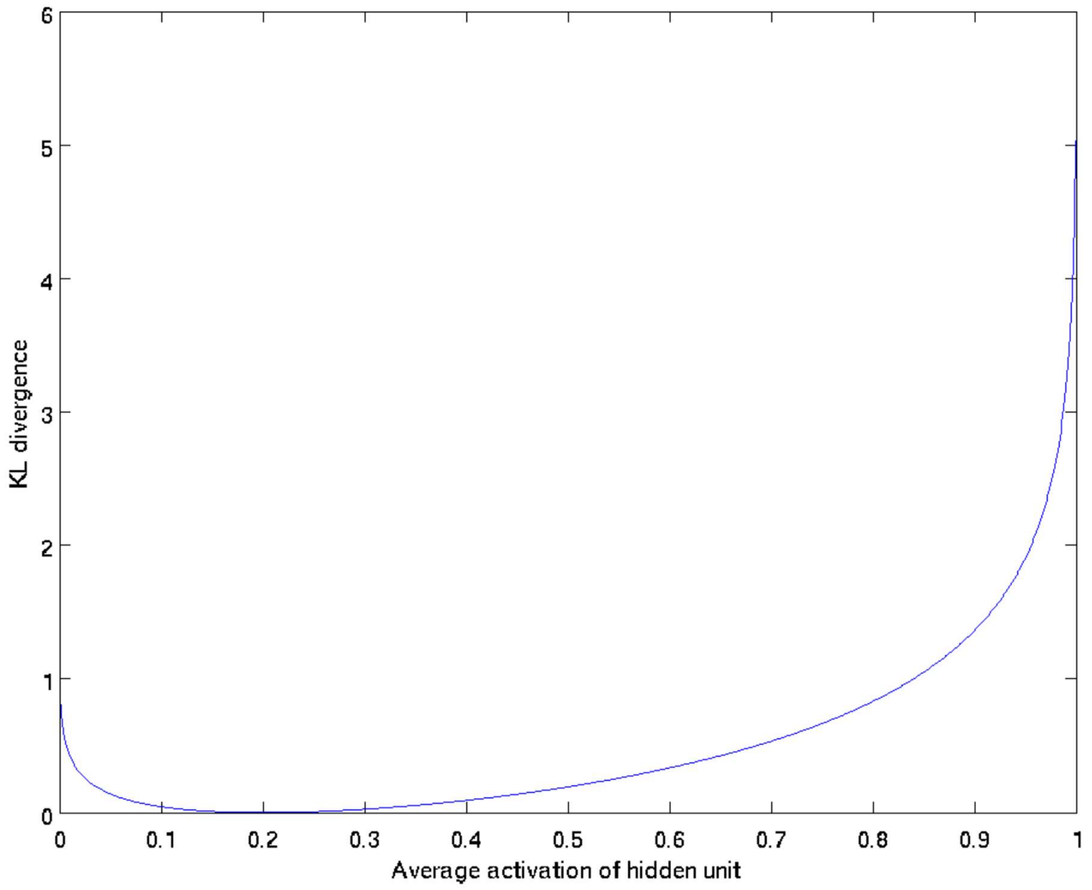


Figure 3.14: KL divergence as $\hat{\rho}_j$ varies

## 3.6 Applications of autoencoders

Autoencoders are simple to understand and useful for many applications. It's possible to build them in every shape, and one can add all the needed hidden layers. Over the years, various applications have been discovered to be easy to implement on this kind of neural network, so it's possible to create a fine model which has a great accuracy while performing its task. Some of these applications are:

• **Image colouring:** By understanding which colour is normally associated to a specific shape, it's possible for an autoencoder which learns with black/white images and their colorized version to take a greyscale image and apply to it a plausible colour palette.
An example in figure 3.15

3.15: Example of a b/w image colored with an autoencoder. The real image is in Figure the middle.

• **Dimensionality reduction:** Instead of using Principal Component Analysis (PCA), a valid technique based on the variance along the dimensions of representation of the data, it's possible to configure an autoencoder to make it learn the patter of the most important features in a piece of data, and use them to rebuild the initial data in the more accurate way possible. This is visible in figure 3.16.

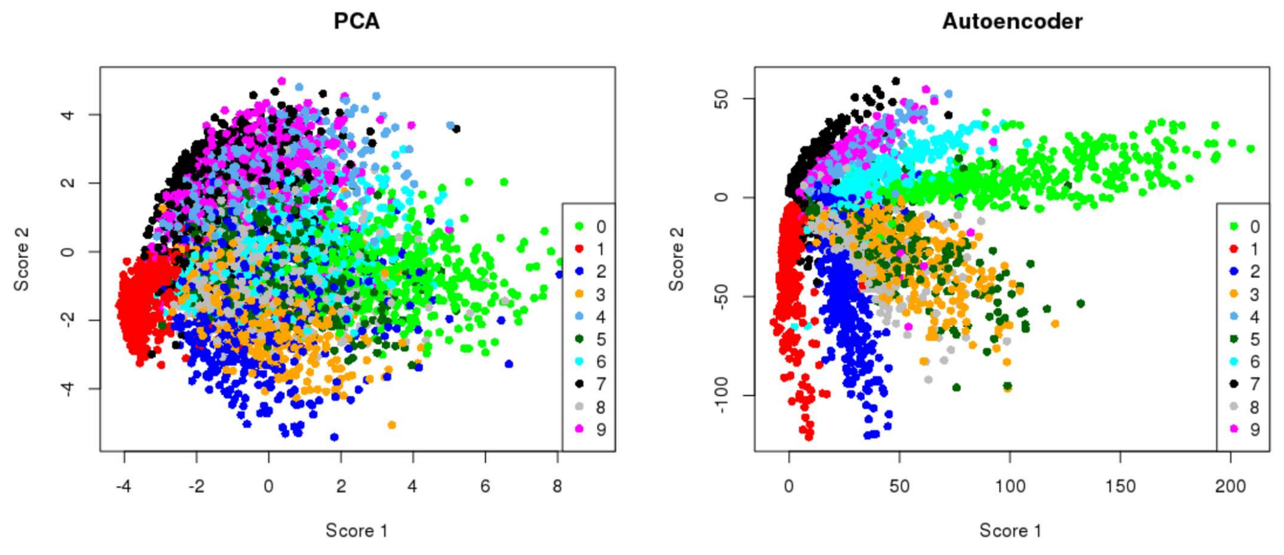


Figure 3.16: Example of feature reduction in a dataset

• **Watermark removal:** Watermarks are generally symbols that are added to data to prevent their usage in an unauthorized way. They can be removed by making an autoencoder learn their pattern and then extracting it like in figure 3.17 from the desired input.

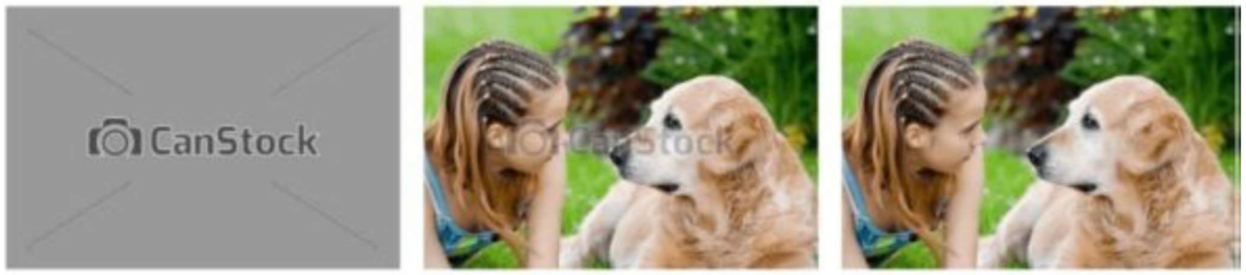


Figure 3.17: The watermark on the left is successfully removed from the middle image. The result is on the right side

Of course, there's continuous research about the possibilities of this neural architecture. These stated above are just examples, but there are many applications of this kind of network, making it one of the most versatile neural network on the scene.

# Chapter 4

# Implementation

This project is divided into two parts. The first part revolves around training a model and the scientific research behind the technology used to do it. The second part is about using the trained model that is supposed to detect the license plate numbers as objects and implement it in the android application. This chapter then will cover the different steps followed along the process of realizing this project.

## 4.1 Software & Tools

The thesis uses free GPUs from Google Colab (Colaboratory). The deep learning framework used is TensorFlow with Keras API.

**TensorFlow:** Neural Networks have been heavily used the last few years to solve different and difficult problems. Since then, many frameworks that facilitate the implementation of these networks have appeared. TensorFlow is one of them. It is developed by Google and until recently it was only used internally in the company, than Google made it open source and it includes now a large and active Community that helps improving it every day. This framework is built to support every aspect of the process from training models to deploying them to servers or mobile devices which make it stand out from other frameworks. (cited in Warden 2017, p.2; cited in TensorFlow n.d. a)

**Tensorboard:** This tool is provided by TensorFlow and used to visualize the training process with different graphs to help optimize the model in the future. (cited in TensorFlow n.d. b)

## 4.2 Some definitions

### 4.2.1 Train, validation and test

The train dataset is used to train the model with. In the case of neural networks, the model learns its weights and biases.
The validation dataset is what the model uses for evaluation after every set of predictions. It helps the model tune its hyperparameters.
The test dataset is used to evaluate the model after it has been completely trained.

### 4.2.2 Overfitting and underfitting

Overfitting occurs when the model captures the noise of the data. Intuitively, it fits the data too well, or in other words it is too dependent on the data used for training.
Conversely, underfitting occurs when the model cannot capture the underlying trend of the data, or intuitively it does not fit the data well enough.
Overfitting and underfitting both result in poor predictions in new datasets.

### 4.2.3 Batch size

Most of the time the whole dataset cannot be fed into the neural network at once, so it has to be divided into parts, or batches. The batch size indicates the number of training samples in a single batch.

### 4.2.4 Epoch

One epoch is when the entire dataset (i.e., every training sample) is fed forward and backward through the neural network only once.

### 4.2.5 Dropout

Dropout is a technique used to reduce overfitting. The term dropout refers to randomly dropping out units and their connections during training.

### 4.2.6 Batch normalization

Batch normalization is also a method to reduce overfitting. It normalizes the input layer by adjusting and scaling the activations. The mathematics behind batch normalization is out of the scope of this thesis.

## 4.3 The Dataset

The dataset that we are going to use for the image classification is Chest X-Ray images, which consists of 2 categories, Pneumonia and Normal. This dataset was published by Paulo Breviglieri, a revised version of Paul Mooney's most popular dataset. This updated version of the dataset has a more balanced distribution of the images in the validation set and the testing set. The data set is organised into 3 folders (train, test, val) and contains subfolders for each image category Opacity(viz. Pneumonia) & Normal.

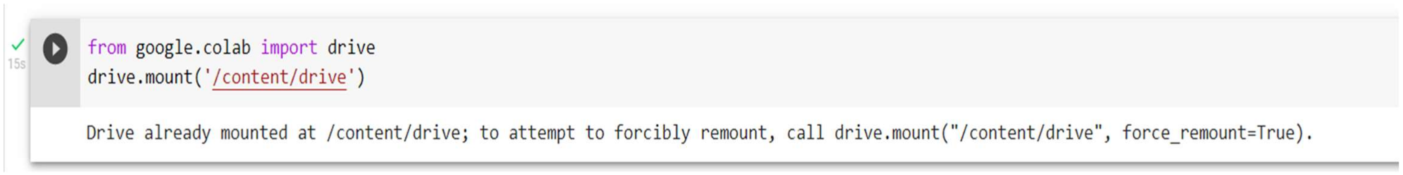
*Total number of observations (images): 5,856*

*Training observations: 4,192 (1,082 normal cases, 3,110 lung opacity cases)*

*Validation observations: 1,040 (267 normal cases, 773 lung opacity cases)*

*Testing observations: 624 (234 normal cases, 390 lung opacity cases)*

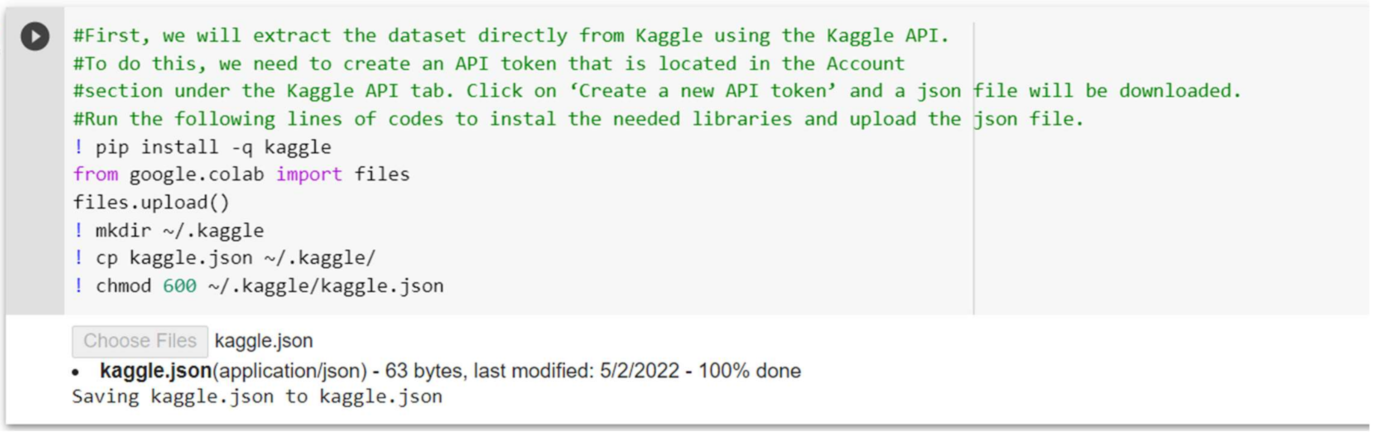
First, we will mount the google drive by two line of code.



```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

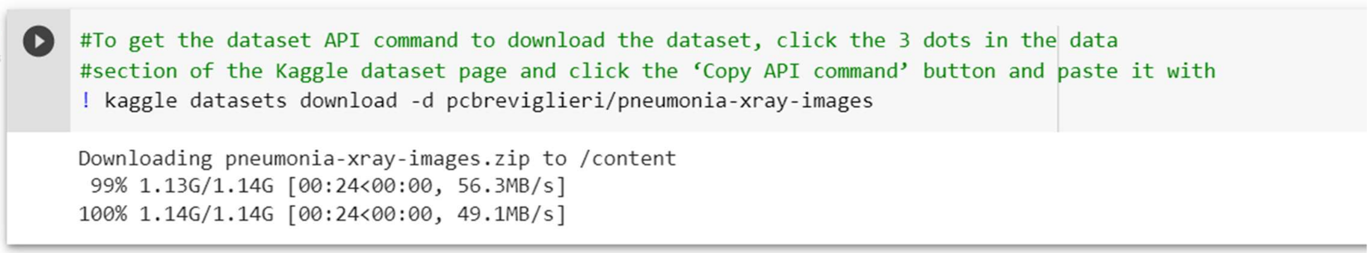
Then we will extract the dataset directly from Kaggle using the Kaggle API. To do this, we need to create an API token that is located in the Account section under the Kaggle API tab. Click on 'Create a new API token' and a json file will be downloaded. Run the following lines of codes to instal the needed libraries and upload the json file.



```
#First, we will extract the dataset directly from Kaggle using the Kaggle API.
#To do this, we need to create an API token that is located in the Account
#section under the Kaggle API tab. Click on ‘Create a new API token’ and a json file will be downloaded.
#Run the following lines of codes to instal the needed libraries and upload the json file.
! pip install -q kaggle
from google.colab import files
files.upload()
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
```

```
kaggle.json
• kaggle.json(application/json) - 63 bytes, last modified: 5/2/2022 - 100% done
Saving kaggle.json to kaggle.json
```

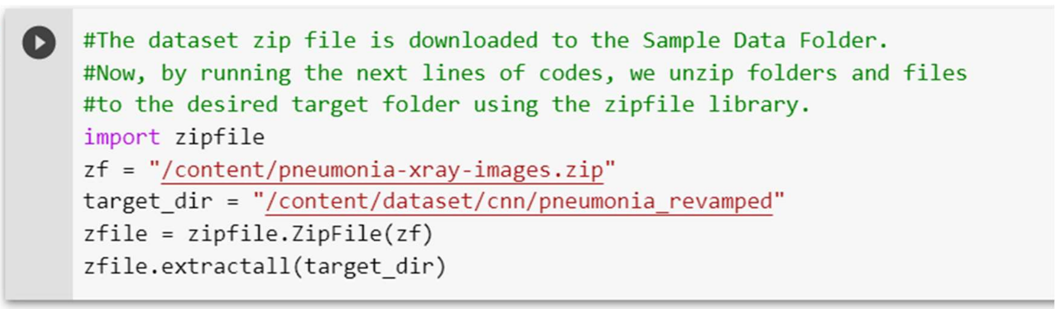
When prompted to ‘Choose Files,’ upload the downloaded json file. Running the next line of code is going to download the dataset. When prompted to ‘Choose Files,’ upload the downloaded json file. Running the next line of code is going to download the dataset. To get the dataset API command to download the dataset, click the 3 dots in the data section of the



```
#To get the dataset API command to download the dataset, click the 3 dots in the data
#section of the Kaggle dataset page and click the ‘Copy API command’ button and paste it with
! kaggle datasets download -d pcbreviglieri/pneumonia-xray-images
```

```
Downloading pneumonia-xray-images.zip to /content
 99% 1.13G/1.14G [00:24<00:00, 56.3MB/s]
100% 1.14G/1.14G [00:24<00:00, 49.1MB/s]
```

Since I use Google Colab to run this project, the dataset zip file is downloaded to the Sample Data Folder. Now, by running the next lines of codes, we unzip folders and files to the desired target folder using the zipfile library.



```
#The dataset zip file is downloaded to the Sample Data Folder.
#Now, by running the next lines of codes, we unzip folders and files
#to the desired target folder using the zipfile library.
import zipfile
zf = "/content/pneumonia-xray-images.zip"
target_dir = "/content/dataset/cnn/pneumonia_revamped"
zfile = zipfile.ZipFile(zf)
zfile.extractall(target_dir)
```

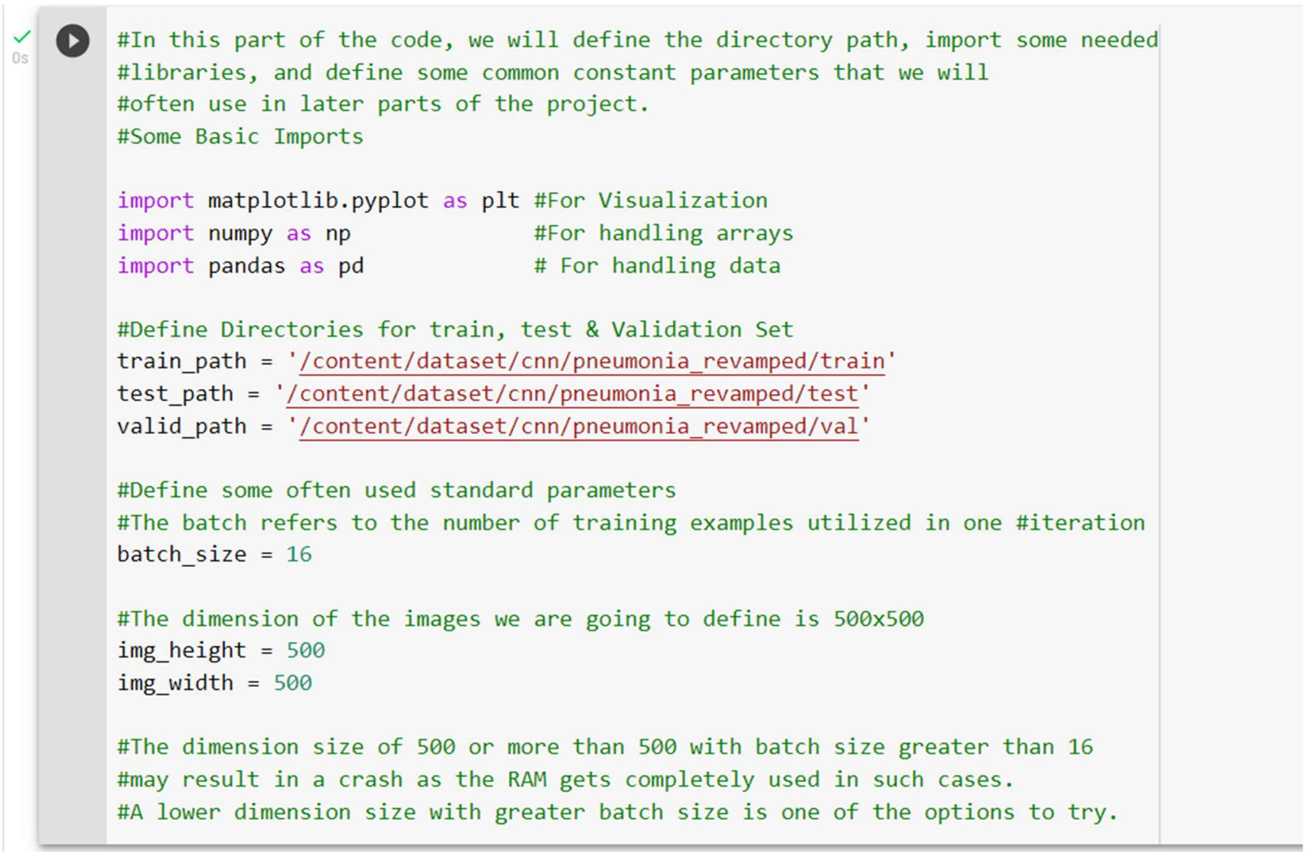
Now our dataset is ready, let's proceed!

## 4.4 Initialize

Let’s take a look at our dataset directory tree.

```
content
 └────dataset
    └────cnn
       └────pneumonia_revamped
          ├────test
          │    ├────Normal
          │    │    ├────image1.jpg
          │    │    └────image2.jpg
          │    └────Opacity
          │       ├────image1.jpg
          │       └────image2.jpg
          ├────train
          │    ├────Normal
          │    │    ├────image1.jpg
          │    │    └────image2.jpg
          │    └────Opacity
          │       ├────image1.jpg
          │       └────image2.jpg
          └────val
             ├────Normal
             │    ├────image1.jpg
             │    └────image2.jpg
             └────Opacity
                ├────image1.jpg
                └────image2.jpg
```

In this part of the code, we will define the directory path, import some needed libraries, and define some common constant parameters that we will often use in later parts of the project.



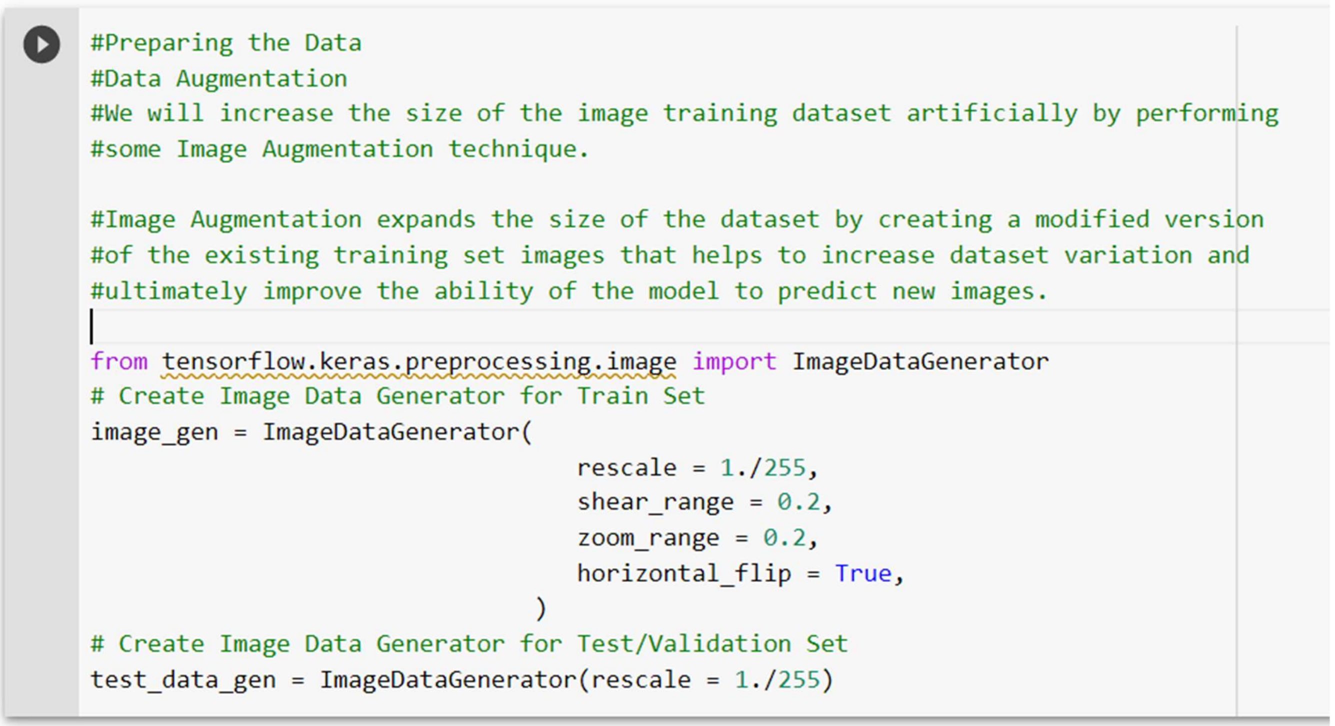
```
#In this part of the code, we will define the directory path, import some needed
#libraries, and define some common constant parameters that we will
#often use in later parts of the project.
#Some Basic Imports

import matplotlib.pyplot as plt #For Visualization
import numpy as np              #For handling arrays
import pandas as pd             # For handling data

#Define Directories for train, test & Validation Set
train_path = '/content/dataset/cnn/pneumonia_revamped/train'
test_path = '/content/dataset/cnn/pneumonia_revamped/test'
valid_path = '/content/dataset/cnn/pneumonia_revamped/val'

#Define some often used standard parameters
#The batch refers to the number of training examples utilized in one #iteration
batch_size = 16

#The dimension of the images we are going to define is 500x500
img_height = 500
img_width = 500

#The dimension size of 500 or more than 500 with batch size greater than 16
#may result in a crash as the RAM gets completely used in such cases.
#A lower dimension size with greater batch size is one of the options to try.
```

## 4.5 Preparing the Data

### 4.5.1 Data Augmentation

We will increase the size of the image training dataset artificially by performing some Image Augmentation technique.

*Image Augmentation expands the size of the dataset by creating a modified version of the existing training set images that helps to increase dataset variation and ultimately improve the ability of the model to predict new images.*


```
#Preparing the Data
#Data Augmentation
#We will increase the size of the image training dataset artificially by performing
#some Image Augmentation technique.

#Image Augmentation expands the size of the dataset by creating a modified version
#of the existing training set images that helps to increase dataset variation and
#ultimately improve the ability of the model to predict new images.

from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Create Image Data Generator for Train Set
image_gen = ImageDataGenerator(
                                  rescale = 1./255,
                                  shear_range = 0.2,
                                  zoom_range = 0.2,
                                  horizontal_flip = True,
                               )
# Create Image Data Generator for Test/Validation Set
test_data_gen = ImageDataGenerator(rescale = 1./255)
```

Using the tensorflow.keras.preprocessing.image library, for the Train Set, we created an Image Data Generator that randomly applies defined parameters to the train set and for the Test & Validation set, we're just going to rescale them to avoid manipulating the test data beforehand.

### 4.5.2 Defining some of the Image Data Generator parameters:

1. **rescale**: Each digital image is created by a pixel with a value between 0 and 255. 0 in black, 255 in white. So, rescale the scales array of the original image pixel values to be between [0,1] which makes the images contribute more equally to the overall loss. Otherwise, higher pixel range image results in greater loss and a lower learning rate should be used, lower pixel range image would require a higher learning rate.

2. **shear_range**: The shape of the image is the transformation of the shear. It fixes one axis and stretches the image at a certain angle known as the angle of the shear.

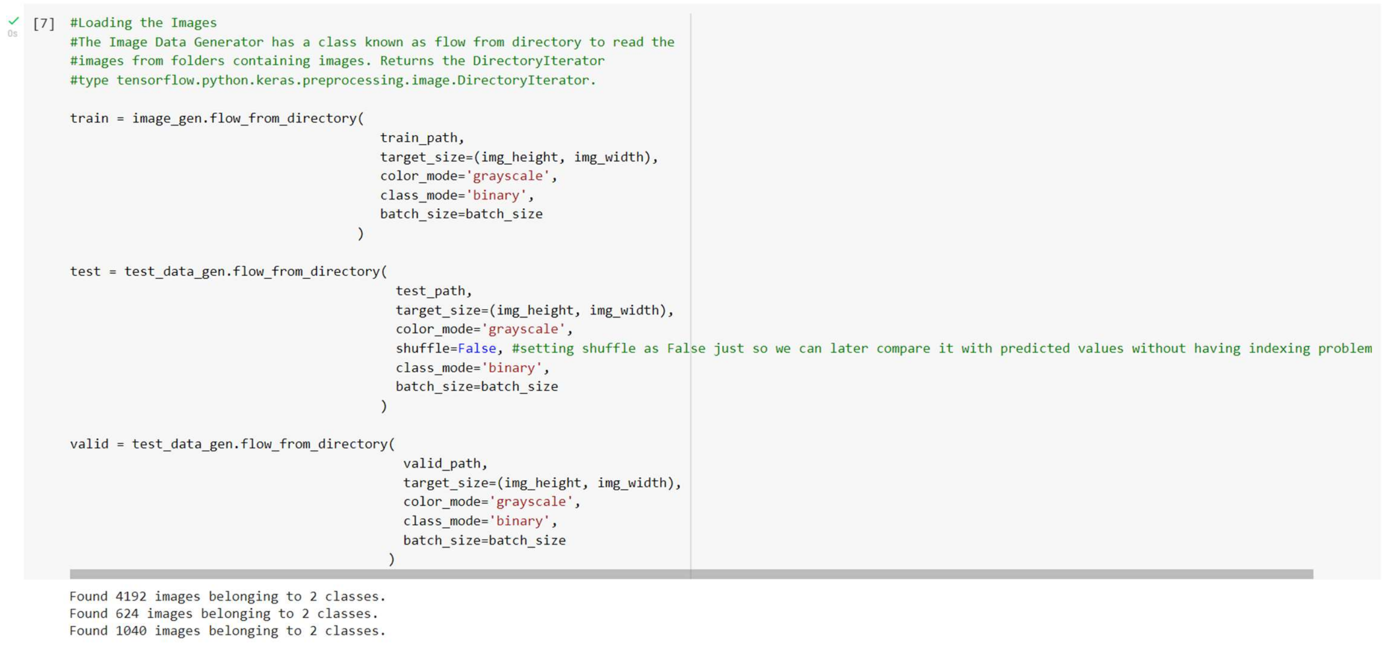
3. **zoom_range**: The image is enlarged by a zoom of less than 1.0. The image is more than 1.0 zoomed out of the picture.

4. **horizontal_flip:** Some images are flipped horizontally at random
5. **vertical_flip**: Some images are flipped vertically at random
6. **roataion_range**: Randomly, the image is rotated by some degree in the range 0 to 180.
7. **width_shift_range**: Shifts the image horizontally.
8. **height_shift_range:** Shifts the image vertically.
9. **brightness_range**: brightness of 0.0 corresponds to absolutely no brightness, and 1.0 corresponds to maximum brightness
10. **fill_mode**: Fills the missing value of the image to the nearest value or to the wrapped value or to the reflecting value.

These transformation techniques are applied randomly to the images, except for the rescale. All images have been rescaled.

## 4.6 Loading the Images

The Image Data Generator has a class known as flow from directory to read the images from folders containing images. Returns the DirectoryIterator.

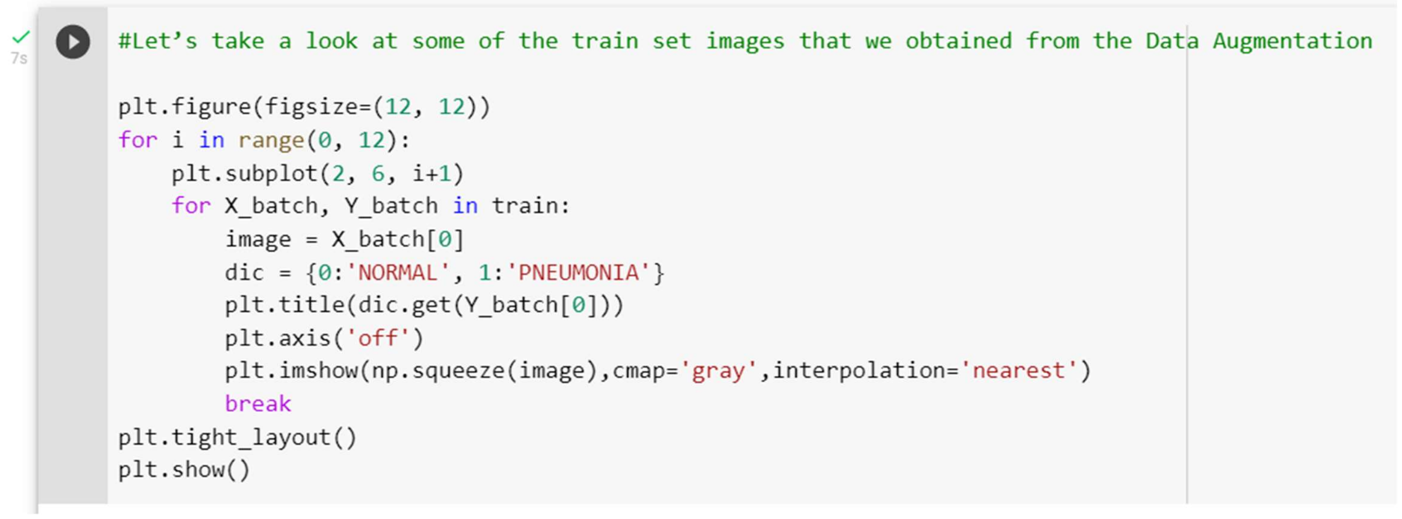

```
[7] #Loading the Images
    #The Image Data Generator has a class known as flow from directory to read the
    #images from folders containing images. Returns the DirectoryIterator
    #type tensorflow.python.keras.preprocessing.image.DirectoryIterator.

    train = image_gen.flow_from_directory(
                                        train_path,
                                        target_size=(img_height, img_width),
                                        color_mode='grayscale',
                                        class_mode='binary',
                                        batch_size=batch_size
                                     )

    test = test_data_gen.flow_from_directory(
                                          test_path,
                                          target_size=(img_height, img_width),
                                          color_mode='grayscale',
                                          shuffle=False, #setting shuffle as False just so we can later compare it with predicted values without having indexing problem
                                          class_mode='binary',
                                          batch_size=batch_size
                                       )

    valid = test_data_gen.flow_from_directory(
                                           valid_path,
                                           target_size=(img_height, img_width),
                                           color_mode='grayscale',
                                           class_mode='binary',
                                           batch_size=batch_size
                                        )

Found 4192 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
Found 1040 images belonging to 2 classes.
```

Found 4192 images belonging to 2 classes. Found 624 images belong to 2 classes. Found 1040 images belonging to 2 classes.

## Some of the parameters it takes in are defined below:

1. **directory:** The first parameter used is the path of the train, test & validation folder that we defined earlier.
2. **target_size:** The target size is the size of your input images, each image will be resized to this size. We have defined the target size earlier as 500 x 500.
3. **color_mode:** If the image is either black and white or grayscale set to "grayscale" or if the image has three colour channels set to "rgb". We're going to work with the grayscale, because it's the X-Ray images.
4. **batch_size:** Number of images to be generated by batch from the generator. We defined the batch size as 16 earlier. We choose 16 because the size of the images is too large to handle the RAM.
5. **class_mode:** Set "binary" if you only have two classes to predict, if you are not set to "categorical," if you develop an Autoencoder system, both input and output are likely to be the same image, set to "input" in this case. Here we're going to set it to binary because we've only got 2 classes to predict.

## Let's take a look at some of the train set images that we obtained from the Data Augmentation:



```
#Let's take a look at some of the train set images that we obtained from the Data Augmentation

plt.figure(figsize=(12, 12))
for i in range(0, 12):
    plt.subplot(2, 6, i+1)
    for X_batch, Y_batch in train:
        image = X_batch[0]
        dic = {0:'NORMAL', 1:'PNEUMONIA'}
        plt.title(dic.get(Y_batch[0]))
        plt.axis('off')
        plt.imshow(np.squeeze(image),cmap='gray',interpolation='nearest')
        break
plt.tight_layout()
plt.show()
```

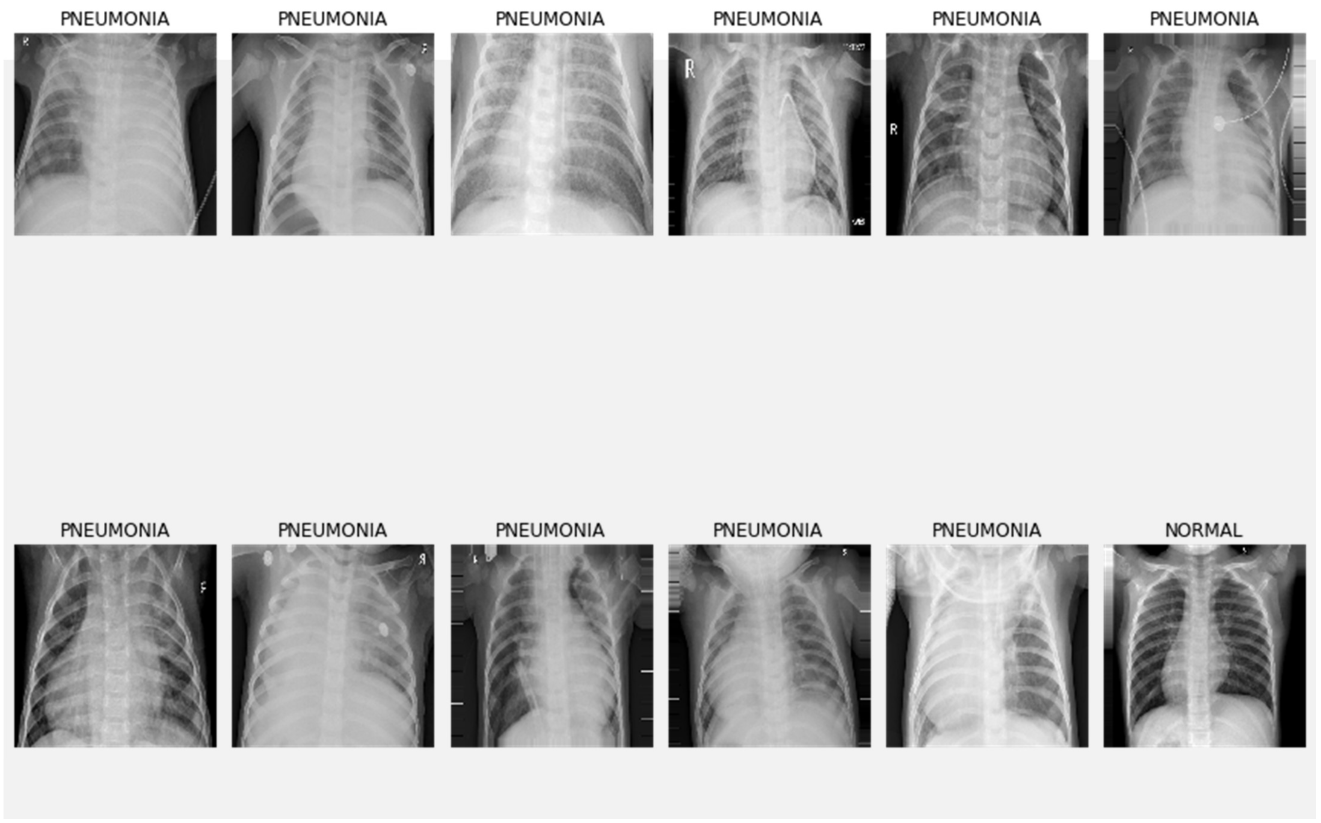


Figure 4.1: X-Ray images after data augmentation

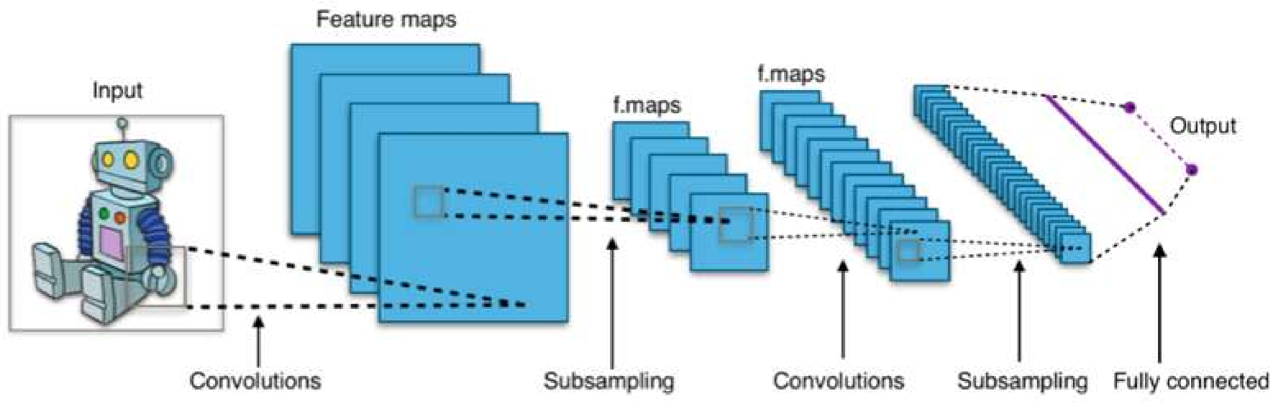
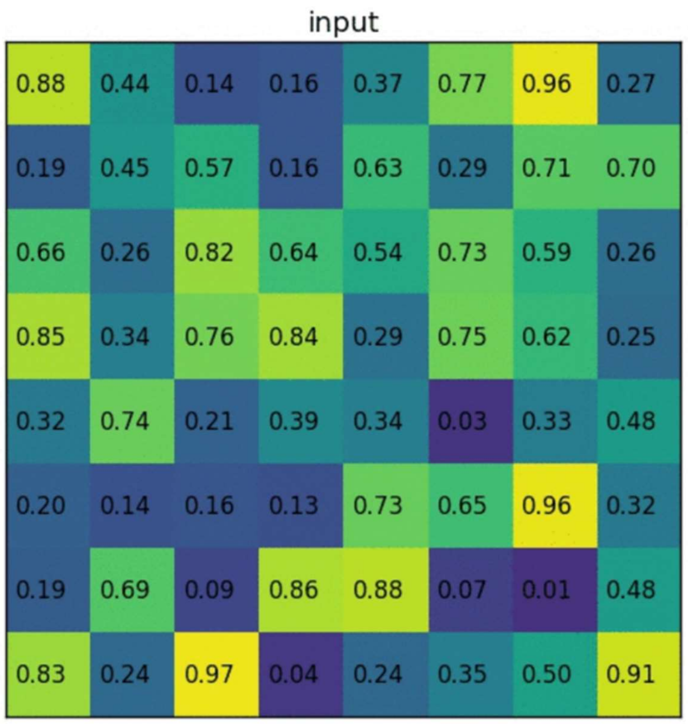


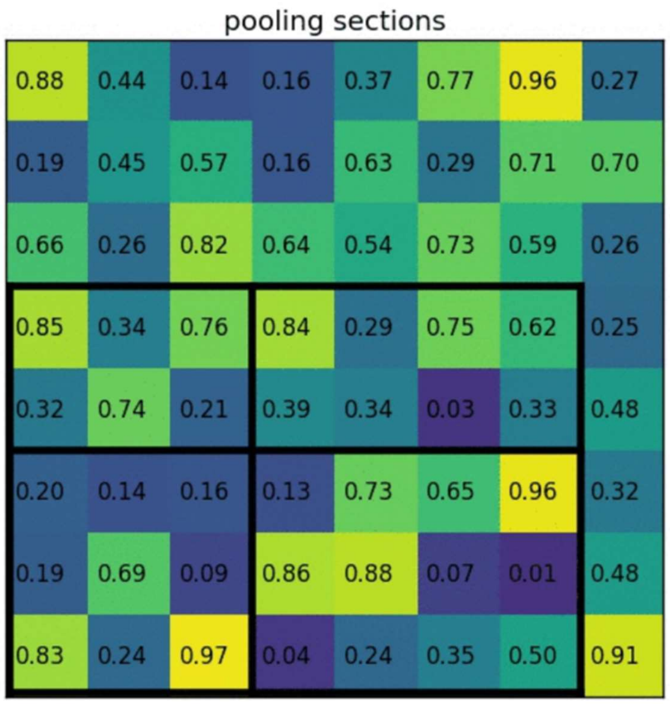
Figure 4.2: Convolutional Neural Network

**What is CNN in one sentence:** It an artificial neural network that has the ability to pin point or detect patterns in the images.

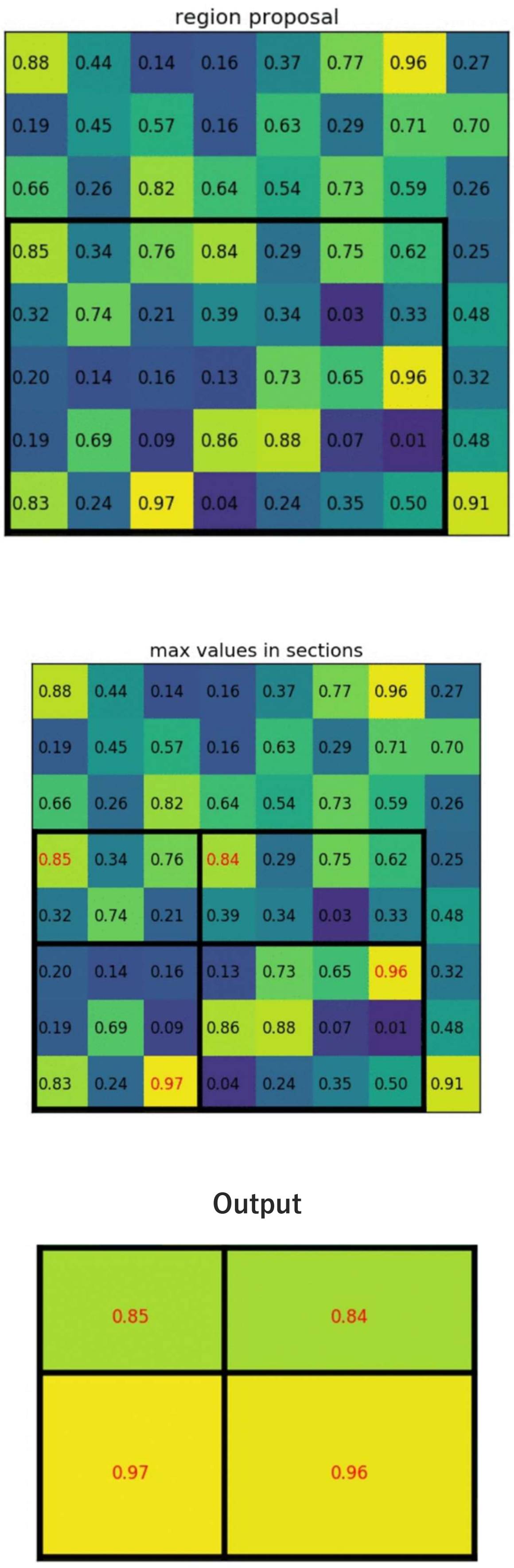
## Step by step Max Pooling Process:



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0.88 | 0.44 | 0.14 | 0.16 | 0.37 | 0.77 | 0.96 | 0.27 |
| 0.19 | 0.45 | 0.57 | 0.16 | 0.63 | 0.29 | 0.71 | 0.70 |
| 0.66 | 0.26 | 0.82 | 0.64 | 0.54 | 0.73 | 0.59 | 0.26 |
| 0.85 | 0.34 | 0.76 | 0.84 | 0.29 | 0.75 | 0.62 | 0.25 |
| 0.32 | 0.74 | 0.21 | 0.39 | 0.34 | 0.03 | 0.33 | 0.48 |
| 0.20 | 0.14 | 0.16 | 0.13 | 0.73 | 0.65 | 0.96 | 0.32 |
| 0.19 | 0.69 | 0.09 | 0.86 | 0.88 | 0.07 | 0.01 | 0.48 |
| 0.83 | 0.24 | 0.97 | 0.04 | 0.24 | 0.35 | 0.50 | 0.91 |



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0.88 | 0.44 | 0.14 | 0.16 | 0.37 | 0.77 | 0.96 | 0.27 |
| 0.19 | 0.45 | 0.57 | 0.16 | 0.63 | 0.29 | 0.71 | 0.70 |
| 0.66 | 0.26 | 0.82 | 0.64 | 0.54 | 0.73 | 0.59 | 0.26 |
| 0.85 | 0.34 | 0.76 | 0.84 | 0.29 | 0.75 | 0.62 | 0.25 |
| 0.32 | 0.74 | 0.21 | 0.39 | 0.34 | 0.03 | 0.33 | 0.48 |
| 0.20 | 0.14 | 0.16 | 0.13 | 0.73 | 0.65 | 0.96 | 0.32 |
| 0.19 | 0.69 | 0.09 | 0.86 | 0.88 | 0.07 | 0.01 | 0.48 |
| 0.83 | 0.24 | 0.97 | 0.04 | 0.24 | 0.35 | 0.50 | 0.91 |



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0.88 | 0.44 | 0.14 | 0.16 | 0.37 | 0.77 | 0.96 | 0.27 |
| 0.19 | 0.45 | 0.57 | 0.16 | 0.63 | 0.29 | 0.71 | 0.70 |
| 0.66 | 0.26 | 0.82 | 0.64 | 0.54 | 0.73 | 0.59 | 0.26 |
| 0.85 | 0.34 | 0.76 | 0.84 | 0.29 | 0.75 | 0.62 | 0.25 |
| 0.32 | 0.74 | 0.21 | 0.39 | 0.34 | 0.03 | 0.33 | 0.48 |
| 0.20 | 0.14 | 0.16 | 0.13 | 0.73 | 0.65 | 0.96 | 0.32 |
| 0.19 | 0.69 | 0.09 | 0.86 | 0.88 | 0.07 | 0.01 | 0.48 |
| 0.83 | 0.24 | 0.97 | 0.04 | 0.24 | 0.35 | 0.50 | 0.91 |

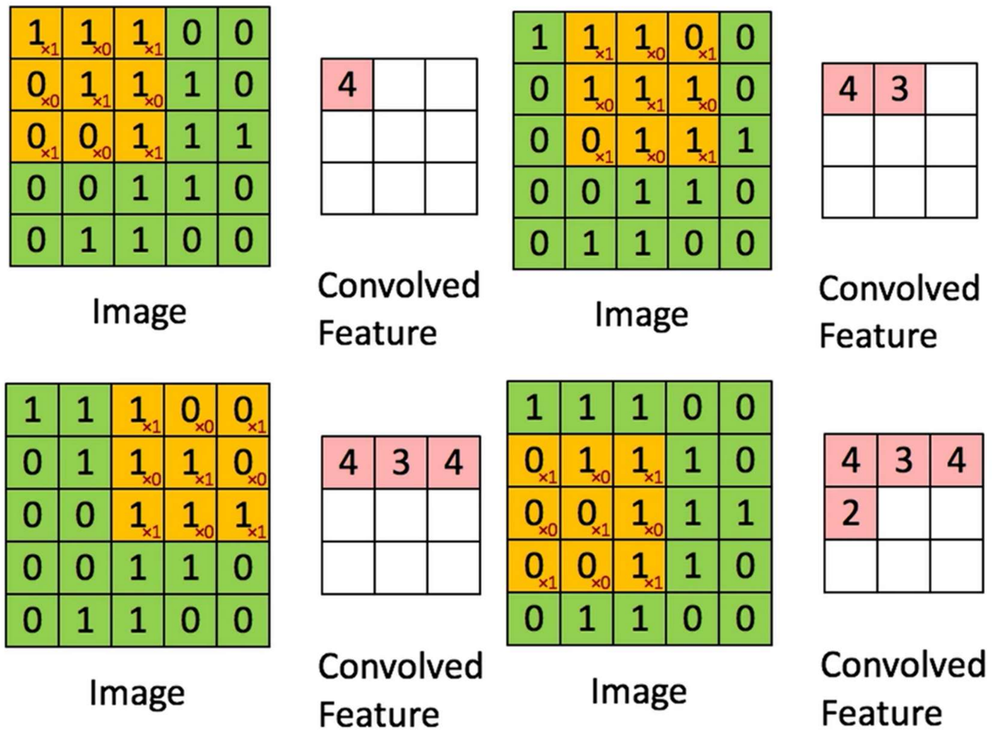
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0.88 | 0.44 | 0.14 | 0.16 | 0.37 | 0.77 | 0.96 | 0.27 |
| 0.19 | 0.45 | 0.57 | 0.16 | 0.63 | 0.29 | 0.71 | 0.70 |
| 0.66 | 0.26 | 0.82 | 0.64 | 0.54 | 0.73 | 0.59 | 0.26 |
| 0.85 | 0.34 | 0.76 | 0.84 | 0.29 | 0.75 | 0.62 | 0.25 |
| 0.32 | 0.74 | 0.21 | 0.39 | 0.34 | 0.03 | 0.33 | 0.48 |
| 0.20 | 0.14 | 0.16 | 0.13 | 0.73 | 0.65 | 0.96 | 0.32 |
| 0.19 | 0.69 | 0.09 | 0.86 | 0.88 | 0.07 | 0.01 | 0.48 |
| 0.83 | 0.24 | 0.97 | 0.04 | 0.24 | 0.35 | 0.50 | 0.91 |

| | |
|---|---|
| 0.85 | 0.84 |
| 0.97 | 0.96 |

Figure 4.3: Max pooling process

## 4.7 Explain what's going on inside a CNN architecture:

CNN architecture is based on layers of convolution. The convolution layers receive input and transform the data from the image and pass it as input to the next layer. The transformation is known as the operation of convolution. We need to define the number of filters for each convolution layer. These filters detect patterns such as edges, shapes, curves, objects, textures, or even colors. The more sophisticated patterns or objects it detects are more deeply layered. In essence, filters are image kernels that we can define as 3x3 or 4x4, which is a small matrix applied to an image as a whole. We will use Pooling layer together with Convolution layer as well as the goal is to down-sample an input representation (image), decrease its dimensionality by retaining the maximum value (activated features) in the sub regions binding. The number of pixels moving across the input matrix is called Stride. When the stride is 1 we move the filter to 1 pixel at a time. When the stride is 2 then we move the filter to 2 pixels at a time, and so on. Larger filter sizes and strides may be used to reduce the size of a large image to a moderate size. Read the steps in row major order.



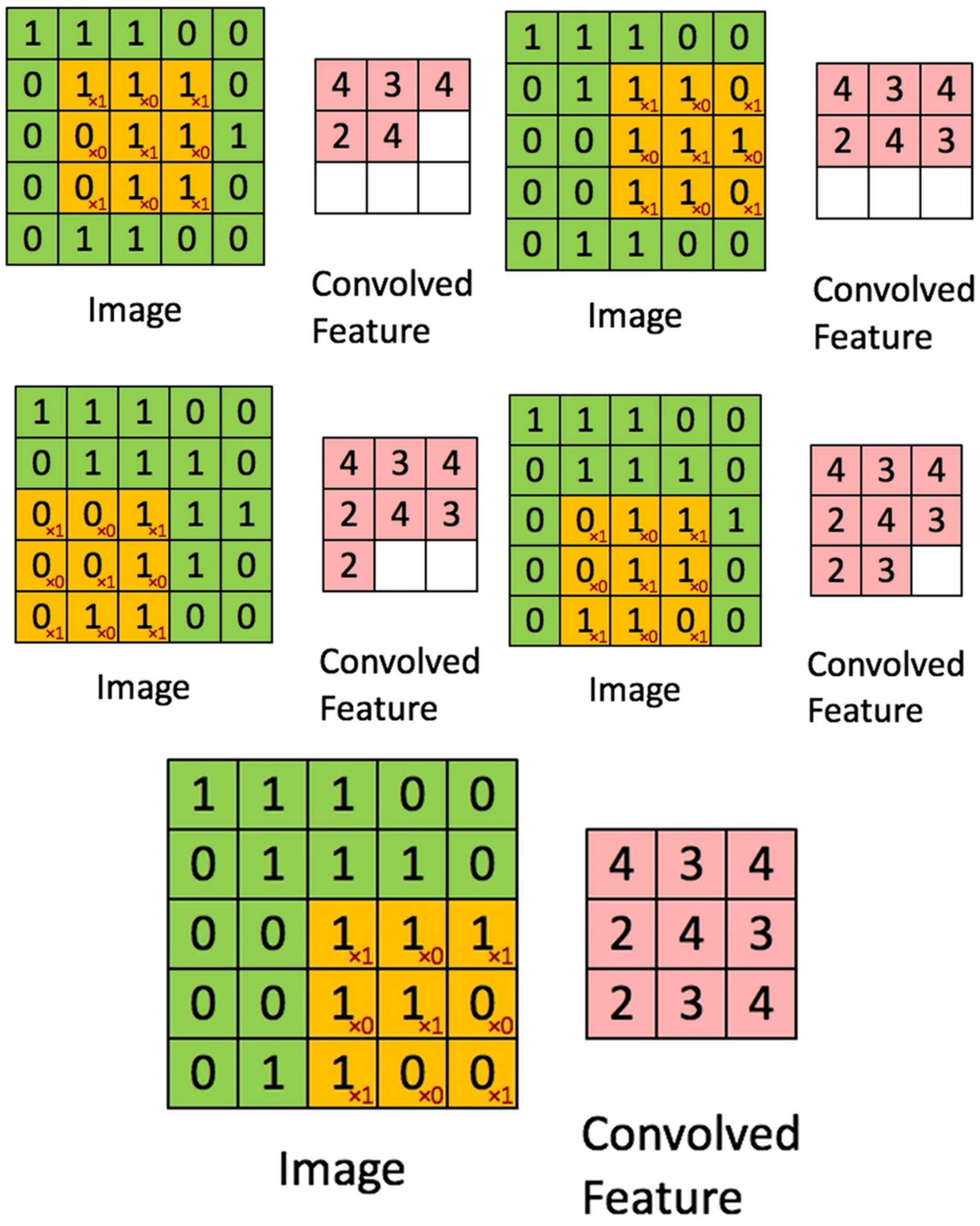


**Figure 4.4: Convolution process**

all these complex mathematical operations are performed behind the scenes, all we need to do is define hyper parameters and layers. You can refer to the links in the reference section if you love math and want to see how these *mathemagicical* operations work.

## Necessary imports:

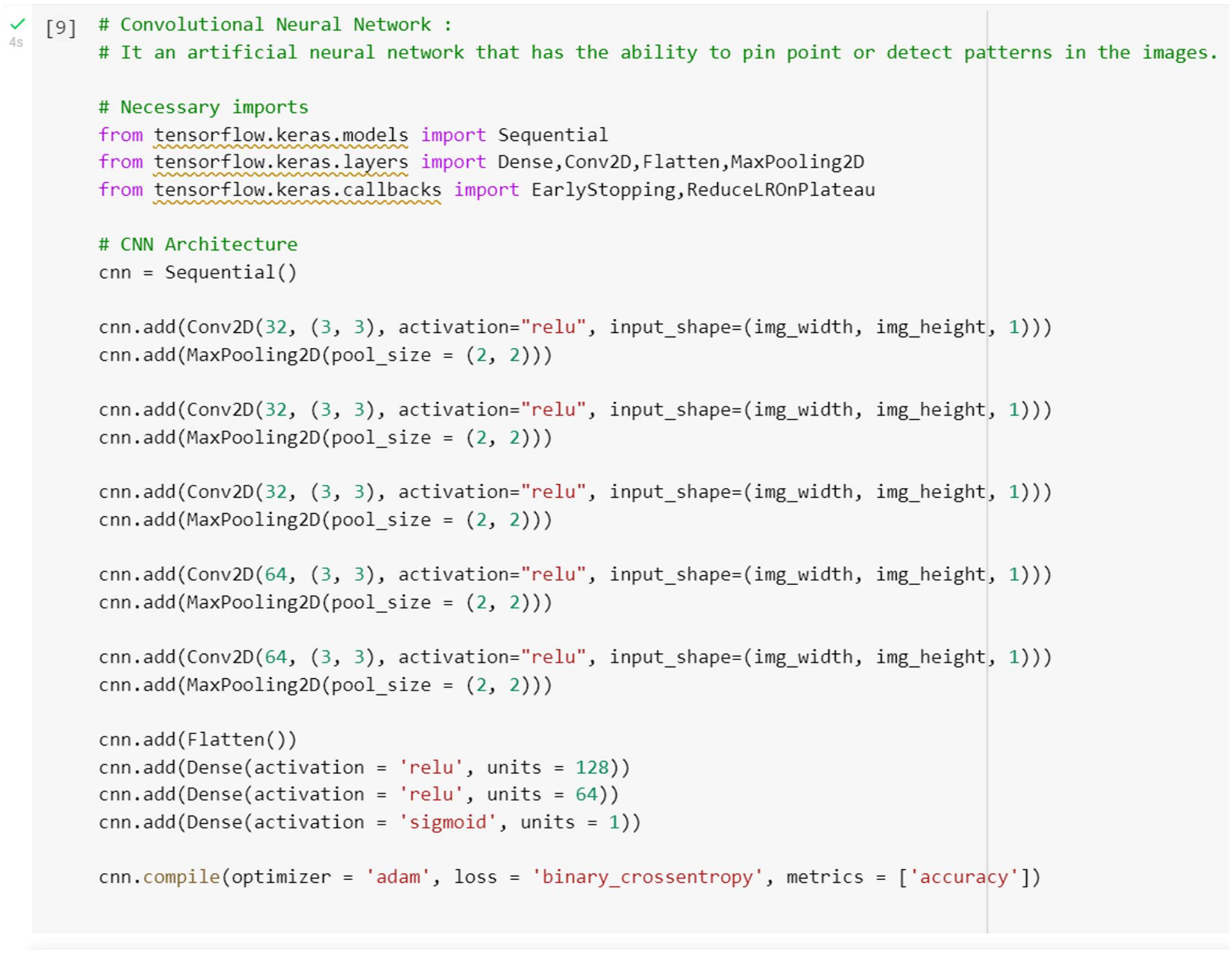
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Conv2D,Flatten,MaxPooling2D
from tensorflow.keras.callbacks import EarlyStopping,ReduceLROnPlateau
```

## 4.8 Building CNN Architecture:

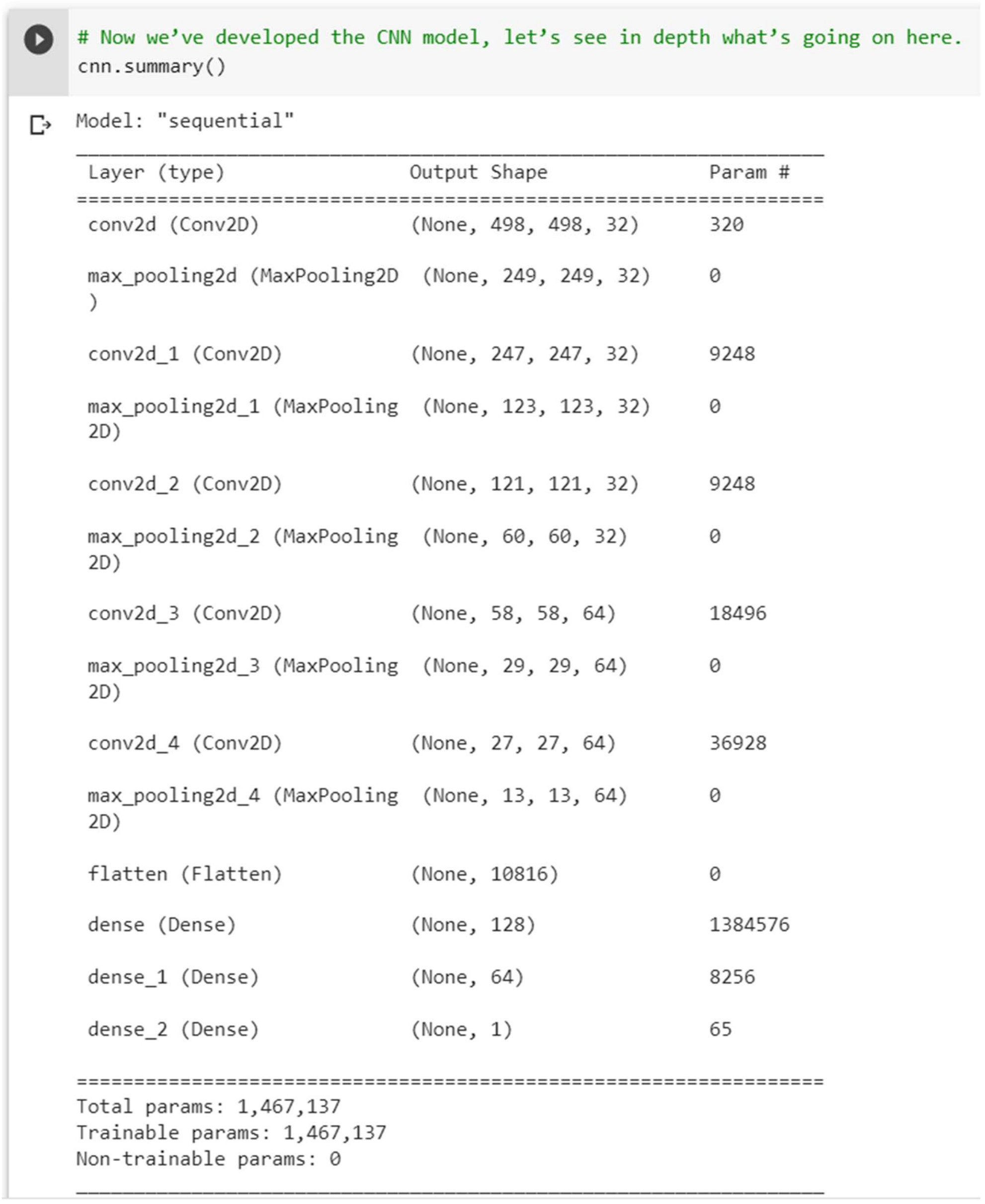
Things to note before starting to build a CNN model:

1. Always begin with a lower filter value such as 32 and begin to increase it layer wise.
2. Construct the model with a layer of Conv2D followed by a layer of MaxPooling.
3. The kernel_size is preferred to be odd number like 3x3.
4. Tanh, relu, etc. can be used for activation function, but relu is the most preferred activation function.
5. `input_shape` takes in image width & height with last dimension as color channel.
6. Flattening the input after CNN layers and adding ANN layers.
7. Use activation function as softmax for the last layer If the problem is more than 2 classes, define units as the total number of classes and use sigmoid for binary classification and set unit to 1.

Note :- You can always experiment with these hyperparameters as there is no fixed value on which we can settle.



```
# Convolutional Neural Network :
# It an artificial neural network that has the ability to pin point or detect patterns in the images.

# Necessary imports
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Conv2D,Flatten,MaxPooling2D
from tensorflow.keras.callbacks import EarlyStopping,ReduceLROnPlateau

# CNN Architecture
cnn = Sequential()

cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(img_width, img_height, 1)))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(img_width, img_height, 1)))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(img_width, img_height, 1)))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(64, (3, 3), activation="relu", input_shape=(img_width, img_height, 1)))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(64, (3, 3), activation="relu", input_shape=(img_width, img_height, 1)))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Flatten())
cnn.add(Dense(activation = 'relu', units = 128))
cnn.add(Dense(activation = 'relu', units = 64))
cnn.add(Dense(activation = 'sigmoid', units = 1))

cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```



```
# Now we’ve developed the CNN model, let’s see in depth what’s going on here.
cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 498, 498, 32)      320

 max_pooling2d (MaxPooling2D  (None, 249, 249, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 247, 247, 32)      9248

 max_pooling2d_1 (MaxPooling  (None, 123, 123, 32)     0
 2D)

 conv2d_2 (Conv2D)           (None, 121, 121, 32)      9248

 max_pooling2d_2 (MaxPooling  (None, 60, 60, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 58, 58, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 29, 29, 64)       0
 2D)

 conv2d_4 (Conv2D)           (None, 27, 27, 64)        36928

 max_pooling2d_4 (MaxPooling  (None, 13, 13, 64)       0
 2D)

 flatten (Flatten)           (None, 10816)             0

 dense (Dense)               (None, 128)               1384576

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 1,467,137
Trainable params: 1,467,137
Non-trainable params: 0
_________________________________________________________________
```

**Figure 4.5: Architecture of the CNN model**

The input shape of the images are (500, 500, 1) as we defined the height & width earlier. And the 1 represents the color channel as the images are grayscale the color channel for it is 1 and for “rgb” images it is 3.

(none, 500, 500, 1) Over here Keras adds an extra dimension none since batch size can vary.

In First Conv2d layer Convolution operation on image of (500,500) with a (3,3) kernel size with strides and dilation set 1 by default and padding set to ‘valid’, it spits out output size of (500-3+1 , 500-3+1 ) = (498,498) And the number of filters we defined is 32, the output shape is now(None,498,498,32)

Now in the first Max Pooling layer, we have defined the kernel size as (2,2) and strides are by default (2,2) applying that to input of image size of (498,498) we get ((498–2//2)+1,(498–2//2)+1))= (249,249)

The Flatten layer takes all of the pixels along all channels and creates a 1D vector without considering batchsize. The input of (13, 13, 64) is therefore flattened to (13*13*64) = 10816 values.

The parameter value is calculated by (kernel_height * kernel_width * input_channels * output_channels) + (output_channels) which gives (3*3*1*32)+(32) = 320 in first layer.

The rectified linear activation function or short-term ReLU is a piecewise linear function that outputs the input directly if it is positive, otherwise it outputs zero. The rectified linear activation function overcomes the problem of vanishing gradients, allowing models to learn faster and perform better.

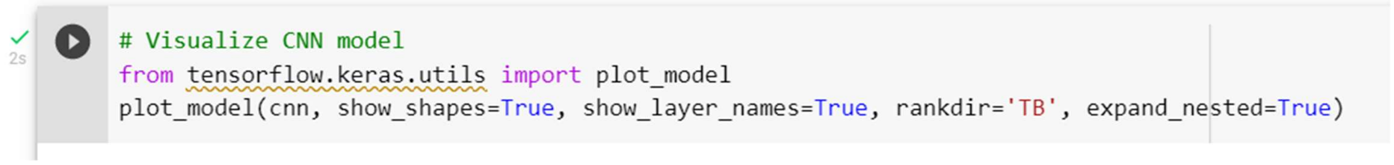
Padding: "SAME": output size is the **same** as input size. This requires the filter window to slip outside input map, hence the need to pad."VALID": Filter window stays at **valid** position inside input map, so output size shrinks by filter_size - 1. No padding occurs.

Activation function: Simply put, activation is a function that is added to an artificial neural network to help the network learn complex patterns in the data. When comparing with a neuron-based model in our brains, the activation function is at the end of the day to decide what to do with the next neuron. Since the classification is between 2 classes we are going to use sigmoid activation function for last layer which returns value in the range of 0 to 1. For more than 2 classes we can use softmax activation function.

## Defining Model Compile:

- Learning Rate — while training the aim for stochastic gradient descent is to minimize loss among actual and predicted values of training set. Path to minimize loss takes several steps. Adam is an adaptive learning rate method, which means, it computes individual learning rates for different parameters.
- loss function — Since it is a binary classification, we will use binary crossentropy during training for evaluation of losses. We would have gone for categorical crossentropy if there were more than 4 classes.
- metrics — accuracy — Calculate how often actual labels are equal to predictions. It will measure the loss and accuracy of training and validation.

## 4.9 Visualize CNN model:



```
# Visualize CNN model
from tensorflow.keras.utils import plot_model
plot_model(cnn, show_shapes=True, show_layer_names=True, rankdir='TB', expand_nested=True)
```

Output of above two lines of code is shown below. With these two lines of code, we are seeing the layers of the model we build just before.

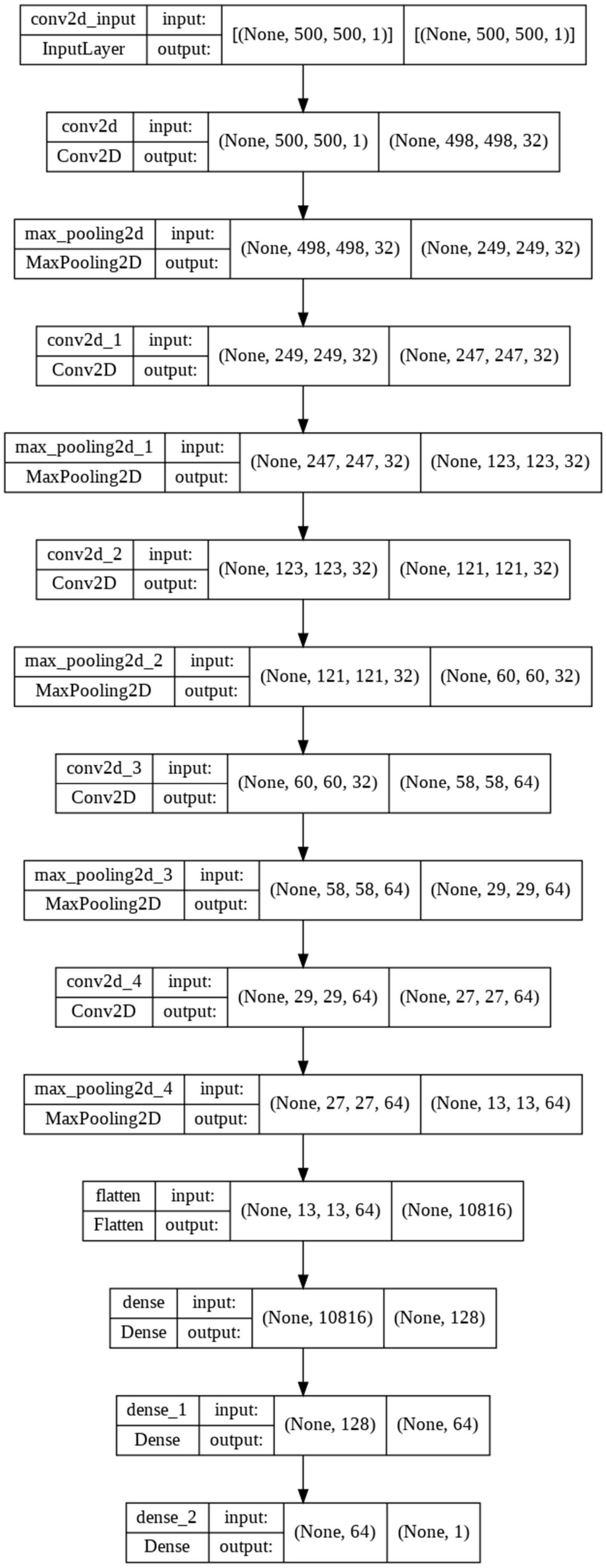


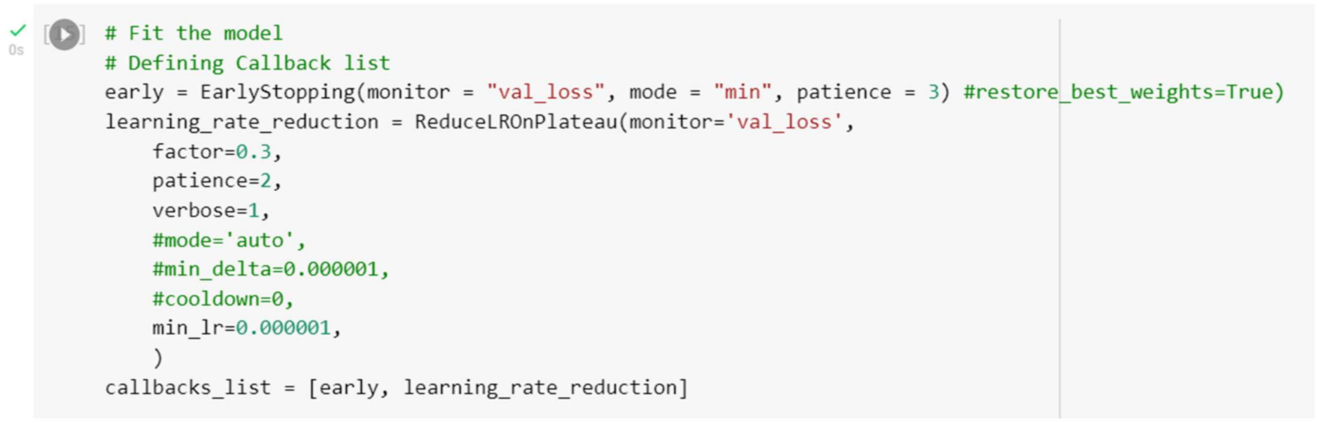
Figure 4.6: Plotting the CNN architecture

## 4.10 Fit the model

### 4.10.1 Defining Callback list

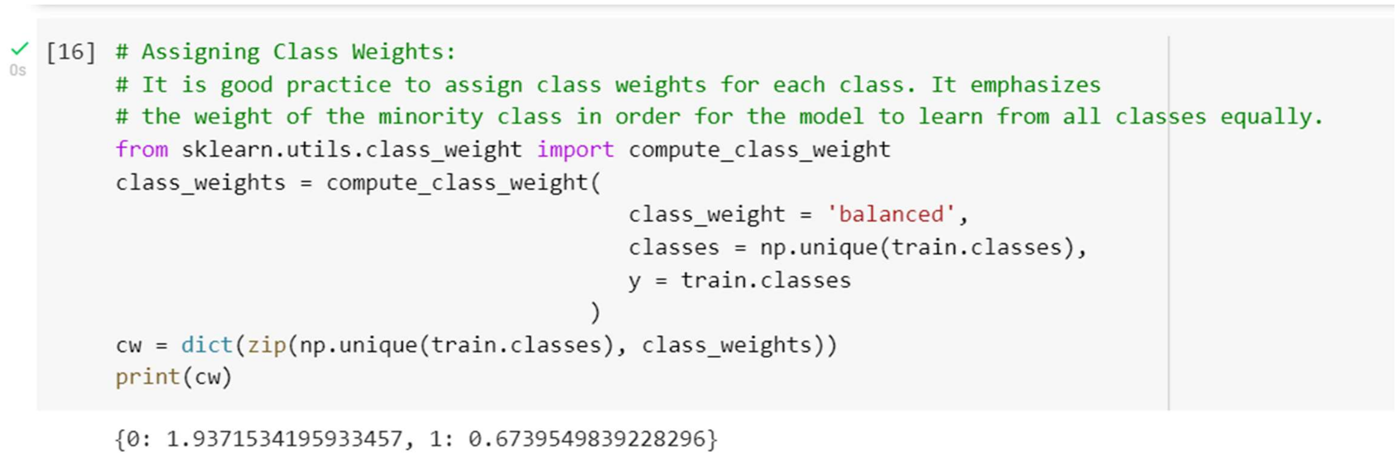
EarlyStopping is called to stop the epochs based on some metric(monitor) and conditions (mode, patience) . It helps to avoid overfitting the model. Over here we are telling to stop based on val_loss metric, we need it to be minimum. patience says that after a minimum val_loss is achieved then after that in next iterations if the val_loss increases in any the 3 iterations then training will stop at that epoch.

Reduce learning rate when a metric has stopped improving. Models often benefit from reducing the learning rate by a factor of 2–10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.


```
# Fit the model
# Defining Callback list
early = EarlyStopping(monitor = "val_loss", mode = "min", patience = 3) #restore_best_weights=True)
learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss',
    factor=0.3,
    patience=2,
    verbose=1,
    #mode='auto',
    #min_delta=0.000001,
    #cooldown=0,
    min_lr=0.000001,
    )
callbacks_list = [early, learning_rate_reduction]
```

### 4.10.2 Assigning Class Weights

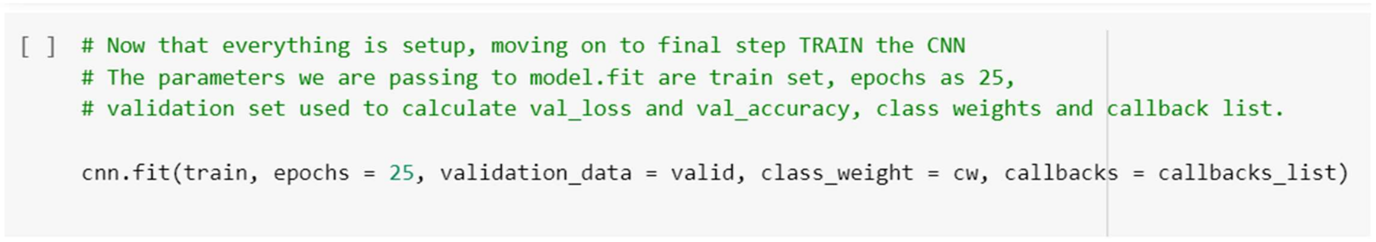
It is good practice to assign class weights for each class. It emphasizes the weight of the minority class in order for the model to learn from all classes equally.



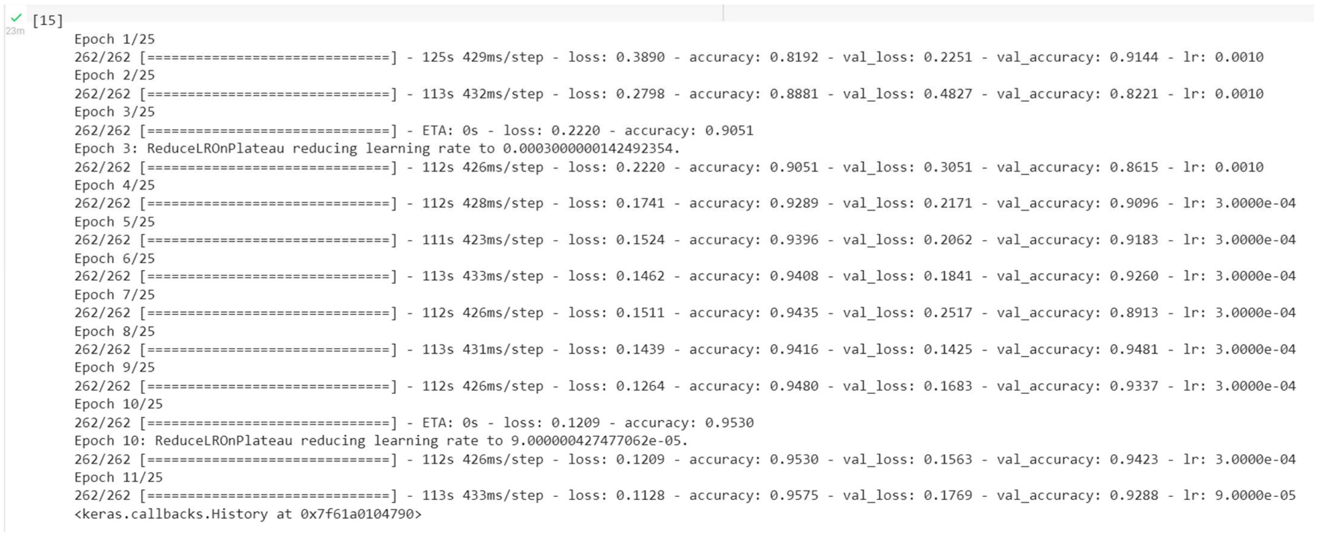
```
[16] # Assigning Class Weights:
     # It is good practice to assign class weights for each class. It emphasizes
     # the weight of the minority class in order for the model to learn from all classes equally.
     from sklearn.utils.class_weight import compute_class_weight
     class_weights = compute_class_weight(
                                             class_weight = 'balanced',
                                             classes = np.unique(train.classes),
                                             y = train.classes
                                         )
     cw = dict(zip(np.unique(train.classes), class_weights))
     print(cw)

     {0: 1.9371534195933457, 1: 0.6739549839228296}
```

Now that everything is setup, moving on to final step TRAINING 💪

The parameters we are passing to model.fit are train set, epochs as 25, validation set used to calculate val_loss and val_accuracy, class weights and callback list.



```
[ ] # Now that everything is setup, moving on to final step TRAIN the CNN
    # The parameters we are passing to model.fit are train set, epochs as 25,
    # validation set used to calculate val_loss and val_accuracy, class weights and callback list.

    cnn.fit(train, epochs = 25, validation_data = valid, class_weight = cw, callbacks = callbacks_list)
```



```
[15]
    Epoch 1/25
    262/262 [==============================] - 125s 429ms/step - loss: 0.3890 - accuracy: 0.8192 - val_loss: 0.2251 - val_accuracy: 0.9144 - lr: 0.0010
    Epoch 2/25
    262/262 [==============================] - 113s 432ms/step - loss: 0.2798 - accuracy: 0.8881 - val_loss: 0.4827 - val_accuracy: 0.8221 - lr: 0.0010
    Epoch 3/25
    262/262 [==============================] - ETA: 0s - loss: 0.2220 - accuracy: 0.9051
    Epoch 3: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
    262/262 [==============================] - 112s 426ms/step - loss: 0.2220 - accuracy: 0.9051 - val_loss: 0.3051 - val_accuracy: 0.8615 - lr: 0.0010
    Epoch 4/25
    262/262 [==============================] - 112s 428ms/step - loss: 0.1741 - accuracy: 0.9289 - val_loss: 0.2171 - val_accuracy: 0.9096 - lr: 3.0000e-04
    Epoch 5/25
    262/262 [==============================] - 111s 423ms/step - loss: 0.1524 - accuracy: 0.9396 - val_loss: 0.2062 - val_accuracy: 0.9183 - lr: 3.0000e-04
    Epoch 6/25
    262/262 [==============================] - 113s 433ms/step - loss: 0.1462 - accuracy: 0.9408 - val_loss: 0.1841 - val_accuracy: 0.9260 - lr: 3.0000e-04
    Epoch 7/25
    262/262 [==============================] - 112s 426ms/step - loss: 0.1511 - accuracy: 0.9435 - val_loss: 0.2517 - val_accuracy: 0.8913 - lr: 3.0000e-04
    Epoch 8/25
    262/262 [==============================] - 113s 431ms/step - loss: 0.1439 - accuracy: 0.9416 - val_loss: 0.1425 - val_accuracy: 0.9481 - lr: 3.0000e-04
    Epoch 9/25
    262/262 [==============================] - 112s 426ms/step - loss: 0.1264 - accuracy: 0.9480 - val_loss: 0.1683 - val_accuracy: 0.9337 - lr: 3.0000e-04
    Epoch 10/25
    262/262 [==============================] - ETA: 0s - loss: 0.1209 - accuracy: 0.9530
    Epoch 10: ReduceLROnPlateau reducing learning rate to 9.000000427477062e-05.
    262/262 [==============================] - 112s 426ms/step - loss: 0.1209 - accuracy: 0.9530 - val_loss: 0.1563 - val_accuracy: 0.9423 - lr: 3.0000e-04
    Epoch 11/25
    262/262 [==============================] - 113s 433ms/step - loss: 0.1128 - accuracy: 0.9575 - val_loss: 0.1769 - val_accuracy: 0.9288 - lr: 9.0000e-05
    <keras.callbacks.History at 0x7f61a0104790>
```

**Figure 4.7: Training the model**

Looks like the EarlyStopping stopped at 11th epoch at val_loss =17.69% and val_accuracy = 92.88%.

## 4.11 Evaluate the model

Let’s visualize the progress of all metrics throughout the total epochs lifetime

```
[16] #Evaluate the model
     #Let’s visualize the progress of all metrics throughout the total epochs lifetime

     pd.DataFrame(cnn.history.history).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6118056fd0>
```

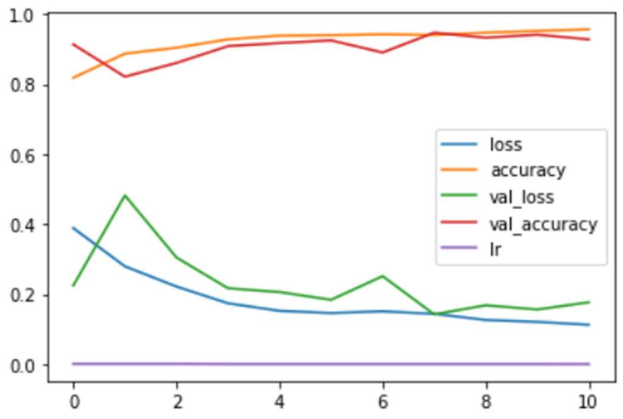


**Figure 4.8: Progress of various parameters throughout all the epochs**

```
[ ] # Accuracy  of the model:
    test_accu = cnn.evaluate(test)
    print('The testing accuracy is :',test_accu[1]*100, '%')
```

```
39/39 [==============================] - 7s 167ms/step - loss: 0.2740 - accuracy: 0.9199
The testing accuracy is : 91.98718070983887 %
```

The accuracy we are getting on Test dataset is of ~92%.

Let’s predict the test dataset and look at some of the performance measurement metrics in detail to evaluate our model.

```
[ ]  # Let’s predict the test dataset and look at some of the performance measurement
     # metrics in detail to evaluate our model.
     preds = cnn.predict(test, verbose = 1)

     39/39 [==============================] - 5s 136ms/step
```

Since the activation function of the last layer is sigmoid, the model gives prediction in the 0 to 1 range and not an exact classification as 0 or 1. So we categorise all the values in the 0.5 to 1 range as 0 and less than 0.5 as 1. Note(0 denotes a normal case and 1 denotes a case of pneumonia)

```
[ ]  # Since the activation function of the last layer is sigmoid, the model gives
     # prediction in the 0 to 1 range and not an exact classification as 0 or 1.
     # So we categorise all the values in the 0.5 to 1 range as 0 and less than 0.5 as 1.
     # Note(0 denotes a normal case and 1 denotes a case of pneumonia)
     predictions = preds.copy()
     predictions[predictions <= 0.5] = 0
     predictions[predictions > 0.5] = 1
```

## 4.12 Confusion Matrix

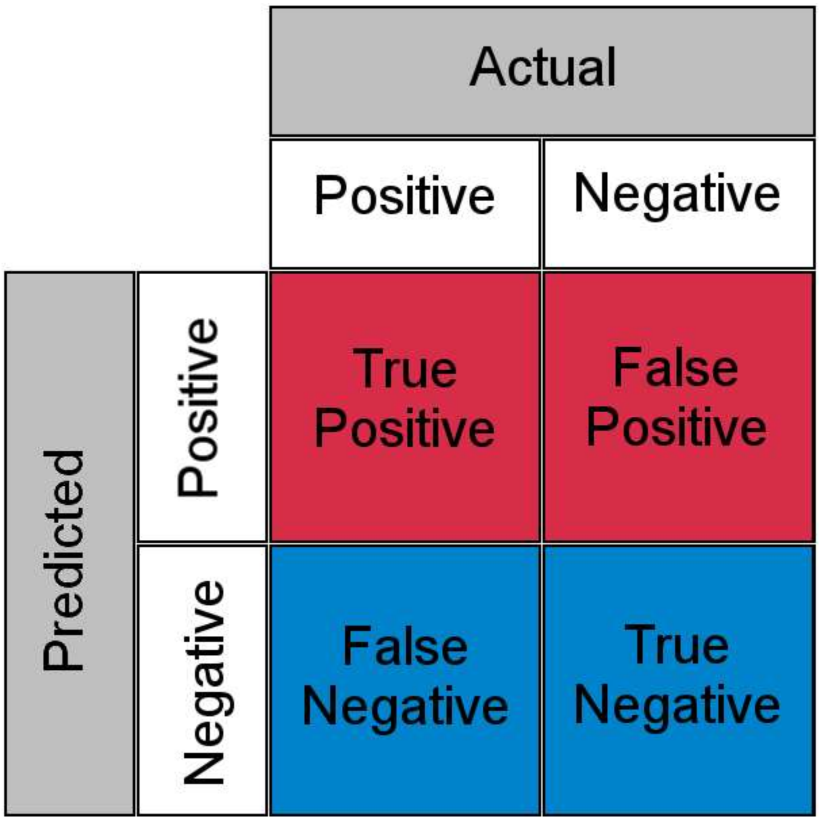
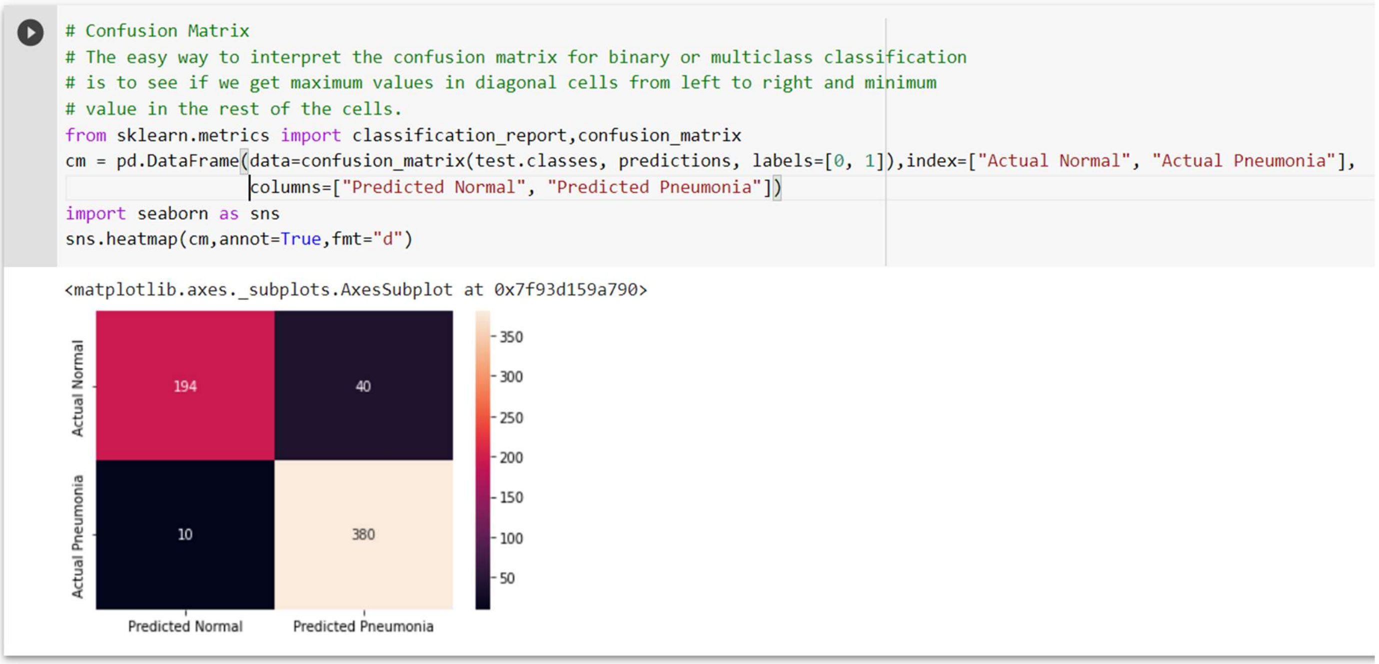


Figure 4.9: Confusion Matrix

Let's interpret the output of the confusion matrix. The upper left (TP) denotes the number of images correctly predicted as normal cases and the bottom right (TN) denotes the correctly predicted number of images as cases of pneumonia. As Pneumonia case, the upper right denotes the number of incorrectly predicted images but were actually normal cases and the lower left denotes the number of incorrectly predicted Normal case images but were actually Pneumonia case.

## Still Confused with Confusion matrix ??

The easy way to interpret the confusion matrix for binary or multiclass classification is to see if we get maximum values in diagonal cells from left to right and minimum value in the rest of the cells.



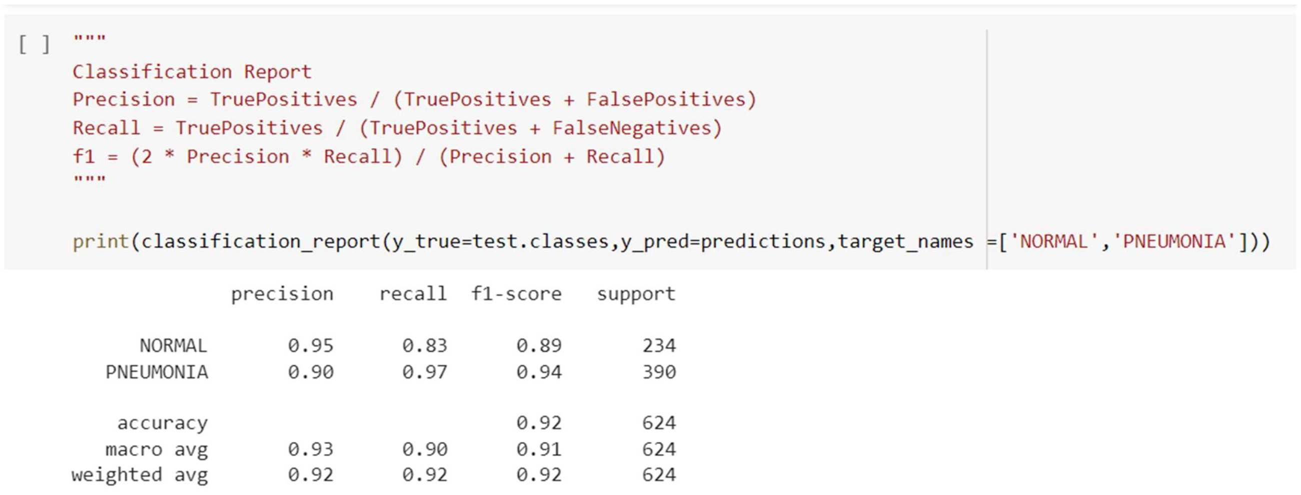
```
# Confusion Matrix
# The easy way to interpret the confusion matrix for binary or multiclass classification
# is to see if we get maximum values in diagonal cells from left to right and minimum
# value in the rest of the cells.
from sklearn.metrics import classification_report,confusion_matrix
cm = pd.DataFrame(data=confusion_matrix(test.classes, predictions, labels=[0, 1]),index=["Actual Normal", "Actual Pneumonia"],
                  columns=["Predicted Normal", "Predicted Pneumonia"])
import seaborn as sns
sns.heatmap(cm,annot=True,fmt="d")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f93d159a790>
```

Figure 4.10: Confusion Matrix for our model

## 4.13 Classification Report

- Precision = TruePositives / (TruePositives + FalsePositives)
- Recall = TruePositives / (TruePositives + FalseNegatives)
- F1 = (2 * Precision * Recall) / (Precision + Recall)

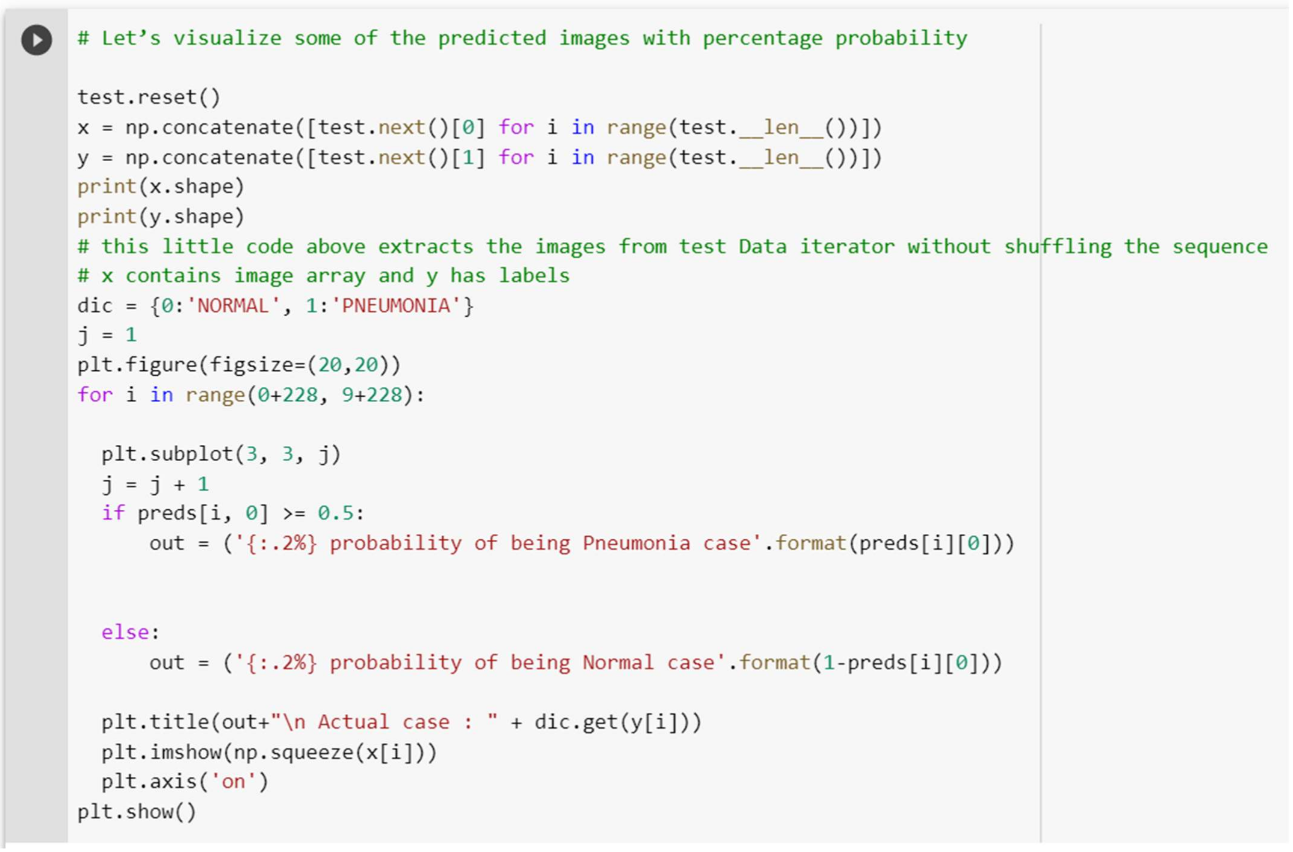


```
[ ] """
    Classification Report
    Precision = TruePositives / (TruePositives + FalsePositives)
    Recall = TruePositives / (TruePositives + FalseNegatives)
    f1 = (2 * Precision * Recall) / (Precision + Recall)
    """

    print(classification_report(y_true=test.classes,y_pred=predictions,target_names =['NORMAL','PNEUMONIA']))
```

```
              precision    recall  f1-score   support

      NORMAL       0.95      0.83      0.89       234
   PNEUMONIA       0.90      0.97      0.94       390

    accuracy                           0.92       624
   macro avg       0.93      0.90      0.91       624
weighted avg       0.92      0.92      0.92       624
```

Figure 4.11: Classification Report

## Let’s visualize some of the predicted images with percentage probability



```
# Let’s visualize some of the predicted images with percentage probability

test.reset()
x = np.concatenate([test.next()[0] for i in range(test.__len__())])
y = np.concatenate([test.next()[1] for i in range(test.__len__())])
print(x.shape)
print(y.shape)
# this little code above extracts the images from test Data iterator without shuffling the sequence
# x contains image array and y has labels
dic = {0:'NORMAL', 1:'PNEUMONIA'}
j = 1
plt.figure(figsize=(20,20))
for i in range(0+228, 9+228):

  plt.subplot(3, 3, j)
  j = j + 1
  if preds[i, 0] >= 0.5:
      out = ('{:.2%} probability of being Pneumonia case'.format(preds[i][0]))


  else:
      out = ('{:.2%} probability of being Normal case'.format(1-preds[i][0]))

  plt.title(out+"\n Actual case : " + dic.get(y[i]))
  plt.imshow(np.squeeze(x[i]))
  plt.axis('on')
plt.show()
```

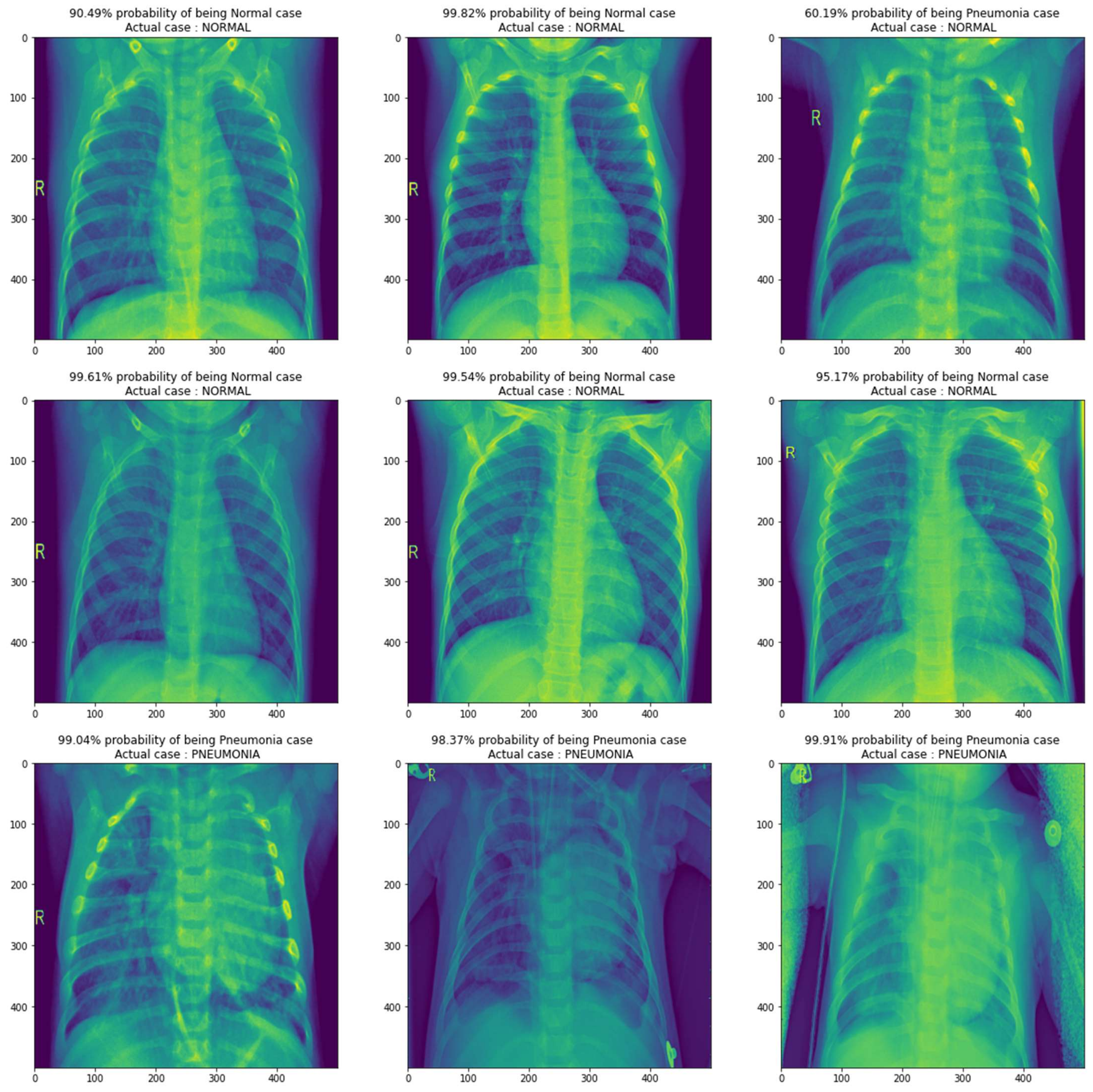

90.49% probability of being Normal case
Actual case : NORMAL
99.82% probability of being Normal case
Actual case : NORMAL
60.19% probability of being Pneumonia case
Actual case : NORMAL
99.61% probability of being Normal case
Actual case : NORMAL
99.54% probability of being Normal case
Actual case : NORMAL
95.17% probability of being Normal case
Actual case : NORMAL
99.04% probability of being Pneumonia case
Actual case : PNEUMONIA
98.37% probability of being Pneumonia case
Actual case : PNEUMONIA
99.91% probability of being Pneumonia case
Actual case : PNEUMONIA

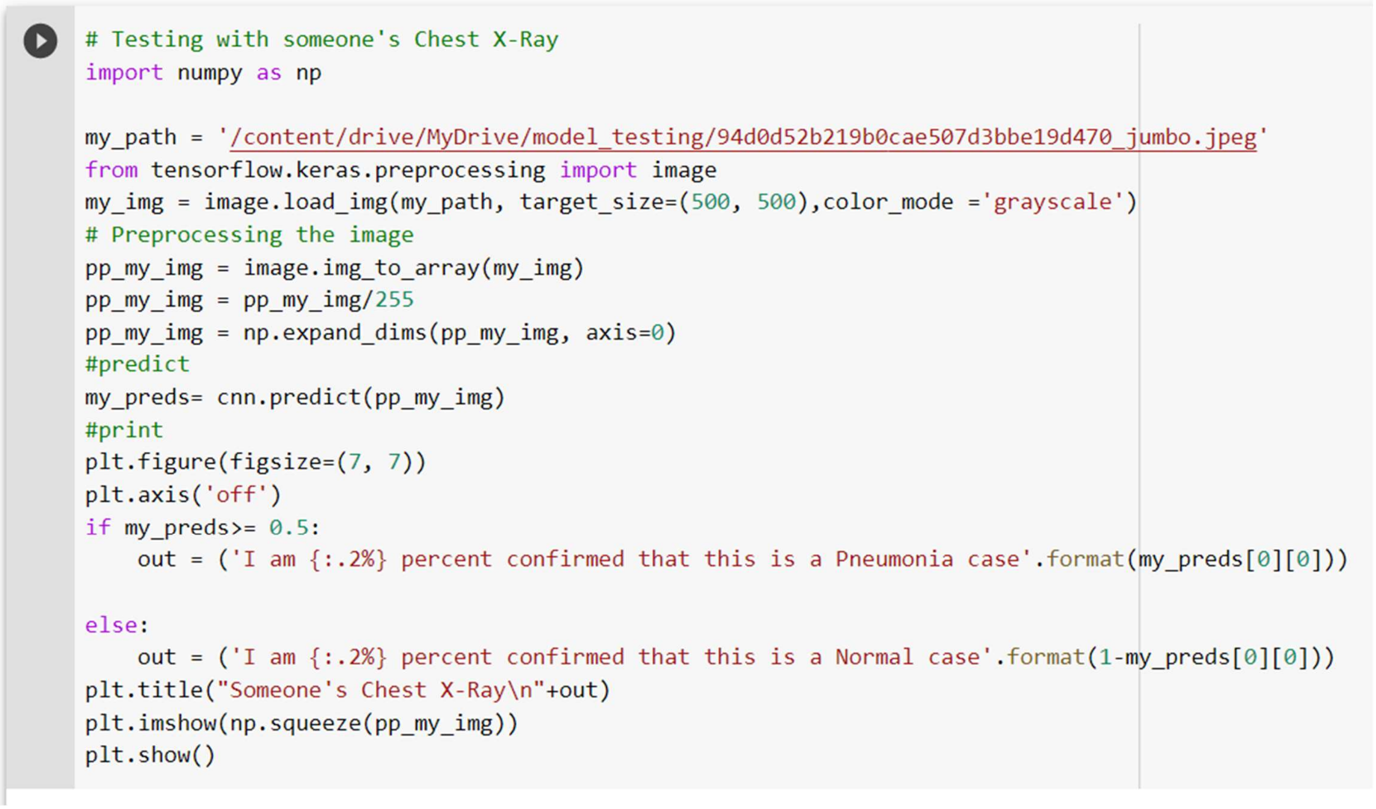
**Figure 4.12: Predictions on Test Data**

This code block gives a percentage prediction of the individual image that can be loaded directly from your drive by specifying its path.

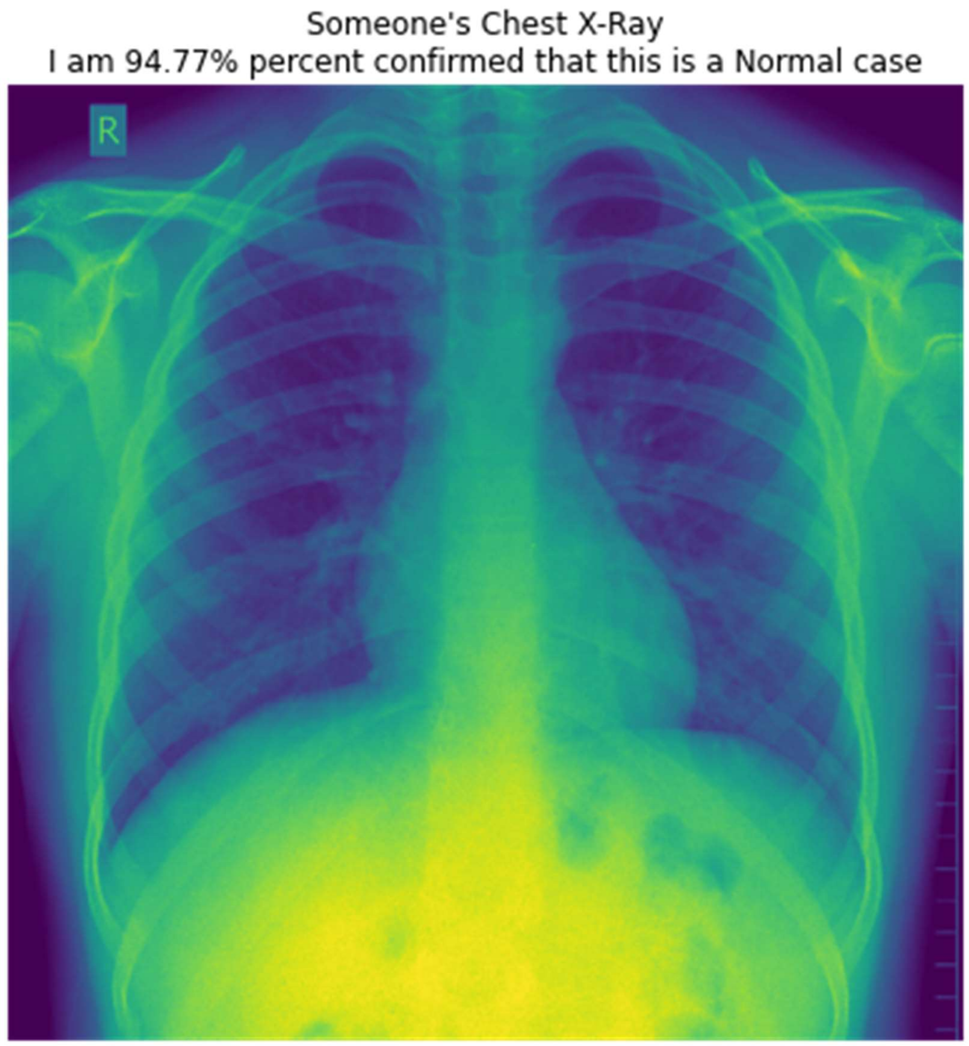
We have to re-create all the data pre-processing steps over here after importing the image as we had done previously to feed the test set into the model to get prediction. For pre-processing we need to import `tensorflow.keras.preprocessing.image` class.

1. Import image and define dimensions as (500, 500) and color channel as grayscale.
2. Convert image to array, rescale it by dividing it 255 and expand dimension by axis = 0 as our model takes 4 dimensions as seen earlier.
3. Finally let's predict the case!

**Let's do some field testing on my model with someone's chest x-ray which was not in the dataset used for training and testing:**



```
# Testing with someone's Chest X-Ray
import numpy as np

my_path = '/content/drive/MyDrive/model_testing/94d0d52b219b0cae507d3bbe19d470_jumbo.jpeg'
from tensorflow.keras.preprocessing import image
my_img = image.load_img(my_path, target_size=(500, 500),color_mode ='grayscale')
# Preprocessing the image
pp_my_img = image.img_to_array(my_img)
pp_my_img = pp_my_img/255
pp_my_img = np.expand_dims(pp_my_img, axis=0)
#predict
my_preds= cnn.predict(pp_my_img)
#print
plt.figure(figsize=(7, 7))
plt.axis('off')
if my_preds>= 0.5:
    out = ('I am {:.2%} percent confirmed that this is a Pneumonia case'.format(my_preds[0][0]))

else:
    out = ('I am {:.2%} percent confirmed that this is a Normal case'.format(1-my_preds[0][0]))
plt.title("Someone's Chest X-Ray\n"+out)
plt.imshow(np.squeeze(pp_my_img))
plt.show()
```



Someone's Chest X-Ray
I am 94.77% percent confirmed that this is a Normal case

**Figure 4.13: Someone’s Chest X-Ray with prediction**

Great job! We’ve just created a CNN model that can classify X-Ray images as a Pneumonia case or a normal case with 91% accuracy.

# Chapter 5

# Conclusions & Future Directions

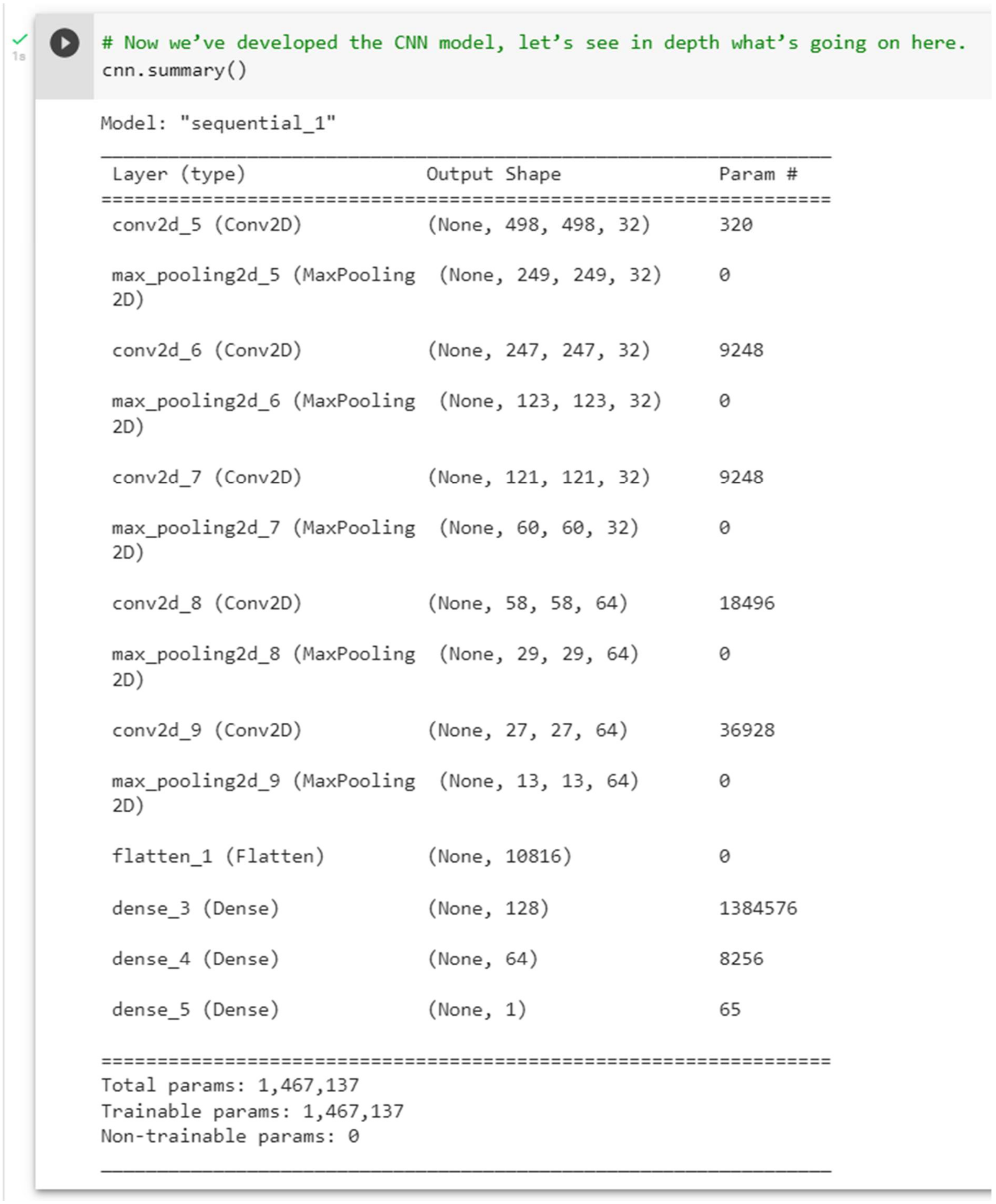
This chapter concludes the thesis providing the highlights of the research work and listing the avenues for possible extensions of the work in future.

## Highlights of the Current Research:

This thesis has brought together a number of aspects of the development of intelligent Machine Learning systems. How to make an ML model more efficient and trustworthy i.e., much more robust system.

For this project I started building the neural network architecture with three convolutional layers. There are three convolutional layers, after each Conv2D layer there is one MaxPooling2D layer. After that there are three dense layers, with 128, 64 and 1 unit(s) respectively. The batch size is 32. After training this model I did not get satisfactory result.  Then I build another CNN model with four convolutional layers. There are four convolutional layers, after each Conv2D layer there is one MaxPooling2D layer. After that there are three dense layers, with 128, 64 and 1 unit(s) respectively. This time I reduce the batch size to 24. After training this model I got better result compared to the first model.

After that I increase the number of convolutional layers to five. This time I build neural network model with five convolutional layers. There are five convolutional layers, after each Conv2D layer there is one MaxPooling2D layer. After that there are three dense layers, with 128, 64 and 1 unit(s) respectively. This time I reduce the batch size to 16. After training this model I got satisfactory result. For this model I got an accuracy measure of approximately 92% on the test data set. I present the final CNN architecture below.



```
# Now we've developed the CNN model, let's see in depth what's going on here.
cnn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 498, 498, 32)      320

 max_pooling2d_5 (MaxPooling  (None, 249, 249, 32)     0
 2D)

 conv2d_6 (Conv2D)           (None, 247, 247, 32)      9248

 max_pooling2d_6 (MaxPooling  (None, 123, 123, 32)     0
 2D)

 conv2d_7 (Conv2D)           (None, 121, 121, 32)      9248

 max_pooling2d_7 (MaxPooling  (None, 60, 60, 32)       0
 2D)

 conv2d_8 (Conv2D)           (None, 58, 58, 64)        18496

 max_pooling2d_8 (MaxPooling  (None, 29, 29, 64)       0
 2D)

 conv2d_9 (Conv2D)           (None, 27, 27, 64)        36928

 max_pooling2d_9 (MaxPooling  (None, 13, 13, 64)       0
 2D)

 flatten_1 (Flatten)         (None, 10816)             0

 dense_3 (Dense)             (None, 128)               1384576

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 1,467,137
Trainable params: 1,467,137
Non-trainable params: 0
_________________________________________________________________
```

## Future Direction:

All the works carried out have significant scopes of extension in future. If I had had more time and better hardware resources, I would have done more research on how decreasing the batch size, increasing the number of epochs and keeping the learning rate at 0.0001 would affect the accuracies and how overfitting the model is. Another difficulty I face is the constraint of resources. I used Google Colab GPU (Graphics Processing Unit) which basically get timed out while training is going on. After that Google Colab will not allow you to use GPU for next three to four days. If we build similar models with different batch size, different number of epochs, different learning rate. Then we may get better result. If we increase number of images in the dataset then obviously we will get better result.

# References:

[1] Make Your Own Neural Network: A Gentle Journey Through the Mathematics of Neural Networks and Making Your Own Using the Python Computer Language by Tariq Rashid

[2] Artificial Intelligence For Dummies (For Dummies (Computer/Tech)) by John Paul Mueller and Luca Massaron

[3] Machine Learning For Absolute Beginners: A Plain English Introduction: 1 (Machine Learning from Scratch) by Oliver Theobald

[4] Superintelligence: Paths, Dangers, Strategies by Nick Bostrom

[5] Artificial Intelligence | Third Edition | By Pearson: A Modern Approach by Stuart J. Russell and Peter Norvig

[6] Artificial Intelligence Engines: A Tutorial Introduction to the Mathematics of Deep Learning by James V Stone

[7] Life 3.0: Being Human in the Age of Artificial Intelligence by Max Tegmark

[8] Deep Learning Illustrated: A Visual, Interactive Guide to Artificial Intelligence (Addison-Wesley Data & Analytics Series) by Jon Krohn and Grant Beyleveld

[9] Predictive Analytics For Dummies, 2nd edition by Dr. Anasse Bari, Mohamed Chaouchi, Tommy Jung

[10] Data Science from Scratch 2e: First Principles with Python by Joel Grus

[11] Hands-On Machine Learning with Scikit-Learn, Keras and Tensor Flow: Concepts, Tools and Techniques to Build Intelligent Systems by Aurelian Geron

[12] Prediction Machines: The Simple Economics of Artificial Intelligence by Avi Goldfarb, Joshua Gans and Ajay Agarwal

[13] Human + Machine: Reimagining Work in the Age of AI by H. James Wilson and Paul R. Daugherty

[14] Artificial Intelligence for Humans, Volume 1: Fundamental Algorithms by Jeff Heaton

[15] Artificial Intelligence and Machine Learning for Business: How modern companies approach AI and ML in their business and how AI and ML are changing their business strategy by Scott Chesterton

[16] Python Machine Learning: Unlock deeper insights into Machine Leaning with this vital guide to cutting-edge predictive analytics by Sebastian Raschka

[17] Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2, 3rd Edition by Sebastian Raschka and Vahid Mirjalili

[18] Deep Learning (Adaptive Computation and Machine Learning series) by Ian Goodfellow and Aaron Courville

[19] TensorFlow in 1 Day: Make your own Neural Network by Krishna Rungta

[20] Introduction to Machine Learning with Python A GUIDE FOR DATA SCIENTISTS by Andreas C. Müller & Sarah Guido

[21] Darlow, L. et al. 2018. CINIC-10 Is Not ImageNet or CIFAR-10, University of Edinburgh. Cited 26.11.2019. Available:
https://doi.org/10.7488/ds/2448

[22] Radice, M. 2019. Basic Structure of the Human Nervous System. Date of retrieval 5.12.2019. Available: https://mikerbio.weebly.com/structure--function.html

[23] Khan Academy 2019. Neuron action potentials: The creation of a brain signal. Date of retrieval 7.12.2019. Available: https://www.khanacademy.org/test-prep/mcat/organ-systems/neuron-membrane-potentials/a/neuron-action-potentials-the-creation-of-a-brain-signal

[24] Richárd, N. 2018. The differences between Artificial and Biological Neural Networks. Date of retrieval 5.12.2019. Available: https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7

[25] Bre, F., Giminez, J. 2017. Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks. Cited 6.12.2019. Available:

https://www.researchgate.net/publication/321259051_Prediction_of_wind_pressure_coefficients_on_building_surfaces_using_Artificial_Neural_Networks

[26] Dormehl, L. 2019. What is an artificial neural network? Here's everything you need to know. Date of retrieval 6.12.2019. Available:

https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/ 28

[27] Moawad, A. 2018. Neural networks and back-propagation explained in a simple way. Date of retrieval 13.12.2019. Available:

https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e

[28] Brownlee, J. 2016. Crash Course On Multi-Layer Perceptron Neural Networks. Date of retrieval 8.12.2019. Available:

https://machinelearningmastery.com/neural-networks-crash-course/

[29] Ahirwar, K. 2017. Everything you need to know about Neural Networks. Date of retrieval 16.12.2019. Available:

https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491

[30] Hu, W. 2018. Towards a Real Quantum Neuron. Cited 8.12.2019. Available:

https://www.researchgate.net/publication/323775654_Towards_a_Real_Quantum_Neuron

[31] Kohl, N. 2010. Answer to Role of Bias in Neural Networks on Stackoverflow. Date of retrieval 16.12.2019. Available:

https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks

[32] Tiwari, S. 2019. Activation functions in Neural Networks. Date of retrieval 17.12.2019. Available:

https://www.geeksforgeeks.org/activation-functions-neural-networks/

[33] Liu, D. 2017. A Practical Guide to ReLU. Date of retrieval 17.12.2019. Available:

https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

[34] Sharma V, A. 2017. Understanding Activation Functions in Neural Networks. Date of retrieval 17.12.2019. Available: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[35] Ng, A. 2011. Backpropagation Algorithm. Date of retrieval 18.12.2019. Available: https://www.coursera.org/lecture/machine-learning/backpropagation-algorithm-1z9WW

[36] McDonald, C. 2017. Machine learning fundamentals (I): Cost functions and gradient descent. Date of retrieval 18.12.2019. Available:

https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220

[37] Murphy, K. 2012. Machine Learning: A Probabilistic Perspective. Cited 18.12.2019. Available:

https://books.google.fi/books?id=NZP6AQAAQBAJ&pg=PA247&redir_esc=y#v=onepage&q&f=false

[38] Kingma, D., Ba, J. 2014. Adam: A Method for Stochastic Optimization. Cited 18.12.2019. Available:

https://arxiv.org/abs/1412.6980

[39] Christensson, P. 2019. RGB Definition. Date of retrieval 10.12.2019. Available:

https://techterms.com/definition/rgb

[40] Courtney, J. 2001. Application of Digital Image Processing to Marker-free Analysis of Human Gait. Cited 10.12.2019. Available:

https://www.researchgate.net/publication/267210444_Application_of_Digital_Image_Processing_to_Marker-free_Analysis_of_Human_Gait

[41] Dertat, A. 2017. Applied Deep Learning - Part 4: Convolutional Neural Networks. Date of retrieval 11.12.2019. Available: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

[42] Cornelisse, D. 2018. An intuitive guide to Convolutional Neural Networks. Date of retrieval 11.12.2019. Available:

https://medium.com/free-code-camp/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

[43] Prabhu 2018. Understanding of Convolutional Neural Network (CNN) — Deep Learning. Date of retrieval 12.12.2019. Available:

https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

[44] Corvil.com 2019. What is the difference between 'SAME' and 'VALID' padding in tf.nn.max_pool of tensorflow? Date of retrieval 12.12.2019. Available:

https://www.corvil.com/kb/what-is-the-difference-between-same-and-valid-padding-in-tf-nn-max-pool-of-tensorflow

[45] Saha, S. 2018. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. Date of retrieval 16.12.2019. Available:

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[46] Dertat, A. 2017. Applied Deep Learning - Part 1: Artificial Neural Networks. Date of retrieval 16.12.2019. Available:

https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6?#860e

[47] Deshpande, A. 2016. A Beginner's Guide To Understanding Convolutional Neural Networks. Date of retrieval 16.12.2019. Available:

https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/ 31

[48] Shah, T. 2017. About Train, Validation and Test Sets in Machine Learning. Date of retrieval 6.1.2020. Available:

https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

[49] Cai, E. 2014. Machine Learning Lesson of the Day – Overfitting and Underfitting. Date of retrieval 6.1.2020. Available:

https://chemicalstatistician.wordpress.com/2014/03/19/machine-learning-lesson-of-the-day-overfitting-and-underfitting/
[50] Sharma, S. 2017. Epoch vs Batch Size vs Iterations. Date of retrieval 6.1.2020. Available:

https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

[51] Srivastava, N. et al. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Cited 6.1.2020. Available:

http://jmlr.org/papers/v15/srivastava14a.html

[52] D., F. 2017. Batch normalization in Neural Networks. Cited 6.1.2020. Available:

https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c

[53] https://stackoverflow.com/questions/61060736/how-to-interpret-model-summary-output-in-cnn

[54] https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7

[55] https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148#:~:text=Strides,with%20a%20stride%20of%202.

[56] https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[57] https://stackoverflow.com/questions/37674306/what-is-the-difference-between-same-and-valid-padding-in-tf-nn-max-pool-of-t

[58] https://deeplizard.com/learn/playlist/PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU

[59] https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

[60] https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253

[61] Convolutional Neural Network (CNN). Date of retrieval 6.1.2020. Available:

https://www.tensorflow.org/tutorials/images/cnn

[62] Keras team, 2019. Train a simple deep CNN on the CIFAR10 small images dataset. Date of retrieval 6.1.2020. Available: https://github.com/keras-team/keras/blob/master/examples/cifar10_cnn.py

[63] https://keras.io/api/callbacks/reduce_lr_on_plateau/