



People's Democratic Republic of Algeria  
Ministry of Higher Education and Scientific Research  
**Ibn Khaldoun university – Tiaret**  
Mathematics and Computer Science faculty  
**Computer Science department**

*Thesis*

---

# **Data Augmentation on the Convolutional Neural Network for Image Classification**

---

*For obtaining Master's degree*

**Specialty: Software Engineering**

**By: MAKBOUL Ilias Sid Ahmed**

The jury composed of:

Mr. BEKKI Khadhir	MAA University-Ibn-Khaldoun Tiaret	President
Mr. CHENINE Abdelkader	MAA University-Ibn-Khaldoun Tiaret	Supervisor
Mr. ABID Khaled	MAA University-Ibn-Khaldoun Tiaret	Examiner

*Academic year 2019-2020*

## Dedication

Praise be to Allah, Lord of the Worlds **الْحَمْدُ لِلَّهِ رَبِّ الْعَالَمِينَ**

I dedicate my dissertation work to my family, A special feeling of gratitude to my loving parents, Khaled and Halima MAKBOUL whose words of encouragement and push for tenacity ring in my ears. My brothers Amine, Youcef and Ismail especially Amine who helped me a lot with his advice and motivation.

I would like to express my gratitude to all my teachers from whom we learn a lot during these five years on the computer science department at Ibn Khaldoun University, a special gratitude to Mr. TALBI Omar who leave a positive impact on me who thanx to him and his lecture i decide to choose computer science.

I also dedicate this dissertation to my many friends who have supported me through the process. Especially those who encourage me to prepare this research: A. Abdeldjalil, A. Hakim, M. M'hamed, Z. Sofiane, D. Mohammed and all my friends.

## **ACKNOWLEDGEMENT**

I would like to express my gratitude and a special thanx to my supervisor MR. CHENINE Abdelkader who offered me a wonderful opportunity to work under his guidance on domain that always fascinated me I got a lot of experiences on artificial intelligence, deep learning field particularly, so I thank him for his advices and availability all the time through the process of realization of this academic research.

I also want to express my gratitude Mr. ABID Khaled and Mr. BEKKI Khadhir for considering to evaluate my final year thesis.

# Abstract

In the field of computer vision, image classification is considered as one of the most dominated research domains. Machine Learning, specifically, deep learning becomes the most useful tool that handles image classification tasks. The Deep Learning model needs to be trained on a huge number of samples, which leads us to one of the most popular issues facing this field "The lack of data". Data Augmentation increase the number of training images.

In this thesis, we clarify the impact of the amount of training data and the effect of data augmentation on the performance of CNN models in image classification.

We evaluate this on Kaggle dataset by manipulating it in a different manner we created three more datasets from the original one. the first one contains 8% of the original dataset, the second one is generated by applying seven image manipulation technique(rotation, shifting, horizontal flipping) on the second dataset with random parameters, the last one was created using supervised data augmentation with specific parameters instead of the random one, then we train each one of the 4 datasets on two different deep learning architecture ResNet and AlexNet. The results obtained show that the more data we feed to the model the better performance we get, using data augmentation increase the level of accuracy besides of using supervised data augmentation show a little better performance than the random augmentation but even a small percentage may make the difference

**keywords:** Computer Vision, Image Classification, Machine Learning, Deep Learning, Data Augmentation, Convolutional Neural Network CNN, AlexNet, ResNet

# Content

<b>List of Figures</b>	VI
<b>List of Table</b>	VIII
<b>INTRODUCTION</b>	<b>1</b>
Background	2
The scope of this Research	2
Research Goal and Question	2
Structure of Manuscript	2
<b>CHAPTER 1 DEEP LEARNING AND IT'S APPLICATION</b>	<b>3</b>
Introduction	3
1.1 Historical context	3
1.1.1 Artificial intelligence	3
1.1.2 Machine learning	4
1.1.2.1 Supervised learning	5
1.1.2.2 Unsupervised learning	5
1.1.2.3 Semi-supervised learning	5
1.1.2.4 Reinforcement learning	6
1.2 Deep learning	6
1.2.1 The Neuron	8
1.2.2 Artificial Neuron	8
1.2.3 Artificial neural networks	10
1.2.3.1 Input layer	11
1.2.3.2 Hidden layers	11
1.2.3.3 Output layer	11
1.2.3.4 Activate functions	11
1.2.3.4.1 Linear activation functions	12
1.2.3.4.2 Non-linear activation function	13
1.2.3.4.2.1 Sigmoid	13
1.2.3.4.2.2 Tanh	14
1.2.3.4.2.3 ReLU	15
1.2.3.4.2.4 SoftMax	16
1.2.3.5 Loss functions	16
1.2.4 Gradient descent	18
1.2.5 The Back-Propagation Algorithm	20
1.2.6 Convolutional Neural Networks (CNNs / ConvNets)	23
1.2.6.1 Convolution operation	23
1.2.6.2 Architecture of CNN	24
1.2.6.2.1 Convolution layer	26
1.2.6.2.1.1 Filter/ Kernel	27
1.2.6.2.1.2 Hyperparameters	27
1.2.6.2.2 Pooling layer	29
1.2.6.2.3 Fully Connected Layer	30
1.3 Regularization for Deep Learning	31

1.3.1 Underfitting and overfitting	31
1.3.2 Dropout	32
1.4 Deep Learning Architecture	33
1.4.1 ResNet	33
1.4.2 AlexNet	34
<b>CHAPTER 2 DATA AUGMENTATION</b>	<b>35</b>
Introduction	35
2.1 Data Augmentation	36
2.1.1 Affine Transformation (Basic Image Manipulation)	37
2.1.1.1 Horizontal Flipping	37
2.1.1.2 Horizontal and Vertical Shift	37
2.1.1.3 Rotation	38
2.1.1.4 Random Cropping	38
2.1.1.5 Random Erasing	39
2.1.2 Color Space Why and how color space affect models	40
2.1.2.1 RGB Color Space	40
2.1.2.2 HSV Color Space	41
2.1.2.2.1 Convert RGB to HSV	42
2.1.2.3 HSL Color Space	44
2.1.2.3.1 Convert RGB to HSL	44
2.1.2.4 TV Color Space YUV	45
2.1.2.4.1 Convert RGB to YUV	46
2.1.2.5 XYZ and LUV Color Space	47
2.1.2 Deep Learning Approach	49
2.1.2.1 GAN-Based Data Augmentation	49
2.1.2.2 Neural Style Transfer	50
2.1.3 Smart Augmentation	51
<b>CHAPTER 3 EXPERIMENTATION AND RESULTS</b>	<b>52</b>
3.1 Development Tools	52
3.1.1 Anaconda	52
3.1.2 Jupyter Notebook	53
3.1.3 Python	53
3.2 Libraries	54
3.2.1 Keras	54
3.2.2 Tensorflow	54
3.2.3 Scikit-Image	54
3.2.4 Pillow	54
3.2.5 Matplotlib	54
3.3 Kaggle Dataset	54
3.4 Implementation	56
3.5 Experimentation	56
3.6 Discussion Results	57
<b>Conclusion</b>	<b>64</b>
<b>Bibliography</b>	<b>67</b>

## Liste of Figures

**Figure 1** Deep Learning is a Subfield of Machine Learning which is a Subfield of AI

**Figure 2** Traditional Programming vs. Machine Learning

**Figure 3** Difference Between a Simple Neural Network and a Deep Learning Neural Network

**Figure 4** Machine Learning Approaches with Algorithm Example

**Figure 5** Comparisons between Machine Learning & Deep Learning

**Figure 6** Neuron in Biology

**Figure 7** Diagram of the Functionality of a Neuron in a Deep Learning Neural Network

**Figure 8** Illustration of an Artificial Neuron

**Figure 9** Artificial Neural Network Architecture

**Figure 10** Activation Function

**Figure 11** Linear Activation Function

**Figure 12** Sigmoid Function

**Figure 13** Tanh Hyperbolic

**Figure 14** ReLU Graph

**Figure 15** ReLU vs Leaky ReLU

**Figure 16** SoftMax

**Figure 17** Neural Network Loss Visualization

**Figure 18** Neural Network Workflow

**Figure 19** The Quadratic Error Surface for a Linear Neuron

**Figure 20** Visualizing the Error Surface as a set of Contours

**Figure 21** Convergence is difficult when our learning rate is too large

**Figure 22** Reference Diagram for the Derivation of the Backpropagation Algorithm

**Figure 23** Artificial Neural Network and Convolutional Neural Network

**Figure 24** Convolutional Neural Network Architecture

**Figure 25** Convolutional Operation

**Figure 26** Example of Convolutional Operation

**Figure 27** Filter with Stride ( $s$ ) = 2

**Figure 28** Zero Padding Example with ( $p$ )=1

**Figure 29** Convolution Operation on Volume

**Figure 30** Max Pooling and Avg Pooling

**Figure 31** Connection Between Convolutional Layer and Fully Connected Layer

**Figure 32** Model Capacity and its Effect on Underfitting and Overfitting

**Figure 33** Detection of Overfitting

**Figure 34** Neural Network with Three Unit Dropped Randomly

**Figure 35** ResNet Residual Learning Block

**Figure 36** ResNet-34 Layers Architecture Diagram

**Figure 37** Image Data Augmentation for Deep Learning

**Figure 38** Image Horizontal Flipping (Mirror)

**Figure 39** Horizontal and Vertical Shift

**Figure 40** Rotation Example

**Figure 41** Random Cropping

**Figure 42** Random Erasing Example

**Figure 43** Representation of the RGB Color Space as a Three-Dimensional Unit Cube

**Figure 44** A Color Image and its Corresponding RGB Channels

**Figure 45** HSV Color Space in Cylindrical Coordinates

**Figure 46** HSV Color Components

**Figure 47** HSL Color Space

**Figure 48** HSL Color Component

**Figure 49** YUV Color Space

**Figure 50** YUV Color Component

**Figure 51** Fake Faces Generated by StyleGAN

**Figure 52** GAN Generator and Discriminator

**Figure 53** Example of Neural Style Transfer

**Figure 54** Anaconda Navigator

**Figure 55** Jupyter Notebook Interface

**Figure 56** Sample of Cats and Dogs from Kaggle Dataset

**Figure 57** Data Augmentation Applied for Each Image

**Figure 58** The Process of Dataset Augmentation

**Figure 59** ResNet-20 Training Loss

**Figure 60** ResNet-20 Validation Accuracy

**Figure 61** ResNet-20 Validation Loss

**Figure 62** AlexNet Training Loss

**Figure 63** AlexNet Validation Accuracy

**Figure 64** AlexNet Validation Loss



## Liste of Tables

<b>Table</b>	<b>Title</b>	<b>Page</b>
Table 1	RGB/HSV Values	43
Table 2	RGB/HSL Values	45
Table 3	RGB/YUV Values	47
Table 4	RGB/XYZ/LUV Values	48
Table 5	Comparison of Results for Different Dataset	57
Table 6	Per Class accuracy of ResNet-20 and AlexNet	58
Table 7	ResNet-20 Result	59
Table 8	AlexNet Result	60

# INTRODUCTION

Deep learning is the new area of machine learning that moving machine learning closer to one of its goals which is Artificial Intelligence, Deep learning transformed application that required vision expertise into an engineering problem that can be solved using this technique, application that used a rule-based algorithm may be solved using training systems that can learn and improve on its own by examining computer algorithm offered by deep learning, we are talking about application and problems that no one expect that they may be handled without human involvement. Machine learning and deep learning expanded the computers limit more that we expect.

Deep learning CNNs convolutional neural network model have shown great performance especially on image classification and speech recognition which exceed human vision sometimes, deep learning model required a large amount of data to train on, so we my face problem related to the lack of data or overfitting problem, data augmentation is one of the most used technique that may handle this problem, can we get a better performance with small dataset? This what shall show in this thesis by training two different famous deep learning architecture using manipulated dataset from Kaggle. First, we train our model on the complete dataset then do the same thing using only 8% of the same dataset this small percentage is extracting randomly, third we augment this 8% of dataset using two method random augmentation and supervised augmentation and train them on the same previous deep learning architecture besides comparing between the results obtained.

## 1. BACKGROUND

This is not the first experimentation made to understand the performance of CNN on different amount of data beside how data augmentation may lead to a better performance Furthermore higher accuracy. We know that there are many controversies published paper in [1] the authors proved that applying a specific small transformation on samples may lead to a better performance and robustness of the classifier. In more recent paper [2] the author compared between different method of data augmentation using three architectures ResNet, VGG16, and InceptionV3 it's shows that the GAN data augmentation shows a little better result than the basic image manipulation, combining different color space and position augmentation gives the maximum accuracy for this research.

## **2. The Scoop of this Research**

This thesis focuses on study of the impact of the amount of data used on training set in image classification by deep learning model, especially architecture Deep Residual Learning ResNet which won the first place on the ILSVRC 2015 and AlexNet using Keras which is a high-level neural network API written in Python and capable of running on top of TensorFlow using either CPU or GPU.

## **3. Research Goal and Question**

Our objective is to answer the question: how the quantity of training dataset may affect the performance of deep learning models and how can we deal with the lack of training data problems.

## **4. Structure of Manuscript**

### **Chapter 1**

Deep Learning is the most powerful tools of machine learning on image classification tasks, in this chapter "DEEP LEARNING AND ITS APPLICATION" we discuss the relation between machine learning and deep learning and how deep learning works from the first idea that is inspired from the biological neuron than passing through the Convolutional Neural Network CNN to describe the secret behind their fame and how CNN works and arrived to the gradient descent and backpropagation algorithm and related problems like overfitting, finally, we introduce two of the most famous architecture ResNet and AlexNet briefly.

### **Chapter 2**

Describe image data augmentation technique used starting with simple image manipulation than color space system and finally, the latest technique GAN-based data augmentation and neural style transfer and smart augmentation.

### **Chapter 3**

Describe the tool, libraries and dataset used in this experimentation, the augmentation process and the dataset manipulation finally, briefly discussion and analysis of the results obtained.

# CHAPTER 1

## DEEP LEARNING AND ITS APPLICATION

### Introduction

in the last years, the artificial intelligence known as AI becomes a subject of intent media hype. machine learning and deep learning which are a subfield of artificial intelligence come up in countless articles, often outside of technology-minded publications. self-driving cars, chatbots virtual assistants, and many other fields based on artificial intelligence (AI) where the human job will be scarce and economic activity will be handled with robots also known as AI agents [3]. our interest in this chapter is Deep Learning which is a subfield of Machine learning which is a part of AI (Figure 1) and what is the relation between Artificial intelligent and Machine learning and deep learning and how and which problems can be handled using these 3 techniques.

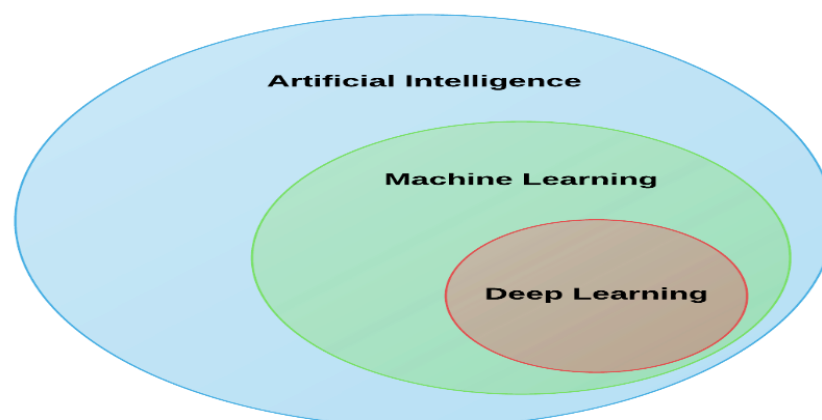
### 1.1 Historical Context

#### 1.1.1 Artificial Intelligence [3]

*“Artificial Intelligence or AI is the art of making computers think and behave in the same manner as human brain in order to solve more complex problems without the need of programmer’s guide”*

Artificial intelligence was born in the 1950s, it all begins when a handful of pioneers from the nascent field of computer science, they start asking questions about whether computers could be made to behave intelligently and to think like a human, a question whose ramifications we’re still exploring today. Chess programmer for example only involved hardcoded rules crafted and it didn't qualify as Machine Learning. many of them believed that the human-level could be achieved by having programmers handcraft a sufficiently large set of explicit rules for manipulating knowledge. this approach was known as symbolic AI in the 1950s to the late of 1980s.

Although symbolic AI shows good improvement in solving well-defined logical problems but it’s so hard to figure out explicit rules for solving complex problems such as image classification or language translation and speech recognition. that gives a lot of space to a new approach which called Machine Learning ML

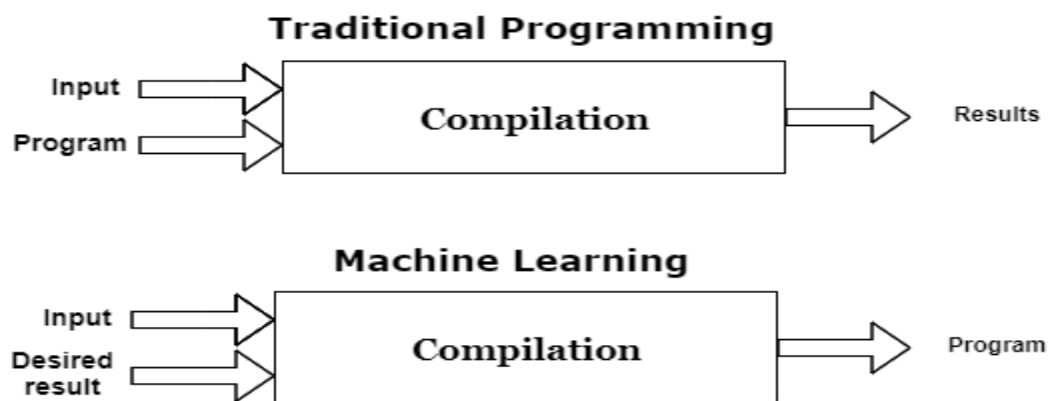


**Figure 1** Deep Learning is a Subfield of Machine Learning which is a Subfield of AI

### 1.1.2 Machine Learning [3]

*“Machine Learning ML is the art of making computers learn from experiences and previous situations this called the natural human learning process. we feed to the computer a dataset and the predicted result and let the computer learn and analyze the relationship between them in order to learn how could that particular data lead to this result”*

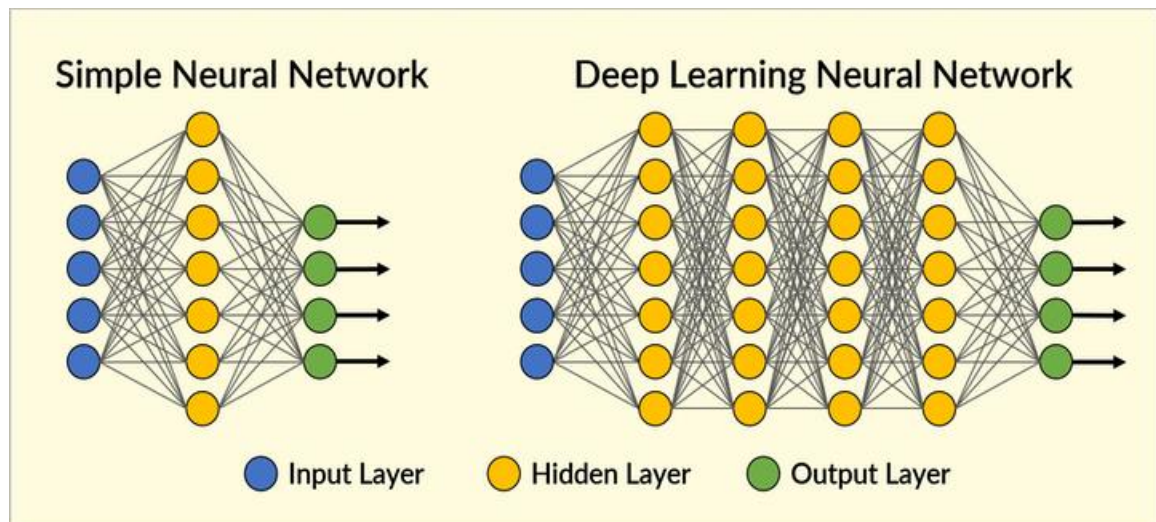
Machine Learning has the attention of searchers in AI and even computer sciences since 1983 [4], the question was always how a computer can learn from experiences (data) rather than programmers crafting data processing rules by hand, and how to learn on its own how to perform a specific task these questions lead searchers to a new programming paradigm called Machine Learning ML. in the classical programming or symbolic AI the programmer input rules or program (algorithm) and data to be processed depending to these rule, and outcome answers, Machine learning everything, the programmer input the data with the answers expected according to this data and outcome the rule than matching between the data and the answers (Figure 2). the rule can be applied to new data in order to produce original answers



**Figure 2** Traditional Programming vs. Machine Learning

*“It would be useful if computers could learn from experience and thus automatically improve the efficiency of their own programs during execution. A simple but effective rote-learning facility can be provided within the framework of a suitable programming” [5]*

Machine learning have the ability to learn and improve with experiences. Machine learning process begin with the raw data which is used for extracting useful information that helps in learning and in decision-making using shallow or deep architecture (Figure 3) to grant that [6]



**Figure 3** Difference Between a Simple neural Network and a Deep Learning Neural Network

Machine Learning system is trained rather than be explicitly programmed, AI focus on teaching computers how to learn without being programmed for specific tasks Machine Learning can be carried out using following approaches:

#### 1.1.2.1 Supervised Learning [6]

in ML and AI, supervised learning is a group of algorithms that determines a predictive model using data with known outcomes. The model is leaned by training on those data while the outputs are clear and try to make relations between the data and its output in order to predict on new data (model generalization) through an appropriate learning algorithm such as Neural Network, Random Forests and Linear Regression that works through some optimization routine to minimize a loss function.

#### 1.1.2.2 Unsupervised Learning [6]

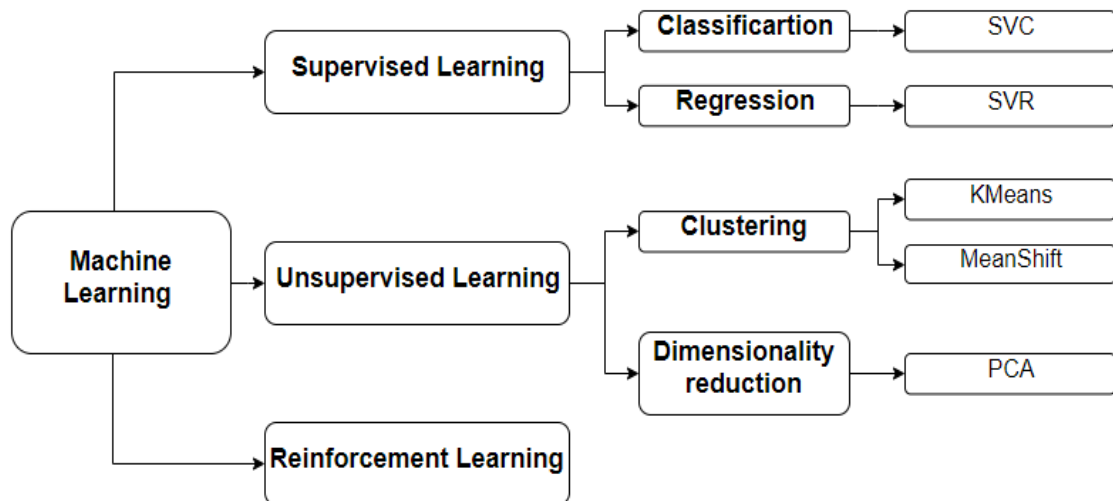
In Machine Learning and Artificial Intelligent Unsupervised learning involves data that comprises input without any target output. The objectives of unsupervised learning are different for example the clustering used to discover groups of similar data items in order to extract the similarities between the data items. The visualization used to reduce our data size by projecting high-dimensional space to two or three dimensions in order to view the similar data items

#### 1.1.2.3 Semi-Supervised Learning [6]

Semi-supervised machine learning is a combination of supervised and unsupervised machine learning methods. We already set that supervised learning is the learning that occurs during the training of an Artificial Neural Network when the data in our training set is labeled, unsupervised learning on the other hand is used when our training data is not labeled. Semi-Supervised learning used when we have a combination of both labeled and unlabeled data. Let's say we have a large amount of no labeled data well we could go forward and manually label some portion of this dataset ourselves and use that portion to train our model this call pseudo-labeling [7].

### 1.1.2.4 Reinforcement Learning

Is a type of dynamic programming that trains algorithms using a system of reward and punishment, a reinforcement learning algorithm, or agent, learns by interacting with its environment. The agent receives rewards by performing correctly and penalties for performing incorrectly. The agent learns without intervention from a human by maximizing its reward and minimizing its penalty. [8] learning. Reinforcement learning has been successful in applications as diverse as autonomous helicopter flight, robot legged locomotion, cell-phone network routing, marketing strategy selection, factory control and efficient webpage indexing.[6]



**Figure 4** Machine Learning Approaches with Algorithm Example

## 1.2 Deep Learning

*“The modern term “deep learning” does beyond the neuroscientific perspective on the current breed of machine learning models. It appeals to a more general principle of machine learning multiple levels of composition, which can be applied in machine learning frameworks that are not necessarily neurally inspired”[9]*

Deep Learning is a subfield of Machine Learning which is a subfield of AI, which gained popularity in recent past [6], it’s referred to the architecture which contains multiple hidden layers (Deep Network), this architecture allows it to learn features with multiple levels of abstraction, from a higher level to low level. The number of layers contributes to a model is called the depth.

Deep learning is a type of machine learning that involves Artificial Neural Networks (ANN), whose designs are inspired by the way that scientists believe the brain works.

Deep learning architectures such as:

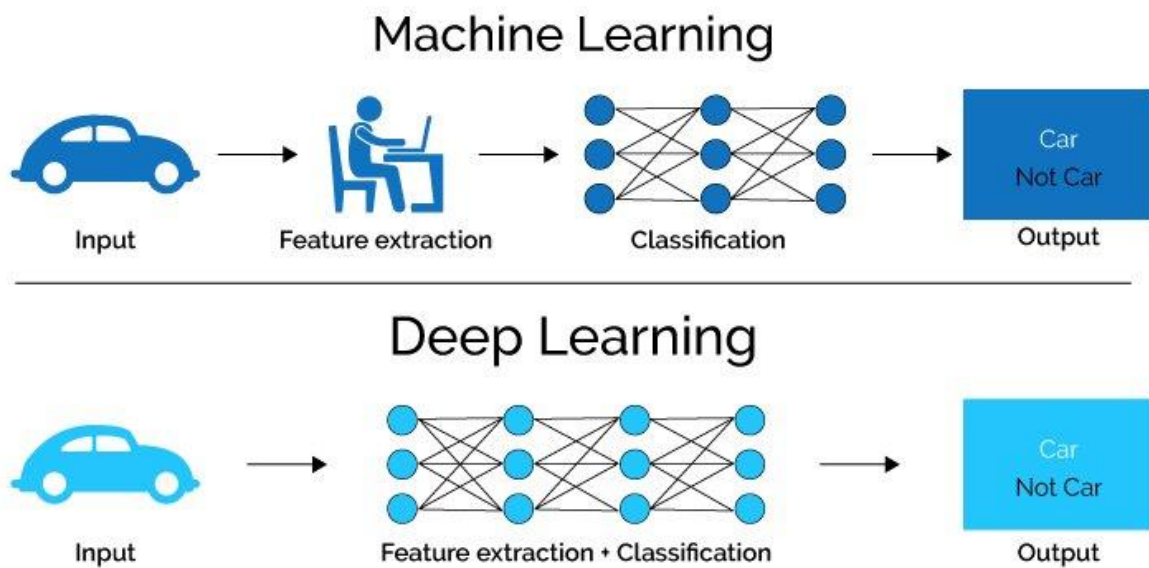
- Deep neural networks
- Recurrent neural networks
- Convolutional neural networks
- Recursive neural networks



have been applied to fields including computer vision, machine vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced great results and in some cases surpassing human expert performance. [10]

Feature engineering is a key step in the model building process. It is a two-step process:

- Feature extraction
- Feature selection



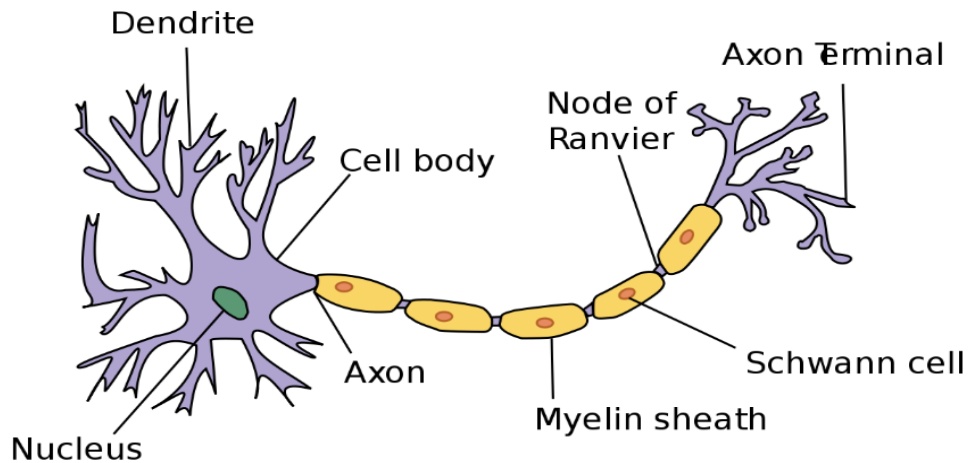
**Figure 5** Comparisons between Machine Learning & Deep Learning

feature extraction for image classification is to get the possible features from the dataset for example corner, edges... and from those features we get we need to extract more features it means more details about our features in order to get small or hidden features. Features Selection consist of selecting features that affect the model performance more this means we need to get some features that we extract from our dataset and feed it to our model to see the impact caused and then select the best features to our model, Convolutional Neural Networks (CNNs) do all this by default [11].



### 1.2.1 The Neuron:

Neurons in deep learning were inspired by neurons in the human brain [3] Figure 6 shows the anatomy of a brain neuron:



**Figure 6** Neuron in Biology

As we can see, neurons have a very interesting structure. Neuron works in groups together inside the human brain in order to perform functionality that human need and requite in life. A question was asked by Geoffrey Hinton during his seminal research in neural networks if we could build an algorithm for the computer to simulate neurons in the human brain [12]. The hop that if they can mimic brain structure, they might capture some of its capability.

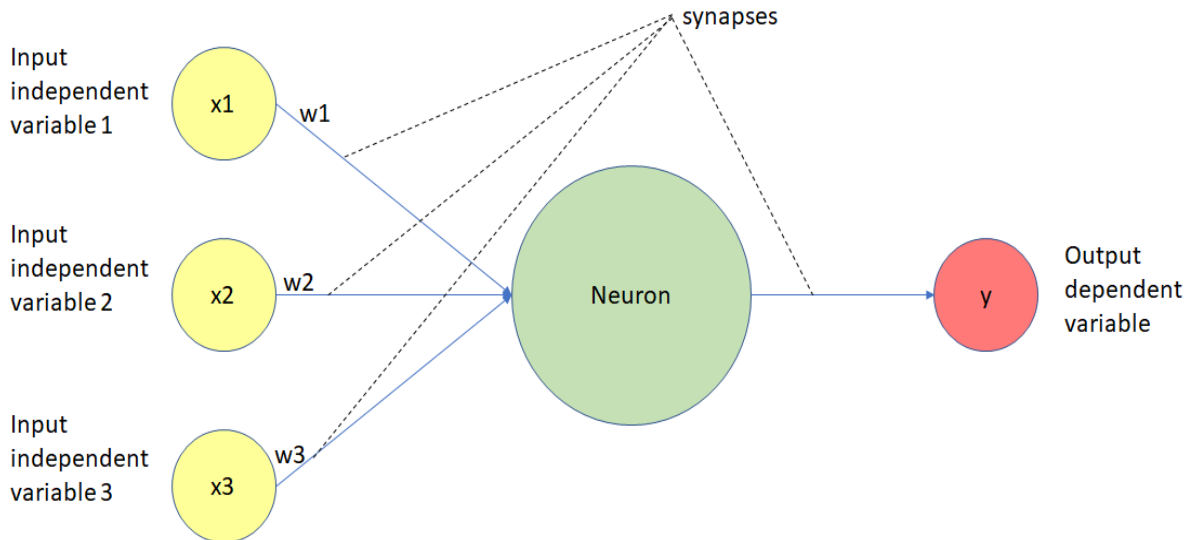
To do this, researchers and scientists studied the behavior of neurons in the human brain. An important observation shows up that the neuron by itself is useless. Instead, it requires networks of neurons (Neuron Network) to generate meaningful tasks. the secret behind that is because the neuron function by sending and receiving signals to other connected neurons. The neuron dendrites have the ability to receive signals from the previous neuron then pass those signals through the axon. the dendrites of the neuron are connected to another neuron axon. we call this connection as a synapse. synapse concept has been generalized on deep learning. [13]

### 1.2.2 Artificial Neuron:

This functional understanding of the neurons in our brain is translated into an artificial model that can be represented on a computer, Neurons in deep learning models are nodes through which data and computations flow.

Neurons work like this (view figure 7):

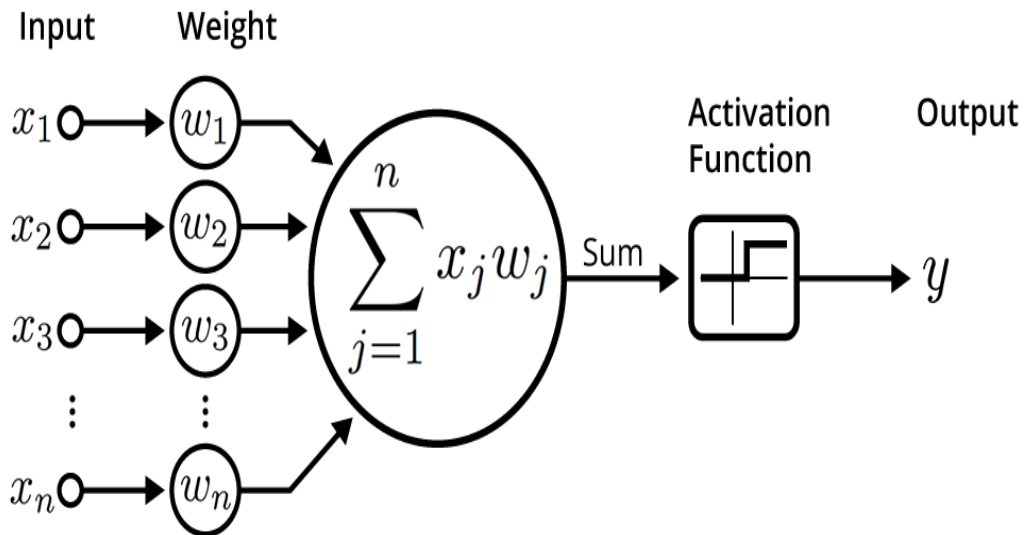
- Neurons receive one or many input signals from either the raw dataset or from the previous neuron (the previous layer) of the network.
- Neurons do some calculations.
- Finally, Neuron sends output signals to neurons in the next hidden layer through a synapse.



**Figure 7** Diagram of the Functionality of a Neuron in a Deep Learning Neural Network

Neurons in Deep Learning models have the ability of connect to more than one neuron in the preceding layer through synapses. [3]

A neuron receives its input from the previous neurons in the preceding layer of the model, then adds up signals multiplied by the corresponding weight then pass the result to an activation function [13] Figure 8 shows the complete process



**Figure 8** illustration of an artificial neuron

Mathematically, we have numbers of inputs  $x_1, x_2, x_3 \dots, x_n$ , each one of those inputs is multiplied by specific weight  $w_1, w_2, w_3 \dots, w_n$ .

The results of this multiplication are summed together to produce the logit of the Neuron:

$$\sum_{j=0}^n x_j w_j \quad (1)$$

in many cases, the logit also include bias, which is a constant:

$$\sum_{j=0}^n x_j w_j + b \quad (2)$$

This logit passed through a function  $f$  in order to produce our output  $y = f(z)$ .

We may also express this functionality in vector form, our input as a vector

$\mathbf{x} = [x_1, x_2, \dots, x_n]$ , and the weights of the neuron as  $\mathbf{w} = [w_1, w_2, \dots, w_n]$ , so our function become  $y = f(\mathbf{x} \cdot \mathbf{w} + b)$ , where  $b$  is the bias term. [13]

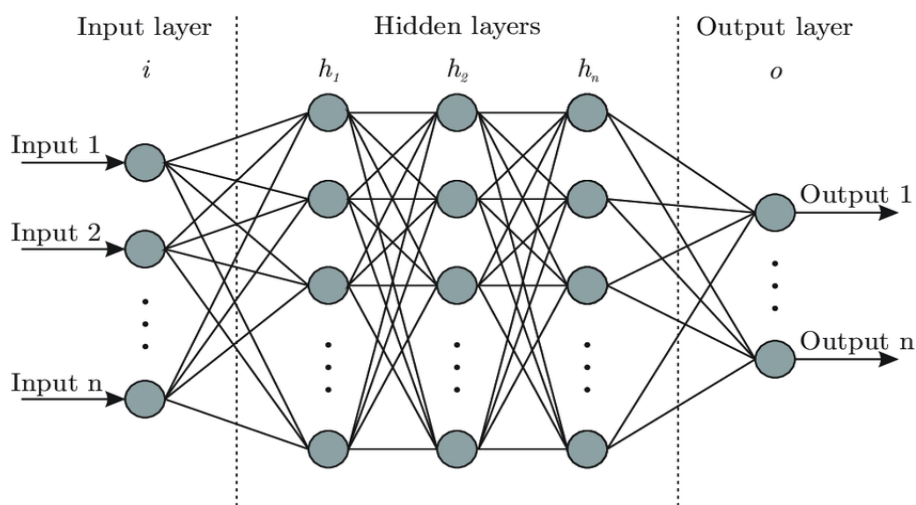
The role of the activation function is to calculate the output value of neurons, the value obtained passed through the next layer of our network using synapse.

### 1.2.3 Artificial Neural Networks

Neural networks are one type of model for machine learning; they have been around for at least 50 years. in the mid-1980s and early 1990s, many important architectural advancements were made in neural networks. However, the amount of time and data needed to get good results slowed adoption. In the early 2000s computational power expanded exponentially and the industry saw a “Cambrian explosion” of computational techniques that were not possible prior to this, this made the interest come back in Neural networks [13].

A feed-forward multilayer ANN is used. (Figure 9) shows the general ANN architecture, which has an input layer, a set of hidden layers and an output layer. In each hidden and output layer, there are artificial neurons interconnected via adaptive weights. [14]

*“Updating the weights is the primary way the neural network learns new information”.*



**Figure 9** Artificial Neural Network Architecture

The simplest type of Artificial Neural Networks ANNs was the feedforward Neural Network cause the information moves in one direction only, forward, from the input layer nodes through the nodes of the hidden layer and to the output layer nodes, Neural Network learn (update weight) by learning algorithm called Back-propagation. [14]

### 1.2.3.1 Input layer [15]

The input layer of Neural Network contains a group of artificial neurons which hold the initial data for the neural network and brings it to into the system for further processing by subsequent layers of the artificial neuron, the input layer is the very beginning of the workflow for the artificial neural network, input layers are followed by one or many hidden layers. On images processing input layer will hold the pixel intensity of the image for example an RGB image with width  $w=64$  and height  $h=64$ , and depth  $d=3$  will have an input dimension of  $64 \times 64 \times 3$ .

### 1.2.3.2 Hidden layers [15]

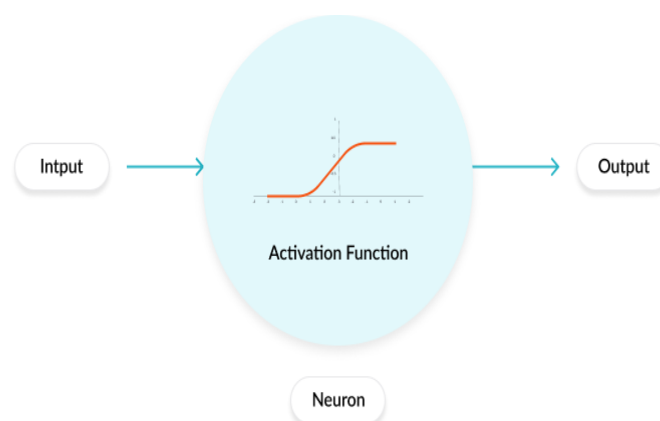
Hidden layers are an Intermediate between the input and output layer of a neural network. A hidden layer is a place where all the computation is done where artificial neurons take in a set of weighted inputs and produce outputs through an activation function. In most cases hidden neural networks layers weights are randomly assigned, sometimes they are fine-tuned (used other model's weights) and calibrated through the backpropagation process.

### 1.2.3.3 Output layer [15]

The output layer is the last layer of an artificial neural network, the output layer neurons produces output value of the network, output layer built in a different way depending on the setup of the neural network. The final output may be a set of probabilities in cases of classification or a real valued output in regression problems. The output is controlled by the type of activation function used on the output layer neurons.

### 1.2.3.4 Activation Functions

The activation function is a mathematical gate between the input which is a value coming from the previous neuron and the output which is the value going to the next layer neurons (Figure 10), we can describe it as a function that turn the neuron output on or off depending on the rule applicated [15].

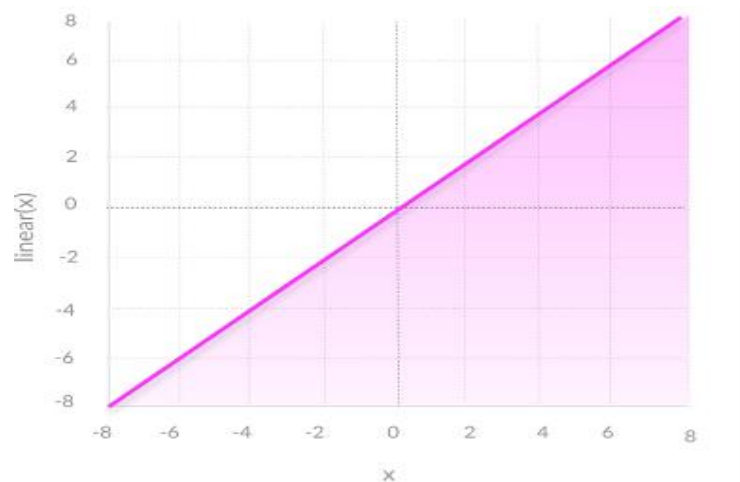


**Figure 10** Activation Function

There are 2 type of activation function that can be used on neural network linear activation functions and no-linear activation function, the second one is the most used because it can help network to learn complex data, Activation function has also the ability of filter out data here some usual activation functions:

#### 1.2.3.4.1 Linear Activation Function

A linear activation function takes the form:  $f(x) = cx$



**Figure 11** Linear Activation Function

This create a signal output identic to the input by taking the inputs and multiply it by the weight corresponding to each neuron. In one sense, a linear function allows multiple outputs, not just yes and no, so we can say that is better than a step function.

Mathematically we can say that the neuron receive input  $x_1, x_2, x_3 \dots x_n$  the output of the linear neuron is given by:

$$y = w_1x_1 + w_2x_2 + w_3x_3 \dots + w_nx_n + b \quad (3)$$

Where  $w_1, w_2, w_3 \dots w_n$  are the weight corresponding to  $x_1, x_2, x_3 \dots x_n$  respectively and  $b$  is the bias. [15]

linear activation function can't use backpropagation and gradient descent in order to train models the derivative of the function is always constant and it has no relation to the input so there is no way to go back through the backpropagation process to understand which weight in our input neuron way provide a better prediction.

we can say that a neural network with a linear activation function is a regression model. Linear activation function has a limited ability to handle with complex varying parameters of input data.

### 1.2.3.4.2 Non-Linear Activation Function [15]

Instead of using linear activation function modern models use non-linear activation function in order to create a complex mapping between the network's inputs and outputs, image processing and dataset that have high dimensionality.

Non-linear activation functions solve the problems of the linear-activation function

- Non-linear activation function allows backpropagation process because they have a derivative function, the derivative of a linear function is always 0.
- Non-linear activation function gives high accuracy comparing to the linear one

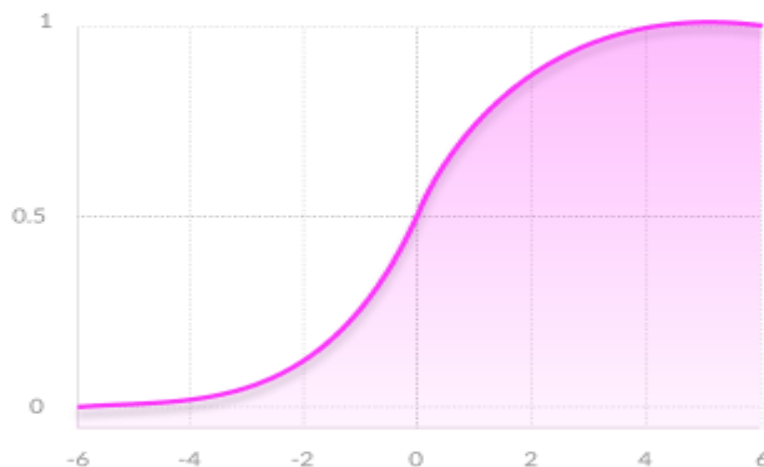
Those are the most used non-linear activation function

#### a) Sigmoid [15]

which uses the function:

$$f(x) = \frac{1}{1 + e^{-x}} \quad (4)$$

The Sigmoid function assumes an S-shape (Figure 12), if the input is small then the output of the sigmoid function is close to 0, otherwise, a large value gives an output closer to 1.



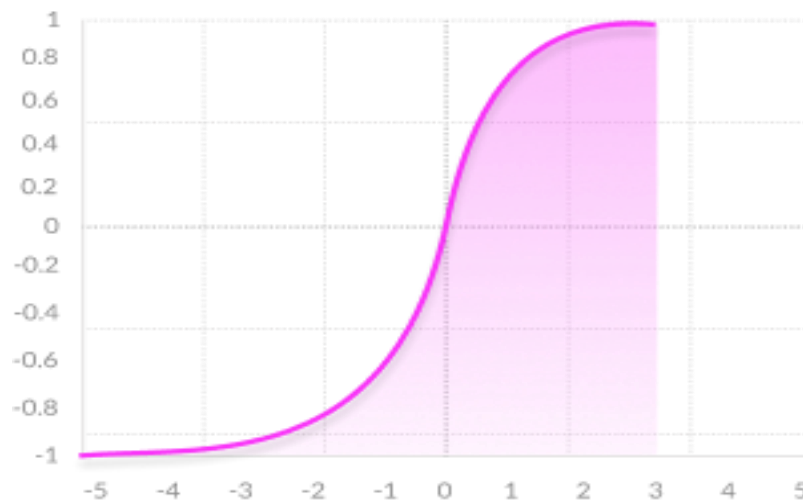
**Figure 12** Sigmoid Function

**b) Tanh [13]**

Which is similar to sigmoid function but instead of ranging from **0** to **1**, the output of tanh range from **-1** to **1**, use  $f(x) = \tanh(x)$  it's the ratio of the hyperbolic sine to the hyperbolic cosine:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} \quad (5)$$

The graph of tanh function is similar to the sigmoid function (Figure 13)

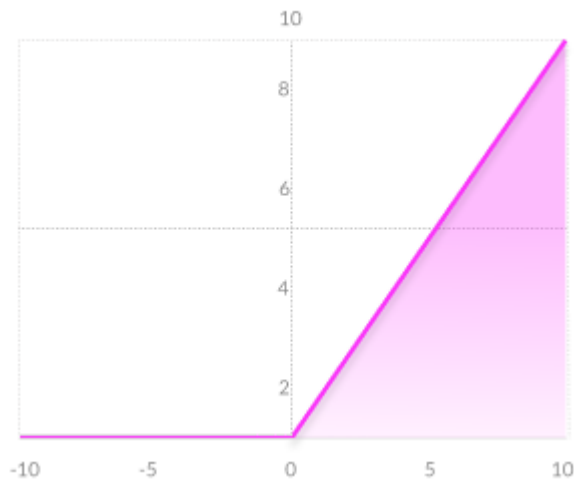


**Figure 13** Tanh Hyperbolic

During the backpropagation step the gradients become smaller and smaller until eventually they vanish, no gradients means no learning this is called the vanish gradient problem. It happens because of the sigmoid function it squeezes information. The solution of this problem is to use an activation function that doesn't squeeze information like RELU [16].

### c) ReLU [16]

Rectified Linear Unit use the function  $f(x) = \max(0, x)$ , ReLU is probably the most used activation function in the world now. It used in almost all the Convolutional Neural Network (CNNs) or deep learning.

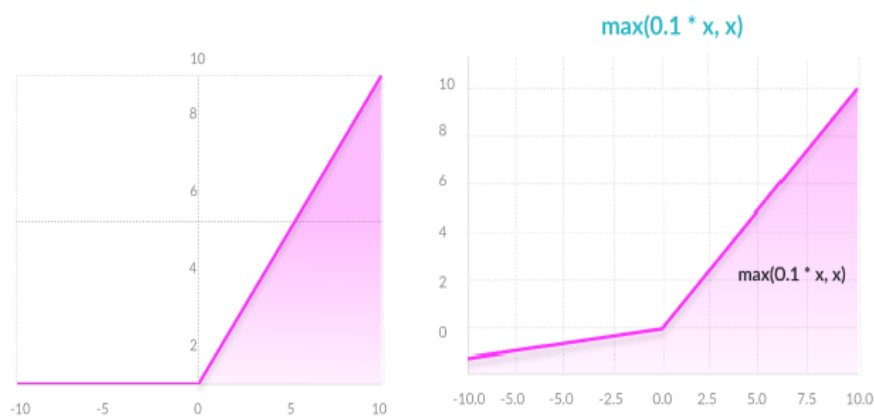


**Figure 14** ReLU Graph

As we can see,  $f(x)$  is zero when  $x$  is less than zero and  $f(x)$  is equal to  $x$  when  $x$  is above or equal to zero.

As we can see all the negative values become zero immediately, that may decrease the ability of the model to fit or train from the data properly because the ReLU block all the inputs less than zero that's called "dying ReLU problem" introducing some activation even in the negative cases solve this problem

Leaky ReLU is an attempt to solve the dying ReLU problem



**Figure 15** ReLU vs Leaky ReLU

Leaky ReLU have handled this problem pretty well.



### d) SoftMax [17]

SoftMax (Figure 16) handles the activation of the output neuron, we can use SoftMax to solve classification problems. the number of classes equal to the number of neurons in the last layer the value obtained from the SoftMax represents the probability of belonging to a particular class.

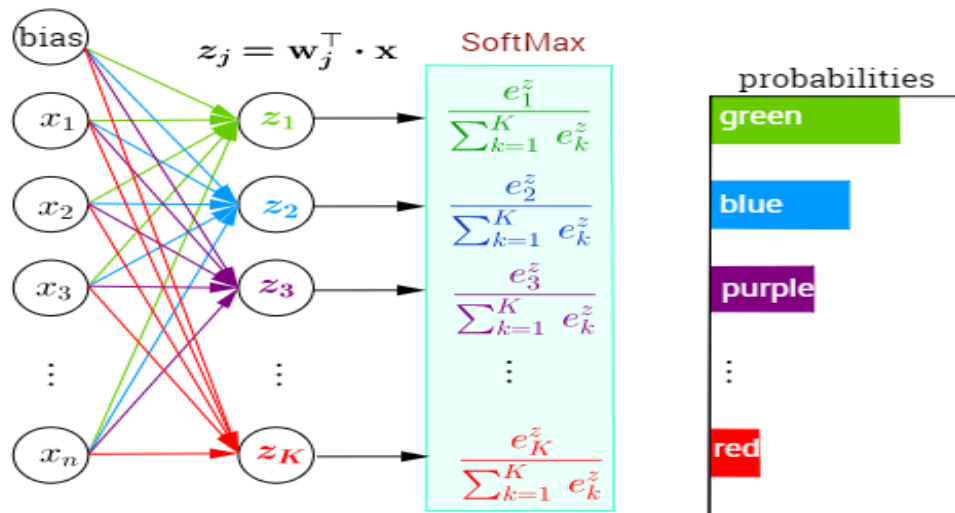


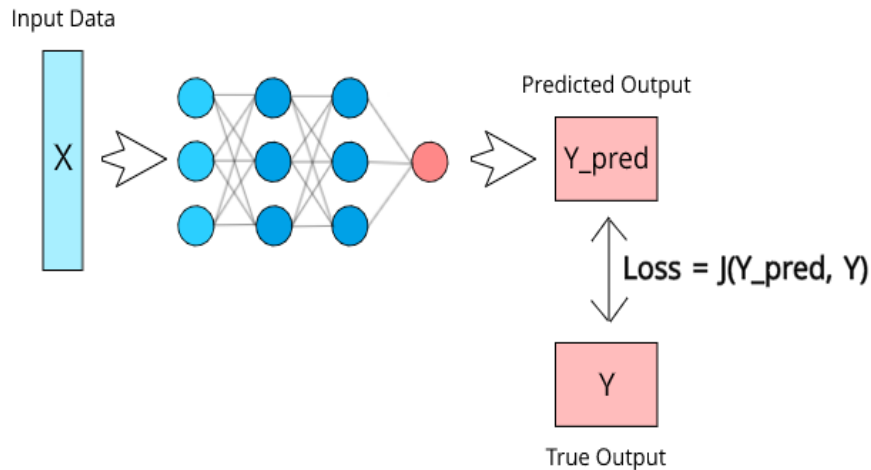
Figure 16 SoftMax

A strong prediction mean that one output is too close to 1, while the other output obviously close to 0, Otherwise, our prediction is weak.

#### 1.2.3.5 Loss Functions

loss function quantifies how close given neural network is to the ideal toward which is training. [18] In deep learning project, configuring the loss function is one of the most important steps to ensure the model will work in the intended manner. The loss function can give a lot of practical flexibility to the neural network.

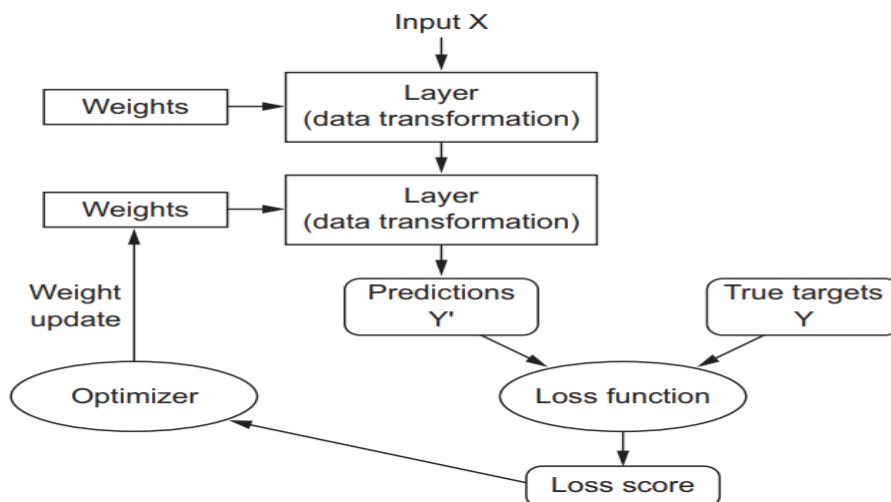
There are serval tasks neural network can perform, from predicting continuous values to classifying discrete classes. Each different task would require a different type of loss function since the output format will be different. For specialized tasks, it's up to us how we want to define the loss. The loss function (Figure 17) can be defined as a function with two parameters: Predicted Output and the True Output



**Figure 17** Neural Network Loss Visualization

the function above calculate how poorly our model is performing by comparing the actual value that we supposed to get as output and what the model is predicting, in the case of  $Y_{pred}$  value is very far from  $Y$  the loss will be high, and if the 2 values are similar the loss value will become low [3].

If the loss is very high, this huge value will propagate through the network while it's training and the weight will be changed let's say optimized a little more than usual (Figure 18). If the loss is small than the weight will not change a lot since the network is already doing a good job [17].



**Figure 18** Neural Network Workflow

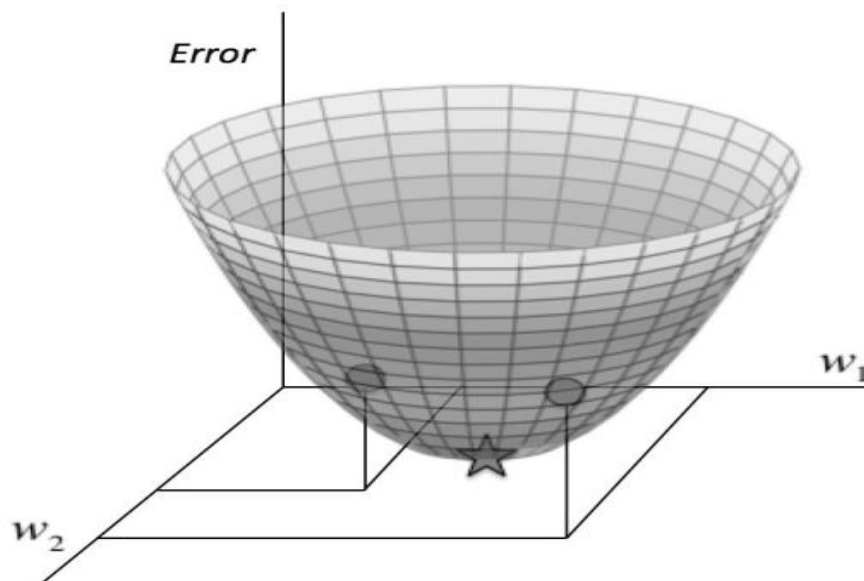
The objective is to find the optimal weights possible for our network, the weights that minimize the error, the way to do that is by applying an optimization algorithm like gradient descent.

If we put  $t^{(i)}$  is the true answer for the  $i^{\text{th}}$  training sample and  $y^{(i)}$  is the value obtained by the Neural Network  $E$  is the function that minimize the value of the square error: [13]

$$E = \frac{1}{2} \sum_i (t^{(i)} - y^{(i)})^2 \quad (6)$$

#### 1.2.4 Gradient Descent [13]

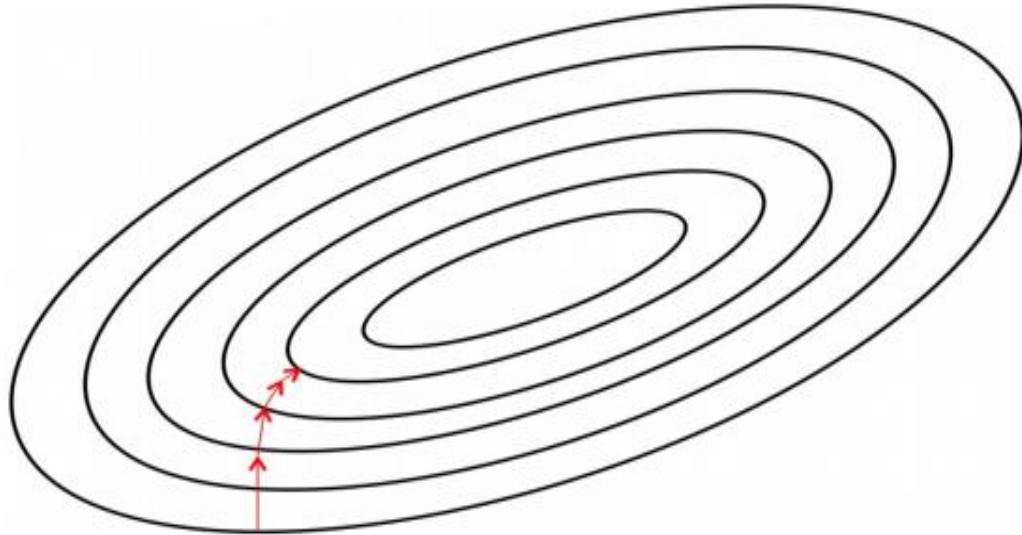
Gradient descent is an optimization technique used to minimize the error by calculating the gradient necessary in order to update the value of neural network parameters [6]. Let's visualize how we might minimize the squared error over all of the training examples by simplifying the problem. Let's say our linear neuron only has two inputs (and thus only two weights,  $w_1$  and  $w_2$ ). Then we can imagine a three-dimensional space where the horizontal dimensions correspond to the weights  $w_1$  and  $w_2$ , and the vertical dimension corresponds to the value of the error function  $E$ . In this space, points in the horizontal plane correspond to different settings of the weights, and the height at those points corresponds to the incurred error. If we consider the errors, we make over all possible weights, we get a surface in this three-dimensional space, in particular, a quadratic bowl as shown in Figure 19



**Figure 19** The Quadratic Error Surface for a Linear Neuron

now we are able to develop a strategy for how to find the values of weights that minimizes the error function, the weights are randomly initialized for our network so we find ourselves somewhere in the horizontal plane. By evaluating the gradient at our current position, we can find the direction of steepest descent, and then take a step on that direction, now we find ourselves in at a new position closer to the minimum than before. By taking the gradient at this new direction we can reevaluate the direction of steepest descent and taking a step in this direction as shown in Figure 20, by following this strategy

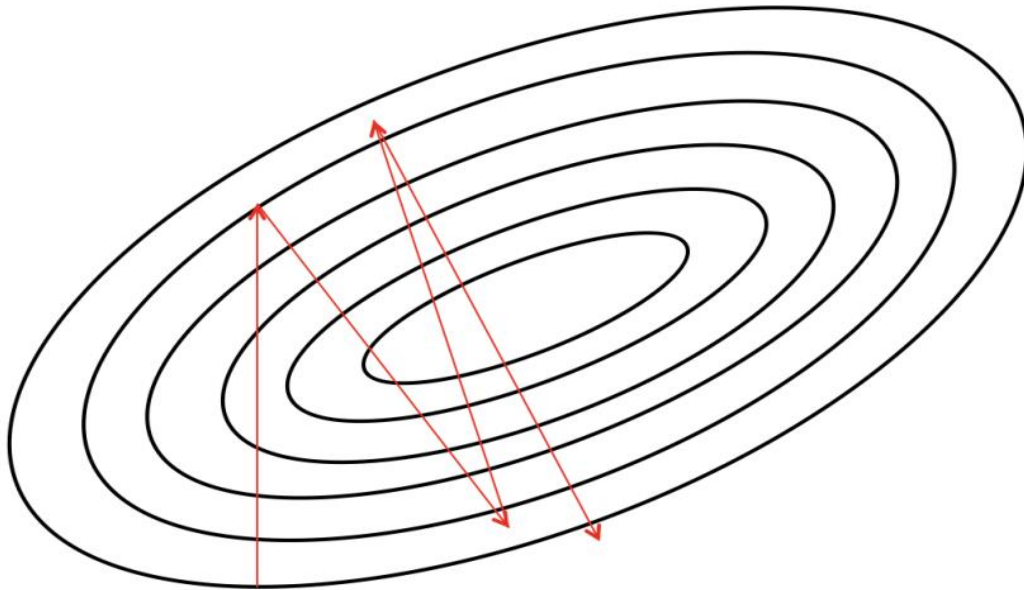
it will get us to the point minimum (minimum error), this is known as the Gradient Descent Algorithm, it used to tackle the problem of training individual neurons and the more general challenge of training entire networks.



**Figure 20** Visualizing the Error Surface as a set of Contours

The learning rate is of the most important hyperparameter in this process.

We need to determine how far we want to walk before recalculating our new direction. This distance need to depend on the steepness of the surface because the closer we are to the minimum the shorter we want to step forward, the closer to the minimum we are the more flatter our surface become, so we can use the steepness as an indicator of how close we are to the minimum, however, if our surface is rather mellow, training can take a large amount of time. As a result, we often multiply the gradient by a factor  $\epsilon$ , the learning rate. Picking the learning rate is a hard problem (Figure 21). If we pick a small learning rate, we risk taking too long during the training process but if we pick a big value for the learning rate we'll mostly likely start diverging away from the minimum.



**Figure 21** Convergence is Difficult when our Learning Rate is too Large

Now, we are finally ready to derive the *delta rule* for training our linear neuron. In order to calculate how to change each weight, we evaluate the gradient, which is essentially the partial derivative of the error function with respect to each of the weights. In other words, we want:

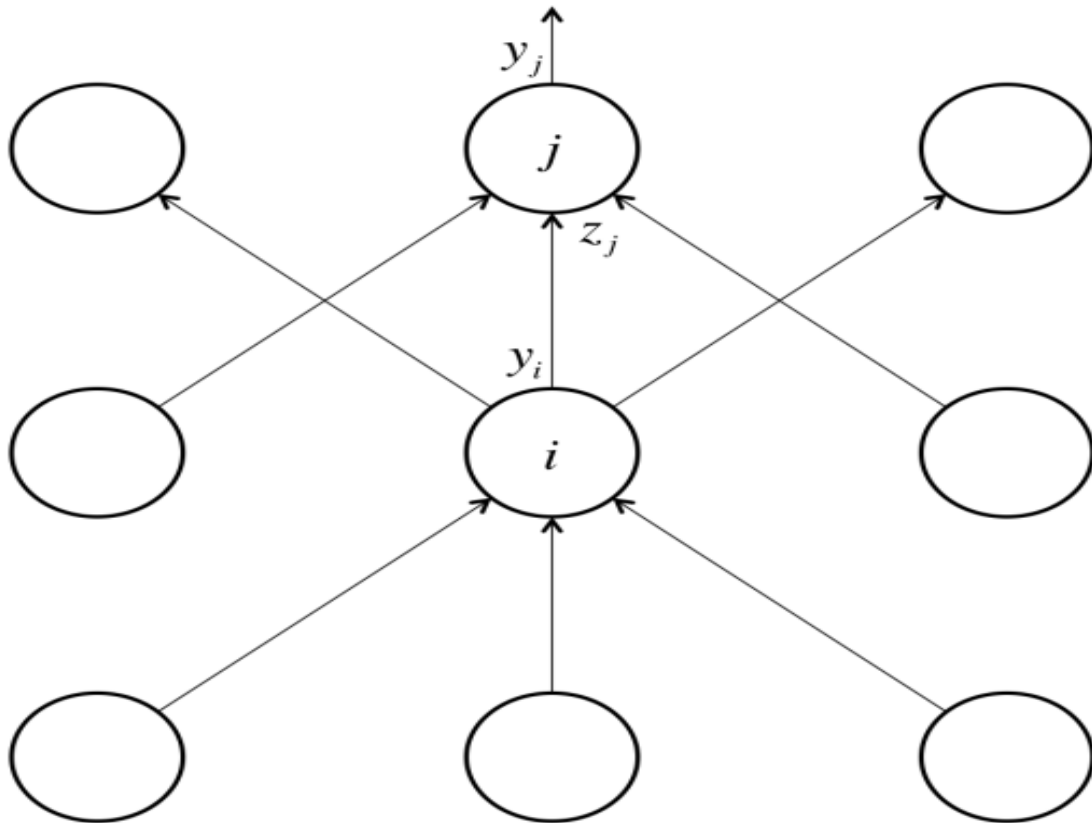
$$\begin{aligned}
 \Delta w_k &= -\epsilon \frac{\partial E}{\partial w_k} \\
 &= -\epsilon \frac{\partial}{\partial w_k} \left( \frac{1}{2} \sum_i (t^{(i)} - y^{(i)})^2 \right) \\
 &= \sum_i \epsilon (t^{(i)} - y^{(i)}) \frac{\partial y_i}{\partial w_k} = \sum_i \epsilon x_k^{(i)} (t^{(i)} - y^{(i)})
 \end{aligned} \tag{7}$$

By applying this function, we are finally able to use gradient descent

### 1.2.5 The Back-Propagation Algorithm [13]

Back propagation is a technique used to train multilayer neural network, by David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams in 1986. [19] The main idea behind backpropagation is to compute how fast the error changes as we change a hidden activity. From there we can figure out how fast the error changes when we change the weight of an individual connection. Essentially, we'll be trying to find the path of steepest descent! The difference now is that we are going to work in an extremely high-dimensional space instead of two weight in the previous example. We start by calculating the error derivatives with respect to a single training example.

Each hidden unit can affect many output units. Thus, we need to combine many separate effects on the error in an informative way. Our strategy will be one of dynamic programming. Once we have the error derivative for one layer of hidden units, we'll use them to compute the error derivative for the activities of the layer below. And once we find the error derivatives for the activities of the hidden units, it's quite easy to get the error derivative for the weights leading into the hidden unit. We'll redefine some notation for ease of discussion and refer to (Figure 22) the subscript we used refer to the layer of the neuron,  $y$  refers to the activity of the neuron,  $z$  refers to the logit of the neuron.



**Figure 22** Reference Diagram for the Derivation of the Backpropagation Algorithm

Now we start by taking a look at the base of the dynamic programming problem. Specifically, we calculate the error function derivative at the output layer:

$$E = \frac{1}{2} \sum_{j \in \text{output}} (t_j - y_j)^2 \Rightarrow \frac{\partial E}{\partial y_j} = -(t_j - y_j) \quad (8)$$

Now we tackle the inductive step. Let's presume we have the error derivatives for layer  $j$ . We now aim to calculate the error derivatives for the layer below it, layer  $i$ . To do so, we must accumulate information about how the output of a neuron in layer  $i$  affects the logits of every neuron in layer  $j$ . This can be done as follows, using the fact that the

partial derivative of the logit with respect to the incoming output data from the layer beneath is merely the weight of the connection  $w_{ij}$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E \partial z_j}{\partial z_j \partial y_i} = \sum_j w_{ij} \frac{\partial E}{\partial z_j} \quad (9)$$

Furthermore, we observe the following:

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j} = y_j(1 - y_j) \frac{\partial E}{\partial y_j} \quad (10)$$

Combining these two together, we can finally express the error derivatives of layer  $i$  in terms of the error derivatives of layer  $j$ :

$$\frac{\partial E}{\partial w_{ij}} = \sum_j w_{ij} y_j(1 - y_j) \frac{\partial E}{\partial y_j} \quad (11)$$

Then once we've gone through the whole dynamic programming routine, having filled up the table appropriately with all of our partial derivatives (of the error function with respect to the hidden unit activities), we can then determine how the error changes with respect to the weights. This gives us how to modify the weights after each training example:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j} = y_i y_j(1 - y_j) \frac{\partial E}{\partial y_j} \quad (12)$$

Finally, to complete the algorithm, just as before, we merely sum up the partial derivatives over all the training examples in our dataset. This gives us the following modification formula:

$$\Delta w_{ij} = - \sum_{k \in \text{dataset}} y_i^{(k)} y_j^{(k)} (1 - y_j^{(k)}) \frac{\partial E^{(k)}}{\partial y_j^{(k)}} \quad (13)$$

### 1.2.6 Convolutional Neural Networks (CNNs / ConvNets)

Convolutional neural networks CNN often called ConvNet, are a family of models that were inspired by how the visual cortex of human brain works when recognizing objects. The development of CNN's goes back to the 1990's, when Yann LeCun and his colleagues proposed a novel neural network architecture for classifying handwritten digits from images. [20] Due to the great performance of CNNs specially for image classification tasks, CNNs gained a lot of attention, this leads to tremendous improvements in Machine Learning and computer vision applications [21]

ConvNets has deep feed-forward architecture and has astonishing ability to generalize in a better way as compared to networks with fully connected layers [10], it can learn highly abstract features and can identify objects efficiently. CNN is considered above other classical models because, CNN can be trained smoothly and does not suffer overfitting and it is much difficult to implement large networks using general models of Artificial Neural Network (ANN) than implementing in CNN. CNNs are widely being used in various domains due to their remarkable performance such as object detection, speech recognition, face detection, facial expression recognition, natural language processing, **image classification** and many more. The main concept of CNNs is to obtain local features from input (usually an image) at higher layers and combine them into more complex features at the lower layers. [11]

#### 1.2.6.1 Convolution Operation

Kernel convolution isn't only used in ConvNets it's also used on many other computer vision algorithms. It's a simple process where we need to take a small matrix (Kernel/Filter), then pass through the target image and transform it based on kernel values, the result of this operation is a feature map. Feature map values are calculated by the following formula:

$$G[m, n] = (f \times h)[m, n] = \sum_j \sum_k h[j, k]f[m - j, n - k] \quad (14)$$

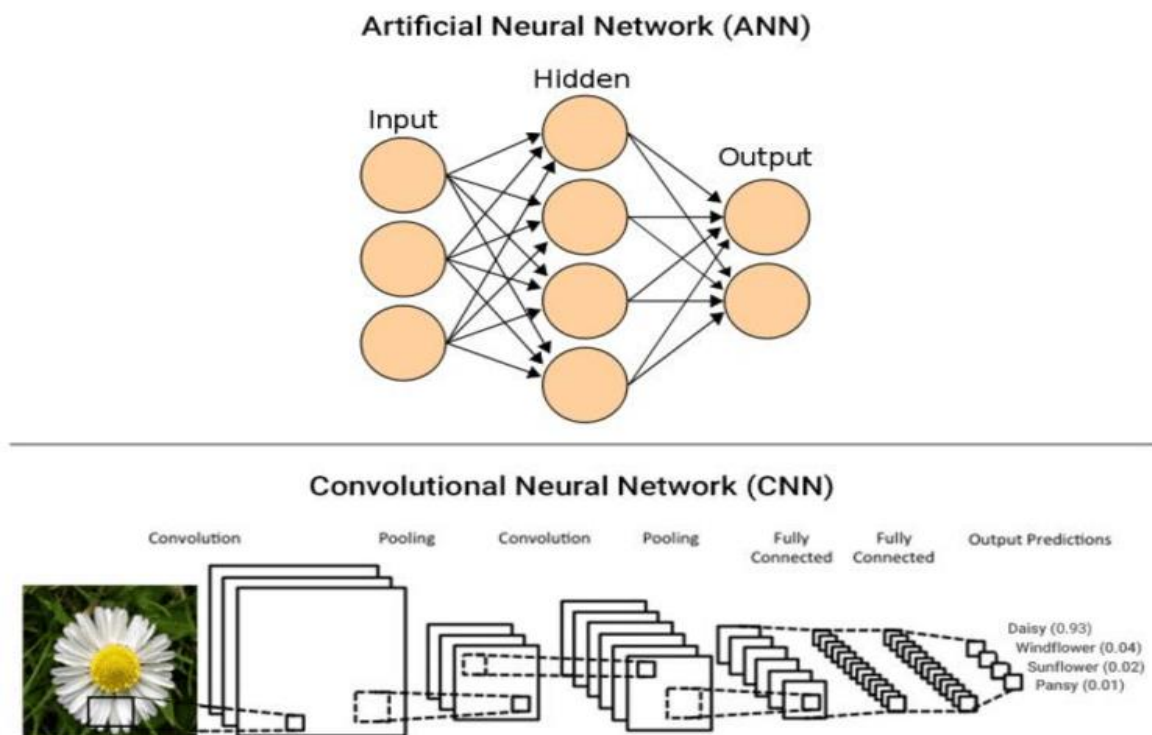
Our input image is denoted with  $f$  and our kernel by  $h$ . The indexes of rows and columns of the result matrix are marked with  $m$  and  $n$  respectively. [15]

First of all, we need to place the filter over a selected pixel (region of the input matrix), then multiplying each value from the kernel with the corresponding value from the image (input matrix) using a dot product, not matrices multiplication. Finally, we sum up the results we get then place it in the right place in our feature map



### 1.2.6.2 Architecture of CNN

In traditional neural networks, each hidden layer is made up of a number of neurons, where each neuron is fully connected to all neurons in the preceding layer. The major difference between a traditional Artificial Neural Network (ANN) and Convolutional Neural Network CNN is that only the last layer of a CNN is fully connected whereas in ANN, it means that each neuron is connected to every other neuron (full connection) as shown in (Figure 23) [22]



**Figure 23** Artificial Neural Network and Convolutional Neural Network

ANNs are not appropriate to images it leads to over-fitting easily due to image size. Consider an image of size  $[32 \times 32 \times 3]$ . If this image is passed through an ANN, it will be flattened into a vector of size  $32 \times 32 \times 3$  which means 3072 rows. so, our ANN must have 3072 weights in its first layer to receive this input vector. For larger images, say  $[300 \times 300 \times 3]$ , it results in a complex vector (270,000 weights), which requires a more powerful processor to process. [22]

All CNN fundamentals are based on three properties: local connectivity, parameter (weight) sharing, pooling and sampling of hidden units.

#### a) Local Receptive Field

Also known as local connectivity we have already explained that the ANN neurons are fully connected, as for CNN have local receptive field architecture, that's means each hidden unit can only connect to a small region of the input called local receptive field. This is accomplished by making the filter/weight matrix smaller than the input. Using the local

receptive field, the neurons will be able to extract elementary visual features like point and corners, etc. [23]

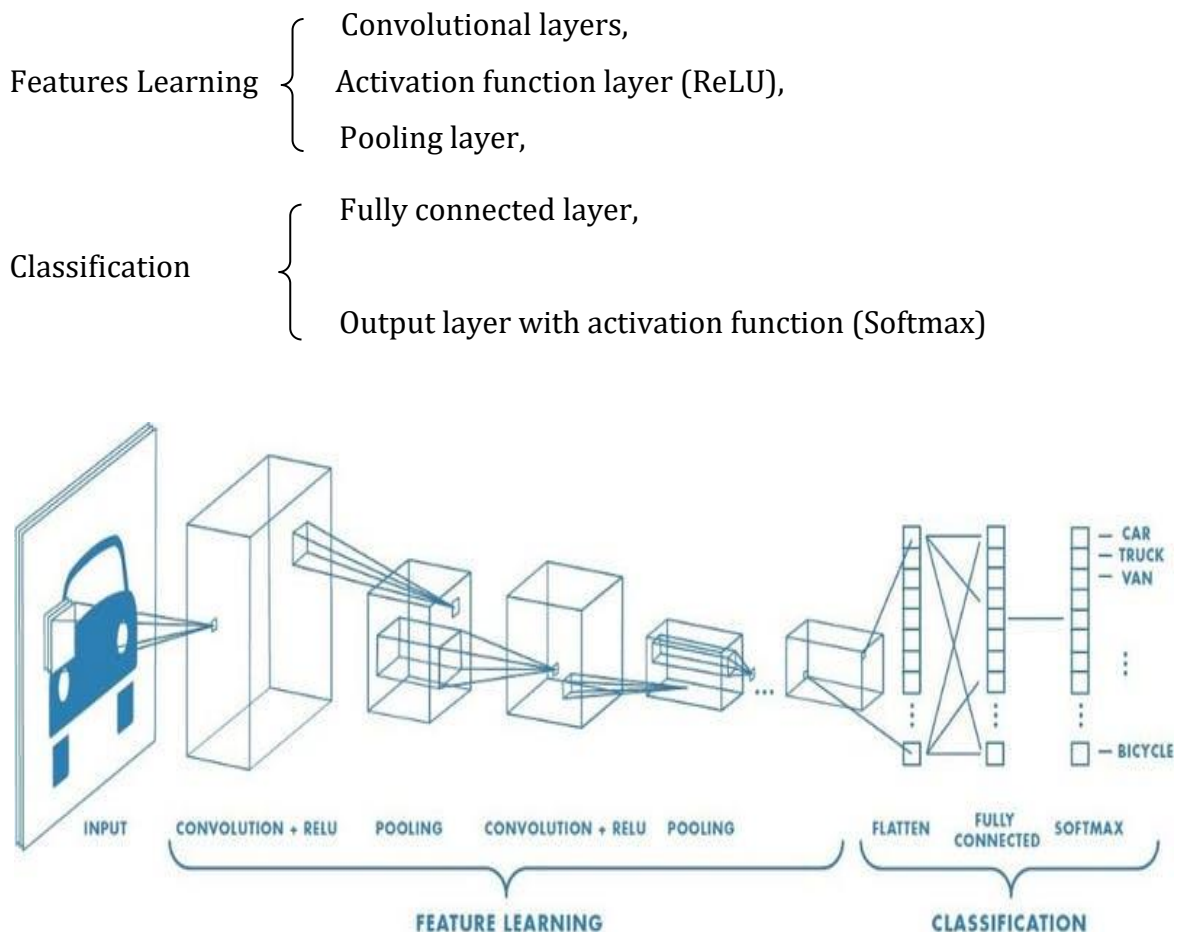
### b) Weights sharing

In CNN, the matrix of parameter (kernel or weight matrix) is shared between the hidden units organized in the same feature map. Hidden units within a feature map cover different position(part) of the image, same filter can be used for all local receptive fields.

### c) Subsampling (pooling)

Pooling and sampling of hidden units, subsampling reduces the size of the input, in order to improve the computation. There are many techniques used, the most common subsampling technique is max-pooling. [15]

Convolutional Neural Network is based on a sequence of layers to achieve different tasks. The figure below shows the architecture of a typical ConvNet that contains the following layers divided on two-part Features Learning and Classification:

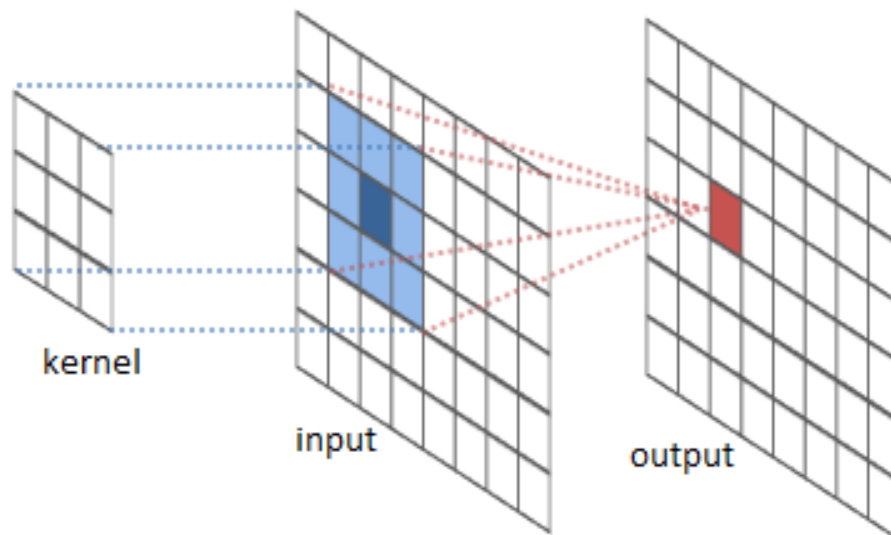


**Figure 24** Convolutional Neural Network Architecture

These layers are connected together to make a full Convolutional Neural Network architecture, Convolutional and activation layer followed by an optional pooling layer. Fully connected layer makes up the last layer of the CNN, [22] using a SoftMax function (Sigmoid function may be used on binary classification) in the last layer produces the probability of which class included our input when it comes to a classification problem like the previous figure. [15]

### 1.2.6.2.1 Convolution layer

Network which uses convolutional operation (\*) which called also an element-wise product used instead of general matrix multiplication who takes too long. The Convolutional Layer consists of a set of filters (kernel or feature detector), where each filter is applied across all areas of the input data (the image) (Figure 25). A filter is defined by a set of learnable weights. [6] The number of feature maps is equal to the specified number of filters. [15]

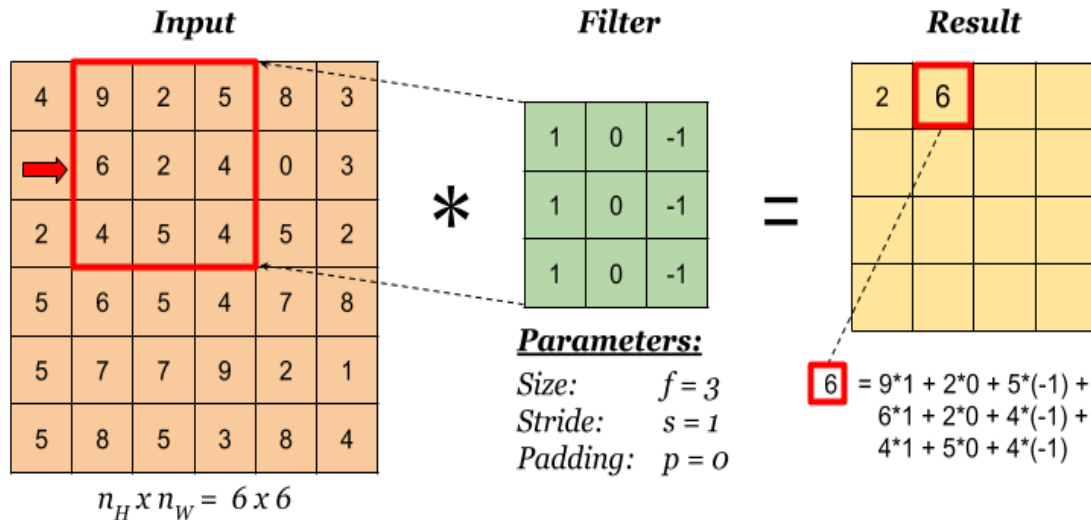


**Figure 25** Convolutional Operation

The main task of convolutional layer is to detect features found within local region of the input image. A feature map is obtained for each filter.

### 1.2.6.2.1.1 Filters/Kernels

Each filter contains some features like corners, edge, and during the pass, the filter is slide across the width and height (according to the stride parameter) of the input generating feature map of that filter, they may be multiple kernels in each convolutional layer. [23]



**Figure 26** Example of Convolutional Operation

A feature map is obtained after adding a bias term and then applying a nonlinear function to the output of the convolutional operation.

### 1.2.6.2.1.2 Hyperparameters

The convolutional and pooling layers have hyperparameter whose value must be defined beforehand, they are used to control the behavior of the model, here some important hyperparameters in the convolutional layer of the CNN:

#### a) Filter Size

Filter can take any size greater than  $2 \times 2$  [23], it should be less than the size of the input. The largest size used is  $7 \times 7$  but only in the first convolutional layer, [13] a 2D convolutional filter will always have a third dimension in size. The third dimension is equal to the number of channels of the input image. For example, we apply a  $3 \times 3 \times 1$  convolution filter on gray-scale image that has 1 black and white channel like the previous example (Figure 26). We apply a  $3 \times 3 \times 3$  convolution filter on a colored image with 3 channels, Red, Green and Blue (figure 29).

In general, each image has dimensions  $W \times H \times D$  where  $W$  is the width in pixels,  $H$  is the height in pixels and  $D$  represent the dimension or the depth which is the number of channels. [23]

#### b) Number of filters

There can be any reasonable number of filters, GoogLeNet has 128 filters of  $3 \times 3$  kernel size and 32 filter of  $5 \times 5$  size, AlexNet used 96 filters of size  $11 \times 11$  in the first convolution layer. [6]

**c) Stride**

It governs how many cells the filter is moved in the input to calculate the next cell in the result, that means the number of pixels to move at a time to define the local receptive field for filter (Figure 27), too small stride will lead to overlapping receptive field and the large one resulting output with smaller dimension. [6]

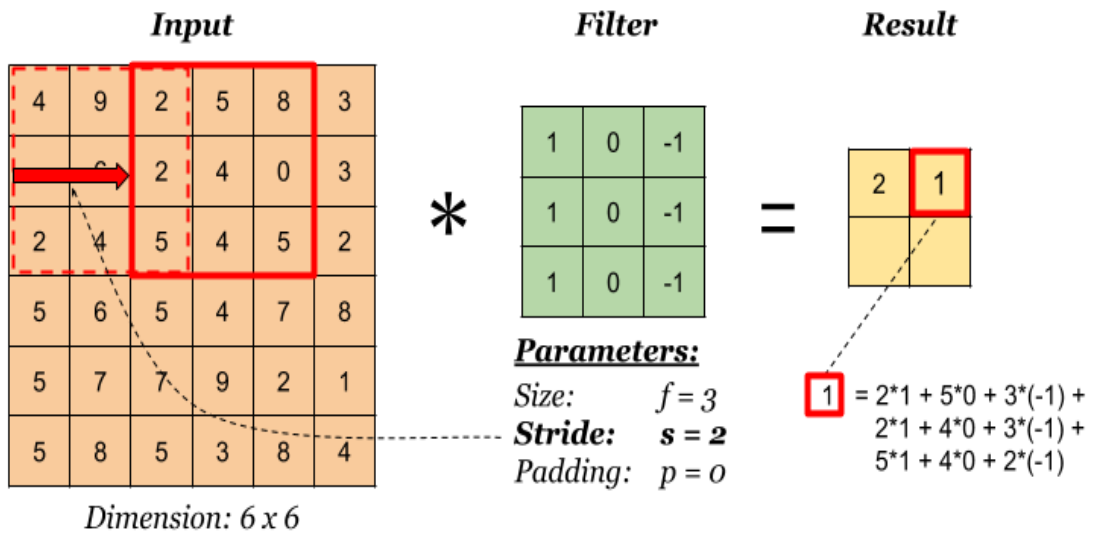


Figure 27 Filter with Stride (s) = 2

**d) Zero padding**

This hyperparameter describes the number of pixels to pad the input image (matrix), [6], we add to the image a padding with p pixel(Figure 28). It helps to keep more of the information at the border of an image [13]. Without padding, very few values at the next layer would be affected [6].

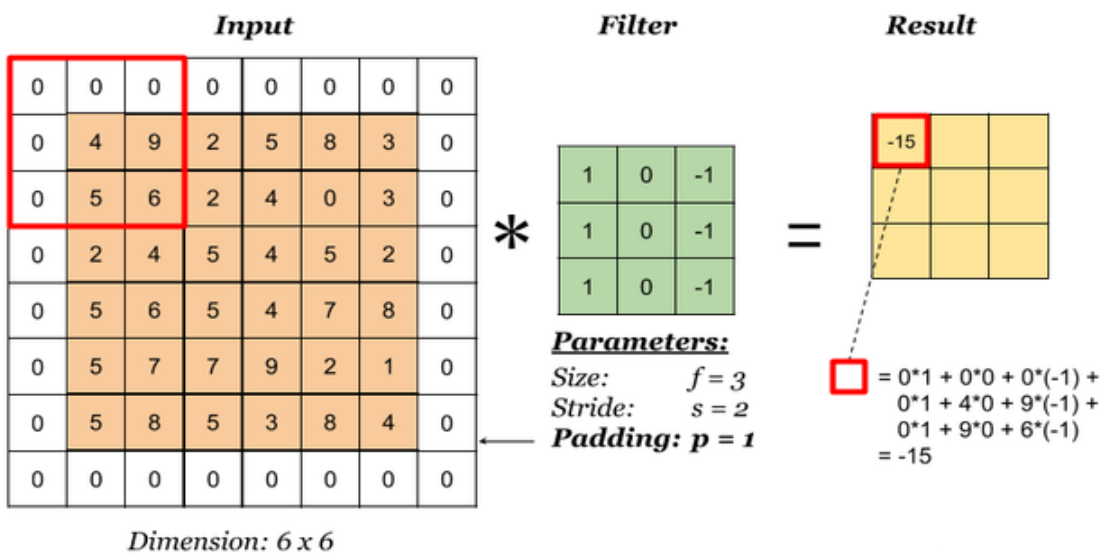


Figure 28 Zero Padding example with (p)=1

Notice that the dimension of the result has changed due to padding if we compare it with the previous example (figure 29)

Each filter in the convolution layer produce a feature map of size  $([A - K + 2P]/S) + 1$ , where: A the input volume size, K size of the filter, P the number of padding applied and S the stride. [6]

Suppose the input image has size  $6 \times 6 \times 3$ , and 3 filters of size  $3 \times 3$  are applied, where stride  $s = 1$  and padding  $p = 0$  (figure 7), we already say that the number of feature maps generated equal to the number of filters/kernels applied i.e. 3. the size of each feature map will be  $(\frac{[6-3+0]}{1}) + 1 = 4$ , therefore, the output volume will be  $4 \times 4 \times 3$ . Convolution of 3D image will give a 2D output.

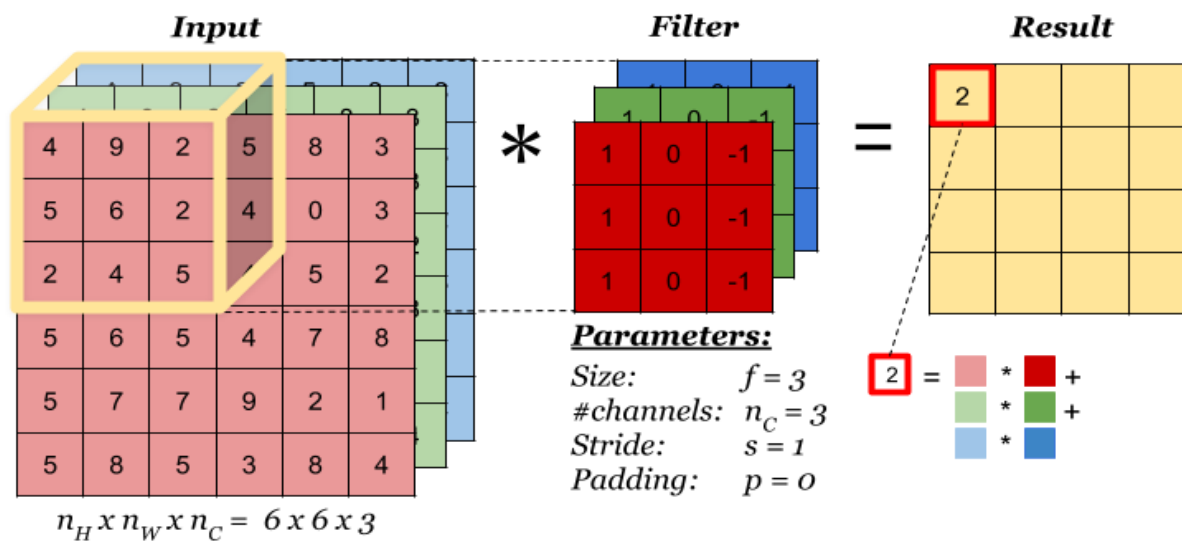
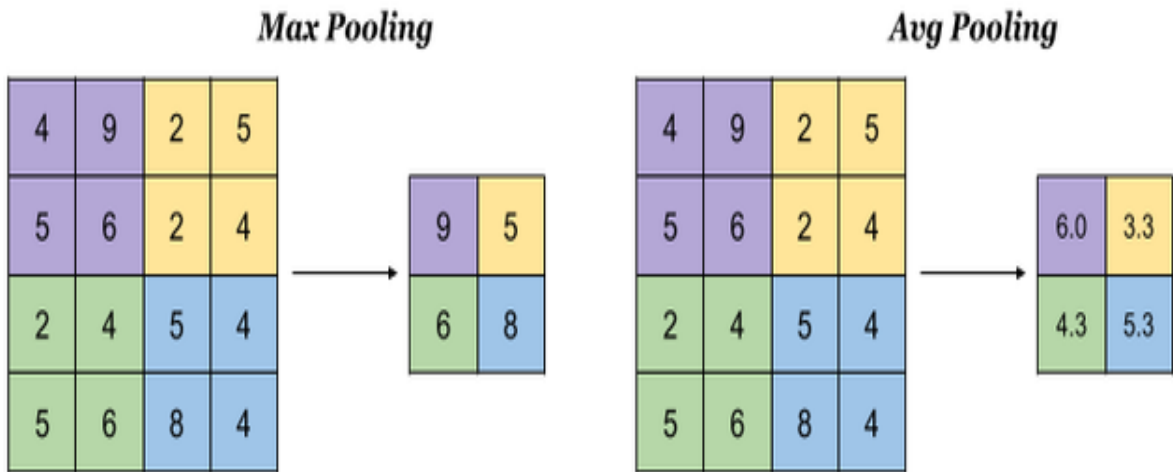


Figure 29 Convolution Operation on Volume

### 1.2.6.2.2 Pooling Layer

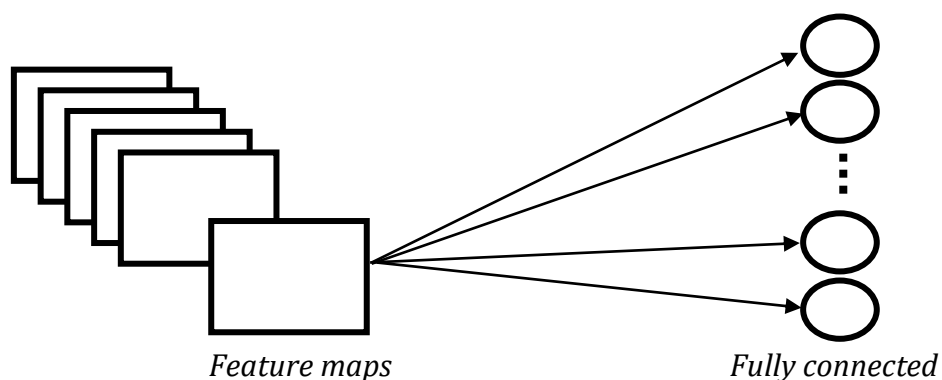
Subsampling layers commonly known as pooling layers, do not have any learnable parameters, for instance, there are no weights or bias units in pooling layers, in CNNs, the sequence of convolution layers and activation function layer is followed by an optional pooling layer [24] to reduce the spatial size of the input this will reduce the number of parameters in the network. [6] Pooling layer makes the model more robust to variations in the position of the features in the input image by taking each feature map output from the convolutional layer and down-sample it, it summarizes a region of neurons in the convolution layer. Many pooling technique max and average pooling are the most common pooling technique used, [23] (Figure 30) it's a pooling operation that select the maximum element (value) from the region of the feature map covered by the filter, the output after max-pooling layer would be a feature map containing the most important features of the previous feature map and discards less signification data. The average pooling calculates the average instead of take the maximum value from the input matrix.



**Figure 30** Max Pooling and Avg Pooling

### 1.2.6.2.3 Fully Connected Layer [6]

In ConvNets the previous sequence (Convolutional Layer, Pooling layer) is followed by a fully connected layer. Convolutional neural network composed of two stage: Feature extraction and classification stage, the stack of convolutional layer and the pooling layer represent the part of the feature extraction, while the classification stage is composed of the fully connected layer (one or more) followed by a SoftMax function layer. The main role of the first part is to detect enough features from input images. The role of the last layer which probably composed of Softmax function will calculate the probability that these features represent each class that mean obtain the class score. each neuron from previous layer (convolution layer or pooling layer or fully connected layer) is connected to every neuron in the next layer and every value contributes in predicting how strongly a value matches a particular class. (Figure 31) fully connected layer can learn more sophisticated combinations of features. the two main classifiers used in CNNs are Softmax and Support Vector Machines (SVMs) as we said Softmax produce the probabilities of each class with total of probability of 1, SVM produce the class scores, the class having the highest score is the treated as the correct one.



**Figure 31** Connection between Convolutional Layer and Fully Connected Layer

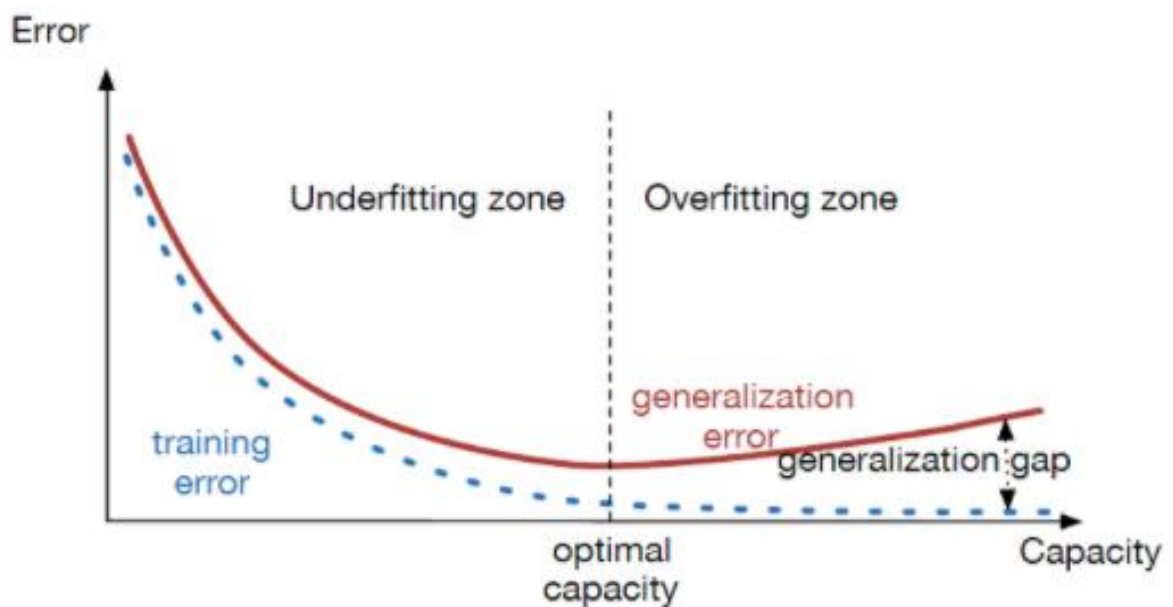


### 1.3 Regularization for Deep Learning

Regularization for deep learning is a collection of techniques that prevent overfitting problems. This technique can improve the accuracy of models, but the problem is that the model shows good performance on the training set and a bad performance on the test data (new data). We can say that the model in this case can't generalize well what it has learned to new data. Many strategies are used in machine learning to solve these problems, and these strategies are known as regularization. [9]

#### 1.3.1 Underfitting and overfitting

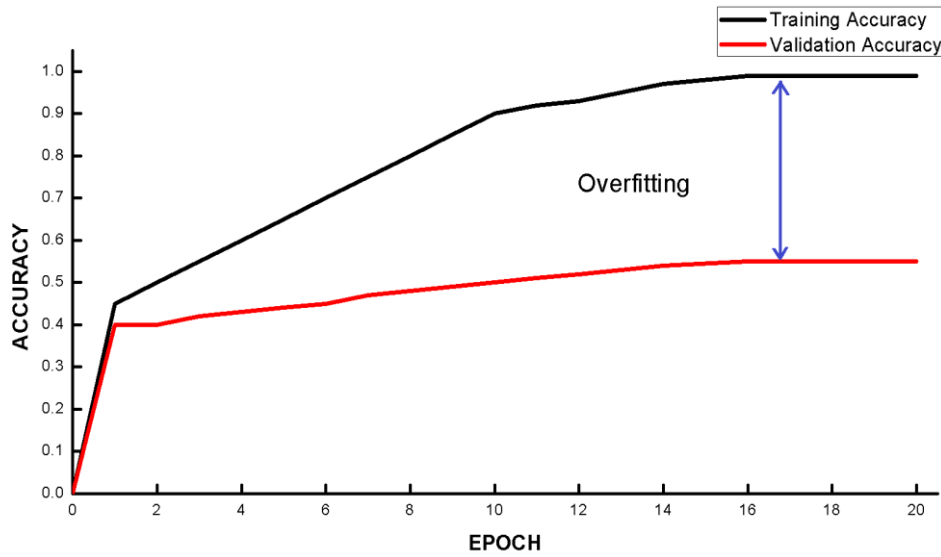
There are two major problems: the first one is underfitting and the second one is overfitting. The first one is when the error rate for the training data is high, which obviously means that the error rate for the test data will be high also. Many experts said that if the model accuracy in the training set is less than 75%, that means that the model architecture needs to be changed or configured. The second problem is the worst; it happens when the model shows a great performance on the training data but the accuracy is low for the test data (validation data) (Figure 32). That happens when the model overfits the data but it can't generalize well on new data that it has never seen. The major cause of the overfitting problem is the lack of training data. [21]



**Figure 32** Model Capacity and its Effect on Underfitting and Overfitting

When the capacity is low, both of the training error and test error are high. When the capacity increases, the training error decreases, the test error initially decreases and then starts to increase, leading to overfitting. When the generalization gap is big (Figure 33), changing model parameters or the model itself is the key to avoid underfitting, but what about overfitting when the model is doing well on the training dataset and can't generalize on test or validation data? Many techniques can be used; the most efficient one is **Data Augmentation** when we don't have enough data to train our model. (see Chapter 2)

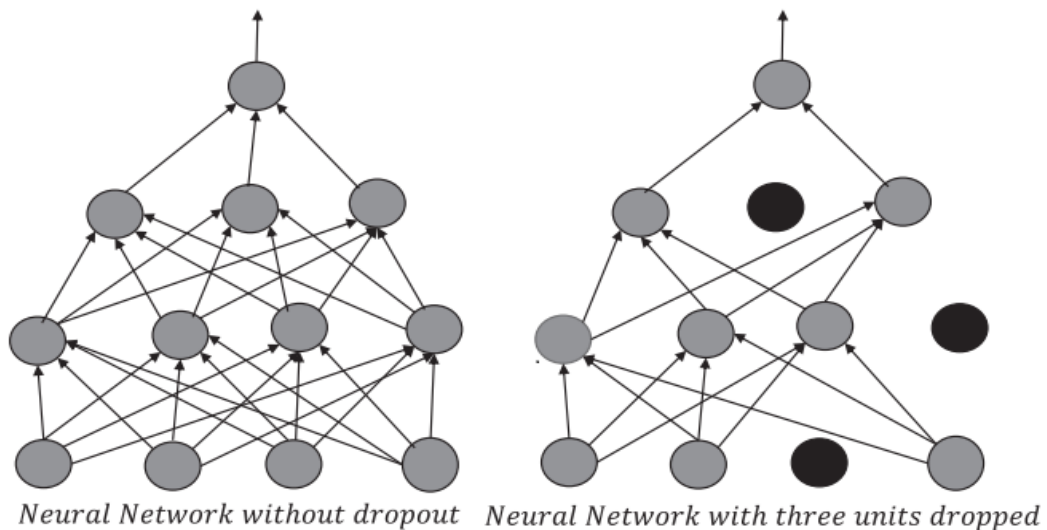




**Figure 33** Detection of Overfitting

### 1.3.2 Dropout

Dropout is an activity to regularize weights in the fully connected layers of convolutional neural network in order to avoid overfitting. [6] Some neuron with their connection are randomly dropped from the network during training set (Figure 34), the remaining neurons can learn important features all by themselves and not rely on cooperation from other neurons. [16] The high cooperation between neuron lead to overfitting since it does well on the training dataset, the random dropout technique show a great improvement on generalization



**Figure 34** Neural Network with three unit Dropped Randomly

## 1.4 Deep Learning Architecture

The last few years many Deep Learning architecture appear and evolved achieving top scores on many tasks including images classification, they can achieve high accuracy, [6] one of the most architecture called ResNet

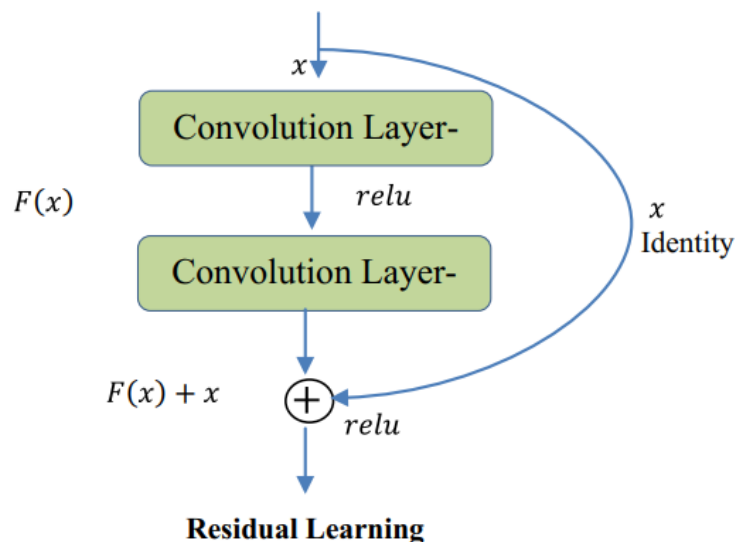
### 1.4.1 ResNet [25]

As the number of layers of deep networks increases, the accuracy improves and the accuracy saturates once the network has converged. However, if the depth is further increased, then the performance starts getting degraded rapidly. This degradation is caused by adding more layers to an already converged deep model which results in higher training error. Thus, there is a need for a strategy that obtains an optimal deep network for a given application. ResNet was proposed with a residual learning framework that lets new layers to fit a residual mapping. It is easier to push the residual to zero when a model has converged than to fit the mapping by a stack of nonlinear layers [6].

Given an underline mapping  $H(x)$  to be fit by a few stacked layers, where  $x$  is the input to these layers, the residual learning use the residual function

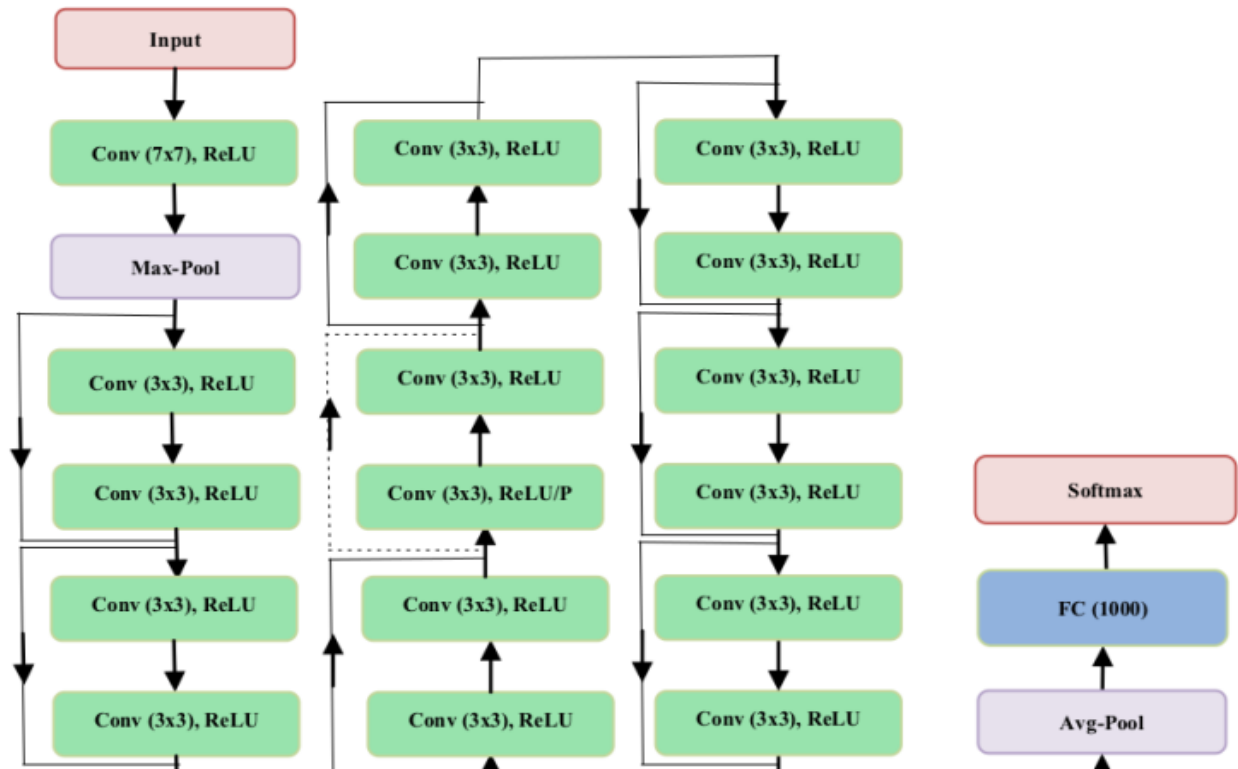
$$F(x) = H(x) - x \quad (15)$$

It is easier to optimize the residual mapping than to optimize the original, and it can be realized by a feedforward neural network with shortcut connection as shown in Fig.35 The shortcut link simply accomplishes identity mapping, and the output of. [6]



**Figure 35** ResNet Residual Learning Block

The architecture diagram of ResNet-34 is shown in (Figure 36)



*Figure 36* ResNet-34 Layers Architecture Diagram

### 1.4.2 AlexNet

The AlexNet CNN architecture [26] was developed by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton in 2012, AlexNet won the ImageNet ILSVRC (ImageNet Large-Scale Visual Recognition Challenge) competition in the same year. The model consists of five convolutional layers, three pooling layers, three fully connected layers, and a 1000-way Softmax classifier [6].

The original paper's primary result proved that the depth of the model has a huge impact on the model performance, a deep model was computationally expensive, but it's become workable due to the utilization of the Graphics Processing Units GPU during training. [15]

# Chapter 2

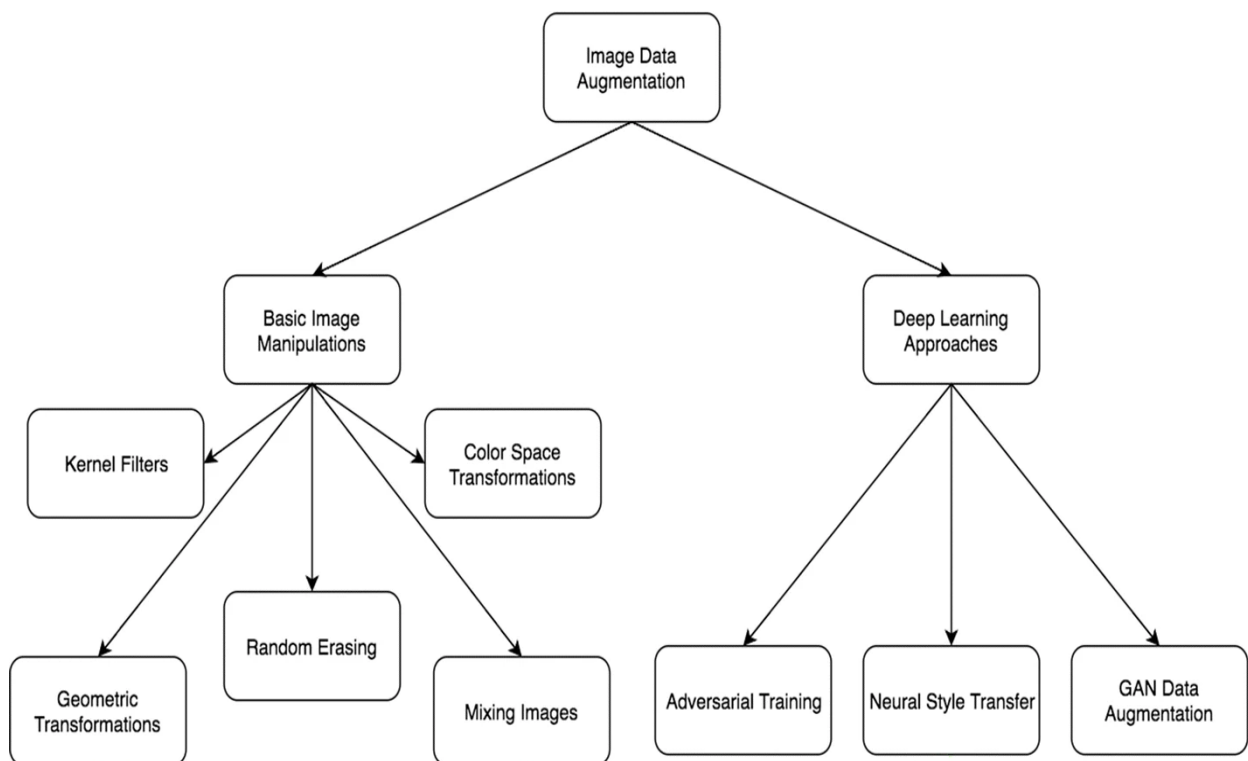
## Data Augmentation

### Introduction

Deep learning is the fastest-growing field these days in the machine learning (ML) field and between many DNN structures, the Convolutional Neural Networks (CNNs) are currently the main tool used for image classification and analysis purposes. Although great achievements and perspectives, deep neural networks have many pertinent challenges to tackle. One of them is the lack of sufficient amount of training data this considered as the most frequent problem in this field for example in the medical imaging domain is how to cope with a limited amount of data and small datasets, [27] especially in the case of supervised machine learning that requires labeled data and larger training examples for the model to train [28]. The more data we fit the more accuracy we get, another problem we may face called Overfitting, many techniques used in order to deal with these two problems (the lack of data and overfitting) called Data Augmentation [29]. In this Chapter, we focus on Data Augmentation techniques used and how can we solve the problems of overfitting and the lack of data using those techniques

## 2.1 Data Augmentation

Data augmentation is concerning the process of creating new data from the original data by manipulating the data we have we can generate new data. A recent study proved that the performance of ConvNets is logarithmically proportional to the number of training samples [30]. Conversely, without enough training samples, Convolutional Neural Networks face overfitting because of memorizing a detailed features of training data that cannot be generalized when the model predicts on new data those features will be useless. [31] For example, for images data augmentation increases the variety of images and generate more data by manipulating them in different ways such as resizing, flipping, rotating, random cropping, ...etc. [32], or by changing images color space and noise injection. This process increases the diversity of the data available for training models without the need of collecting new data, not only manipulating images can be used, many techniques and studies address this problem (Figure 37), translating the training images a few pixels in each direction can improve generalization also, even if the model has already been designed to be partially translation invariant by using the convolution and pooling techniques.



**Figure 37** Image Data Augmentation for Deep Learning

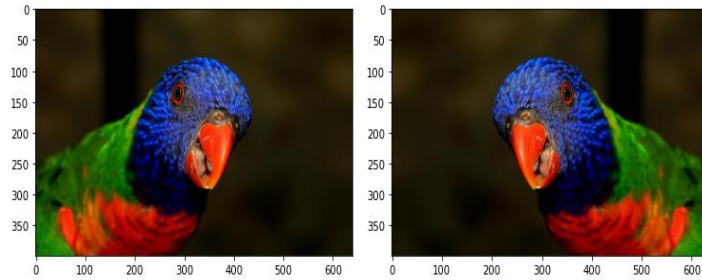
Here is some basic image augmentation technique that can be applied to generate more images data

### 2.1.1 Affine transformation (Basic image manipulation)

the most popular practice for data augmentation is the affine image transformation, [29] and color modification, as the affine transformation we define:

#### 2.1.1.1 Horizontal Flipping [33]

the amount of data will be doubled by applying a single operation of horizontal flipping on the dataset



**Figure 38** Image Horizontal Flipping (Mirror)

Mathematically this operation consists of flipping the 3 matrices R, G and B matrices values horizontally:

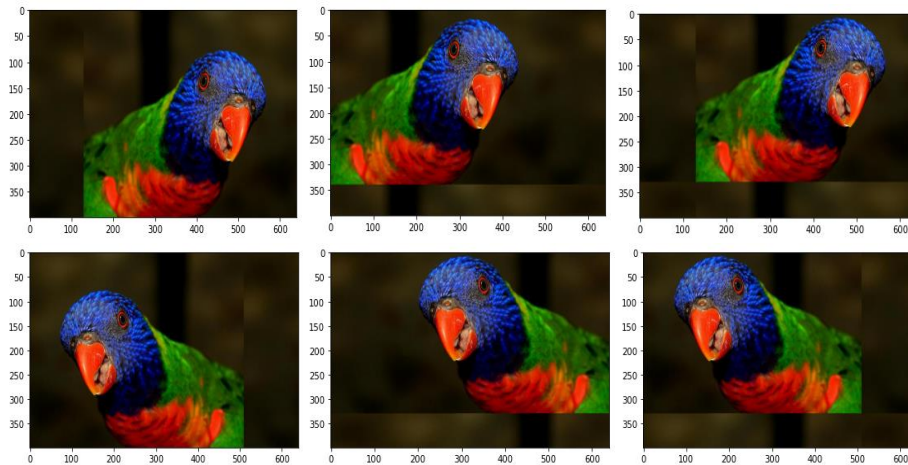
$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad M' = \begin{bmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \\ 9 & 8 & 7 \end{bmatrix} \quad (16)$$

A horizontal flip of a picture of a cat or a dog, a bird (figure 38) for example may make sense, because the photo could have been taken from the left or the right, a vertical flip of the images not make any sense and would probably not be efficient on improving models accuracy in most of cases, it may also cause wrong information detection this taken from the hand-written digit recognition models, for example, we pass an image with the label 6, if we flip that image vertically our input image will become 9, but our label is still 6, so we are passing wrong labels through our model, and this will affect the performance of our model.

#### 2.1.1.2 Horizontal and vertical shift

In shift-invariance CNN such preprocessed test images will make no difference in prediction, but this operation can generate more training samples (Figure 39), small displacement of object can make the model more efficient on learning features and allow it to generalize and predict correctly. This process can be done by shifting the 3 RGB matrices in the target direction here is an example of shifting matrix by one pixel to the right

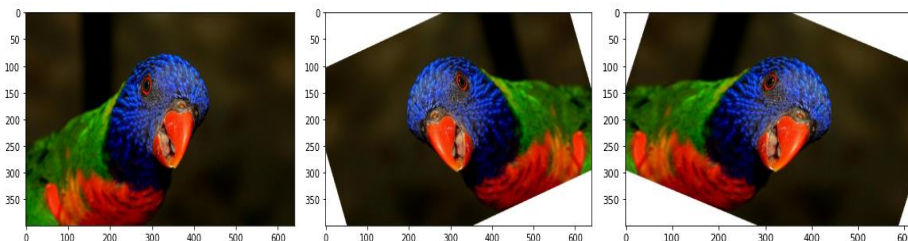
$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad M' = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 4 & 5 \\ 0 & 7 & 8 \end{bmatrix} \quad (17)$$



**Figure 39** Horizontal and Vertical Shift

#### 2.1.1.4 Rotation [34]

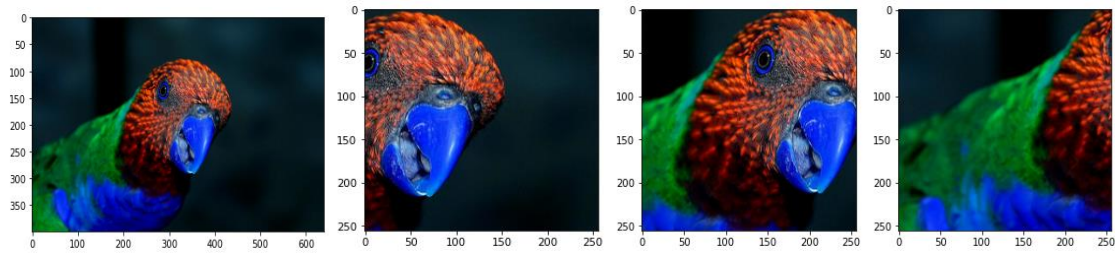
Before applying rotation, we should analyze every class and its possible rotations in order not to ruin image features. This operation can be done by flipping images with a specific degree, this can generate more samples and make our model more detectable to small changes in samples.



**Figure 40** Rotation Example

#### 2.1.1.5 Random cropping [35]

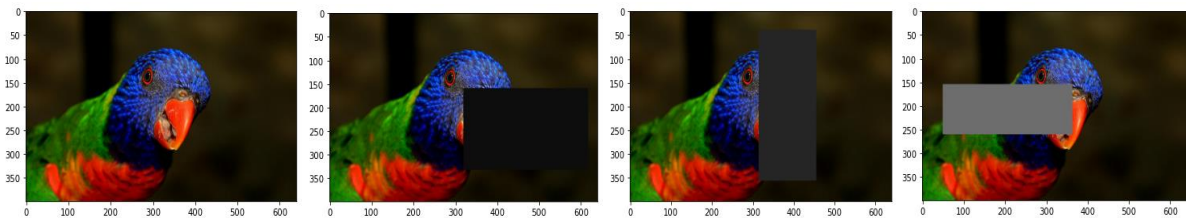
Random crop is a data augmentation technique based on taking a random subset from the original image. This technique allows the model to generalize better because the object of interest and features we want our model to learn and detect are not always wholly visible in the image or the same scale in our training data.



**Figure 41** Random Cropping

### 2.1.1.6 Random erasing: [36]

Occlusion is a critical influencing factor on the generalization ability of CNNs (Figure 42). It is desirable that invariance to various levels of occlusion is achieved. When some parts of an object are occluded, a strong classification model should be able to recognize its category from the overall object structure



**Figure 42** Random Erasing Example

Random erasing is considered as one of the most effective augmentation techniques, it shows great results

There are many other transformations that could be used to augment dataset like Kernel filter, mixing images [35] and other technique that manipulate images to get more data.

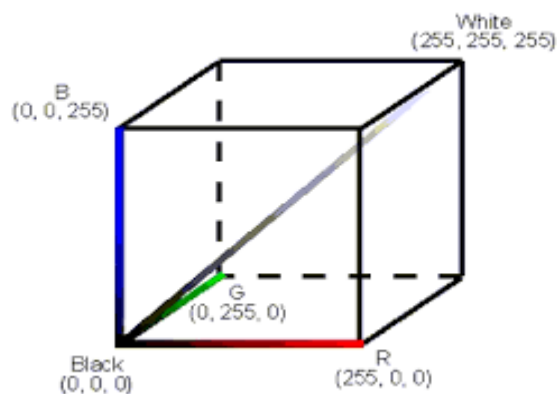


## 2.1.2 Color Space Transformations

Color images play an important role in everyday activities such as photography, television. It's involved in every aspect of our lives, it's a complicated phenomenon that has occupied the interest of scientist for hundreds of years, there are many color space that we can use for our treatment (OpenCV library contain more than 150 color space), changing color space of dataset could improve the performance of our models [37] to get better accuracy and avoid overfitting

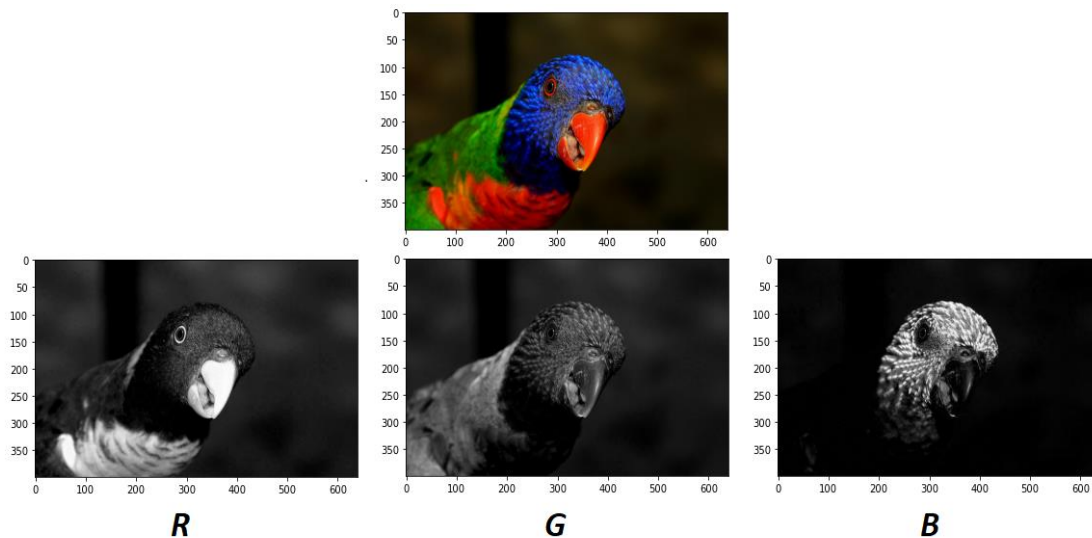
### 2.1.2.1 RGB Color Space [38]

It's the most used color space as a standard in training models cause it's the simplest color space, the RGB color schema encodes color as a combination of the three primary colors: red (R), green(G), and blue (B) with different proportions, each parameter (red, green and blue) defines the intensity of the color as an integer between 0 and 255(one byte). the color depth is another description of the range of intensity, the good thing on RGB is being supported in analog devices such as TV and digital device such as computer and cameras and in all browsers. This make RGB as the most useful color space, mixing the three primary colors value allow us to get all combination possible, we can represent it also as a three-dimensional coordinate plane where we place for R (red), G (green), and B (blue) on each axis, (Figure 43) This coordinate plane yields a cube represent the RGB color space:



**Figure 43** Representation of the RGB Color Space as a Three-Dimensional Unit Cube

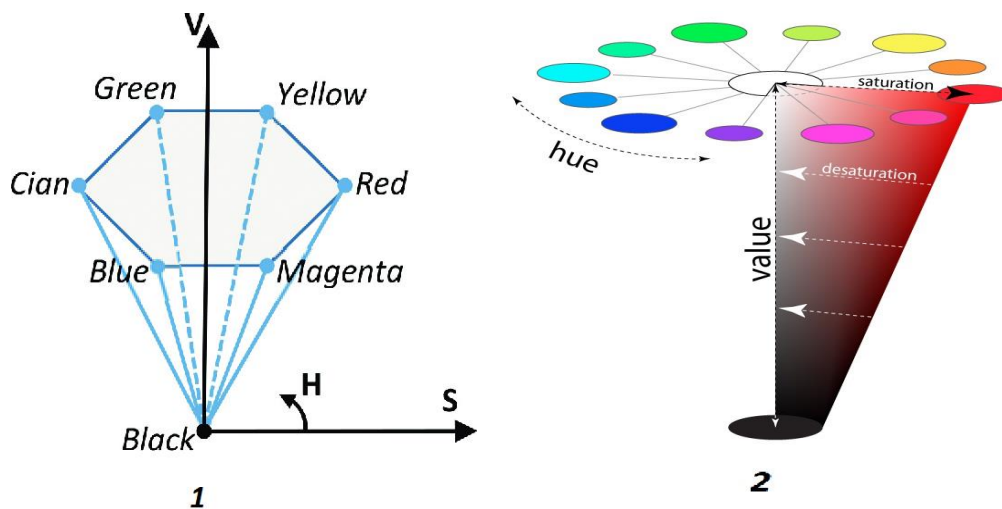
There is another representation where RGB value are normalized to the interval  $[0, 1]$  instead of  $[0, 255]$  so that the resulting color space forms a unit cube. The point  $S = (0,0,0)$  corresponds to the color black,  $W = (1, 1, 1)$  corresponds to the color white. [39] All points are between S and W are shades of gray created from equal color components  $R = G = B$  (Figure 44) show a color test images and its corresponding RGB color components



**Figure 44** A Color Image and its Corresponding RGB Channels

### 2.1.2.2 HSV Color Space [40]

which mean Hue Saturation Value it's based on cylindrical coordinate representation of point in an RGB color model (Figure 45 (1)) where the vertical axis  $V$  represents the brightness value (The chromatic notion of intensity), the  $H$  which is the angle,  $S$  which is the radius (The amount of white color mixed with a hue). It's a conversion from the RGB color model into this cylinder, the lower the value the darker the outcome color and the higher the value the more the outcome color resemble the color itself (Figure 45 (2)), then we change these three parameters in order to create different combination of colors



**Figure 45** HSV Color Space in Cylindrical Coordinates

### 2.1.2.2.1 Convert RGB to HSV

The RGB values are divided by 255 to change the range from 0...255 to 0...1

$$R' = \frac{R}{255} \quad , \quad G' = \frac{G}{255} \quad , \quad B' = \frac{B}{255} \quad (18)$$

$C_{\max}$  and  $C_{\min}$  are the max and the min of  $(R, G, B)$  then obtain the value of  $\Delta$

$$\begin{aligned} C_{\max} &= \max(R', G', B') \\ C_{\min} &= \min(R', G', B') \\ \Delta &= C_{\max} - C_{\min} \end{aligned} \quad (19)$$

When the RGB value have the same value ( $R = G = B$ ) then we are dealing with achromatic (gray) pixel the value of  $\Delta$  will be obviously 0

Hue calculation:

$$H = \begin{cases} 0^\circ \quad , \quad \Delta = 0 \\ 60^\circ \times \left( \frac{G' - B'}{\Delta} \bmod 6 \right) \quad , \quad C_{\max} = R' \\ 60^\circ \times \left( \frac{B' - R'}{\Delta} + 2 \right) \quad , \quad C_{\max} = G' \\ 60^\circ \times \left( \frac{R' - G'}{\Delta} + 4 \right) \quad , \quad C_{\max} = B' \end{cases} \quad (20)$$

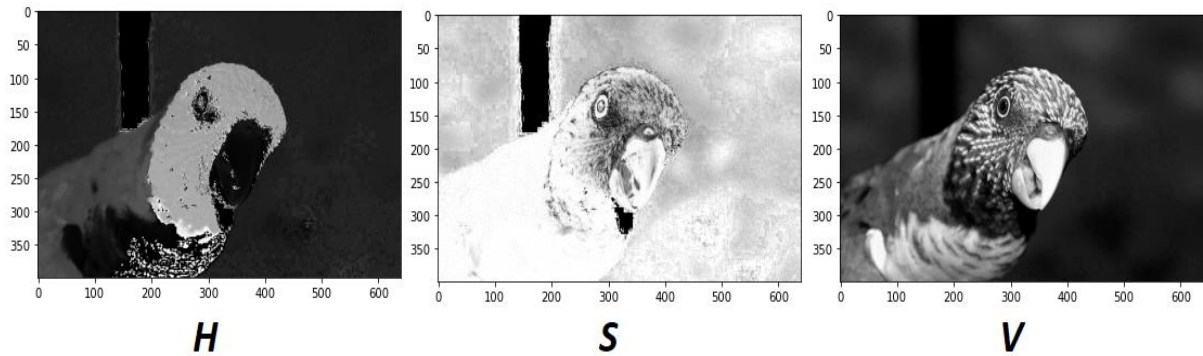
Saturation calculation

$$S = \begin{cases} 0 \quad , \quad C_{\max} = 0 \\ \frac{\Delta}{C_{\max}} \quad , \quad C_{\max} \neq 0 \end{cases} \quad (21)$$

Value calculation

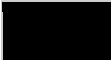



$$V = C_{\max} \quad (22)$$

So, the RGB space unit cube is mapped to a cylinder. All RGB color coordinate may be represented on this cylinder in HSV space. The mapping from the RGB to HSV space in nonlinear can be noted by examining how the black point stretches [37] Figure 46 shows the individual HSV components of the test image as grayscale images



**Figure 46** HSV Color Components

The table below show the variation between the RGB and HSV color space

Color	Color name	Hex	(R,G,B)	(H,S,V)
	Black	#000000	(0,0,0)	(0°,0%,0%)
	White	#FFFFFF	(255,255,255)	(0°,0%,100%)
	Red	#FF0000	(255,0,0)	(0°,100%,100%)
	Lime	#00FF00	(0,255,0)	(120°,100%,100%)
	Blue	#0000FF	(0,0,255)	(240°,100%,100%)
	Yellow	#FFFF00	(255,255,0)	(60°,100%,100%)
	Cyan	#00FFFF	(0,255,255)	(180°,100%,100%)
	Magenta	#FF00FF	(255,0,255)	(300°,100%,100%)

**Table 1** RGB/HSV Values

### 2.1.2.3 HSL Color Space [41]

HSL is short of Hue, Saturation and Luminance (or brightness) similar to the HSV, the hue is what we normally call color is based on the position around the wheel, Saturation define how pure a color is and saturation is based on the distance from the center of this wheel, if we want to desaturate we move inward, and if we want to saturate we move outward. At full desaturation our color becomes achromatic, or colorless finally the Lightness and its represented as the value of the L axis, HSV take the form of a double (Figure 47), the black color in the bottom tip and white on the top.

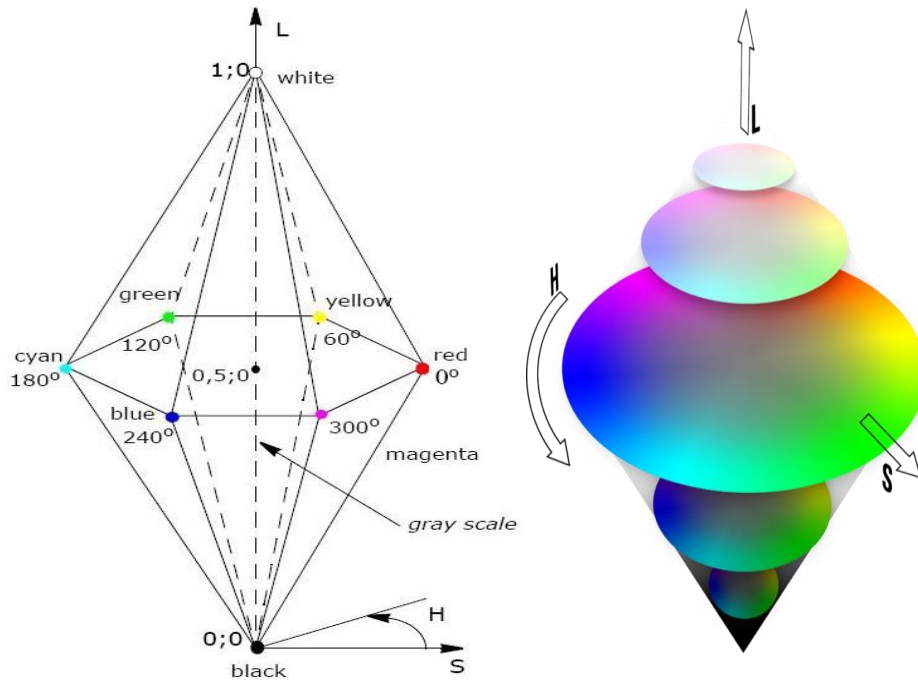


Figure 47 HSL Color Space

#### 2.1.2.3.1 Convert RGB to HSL

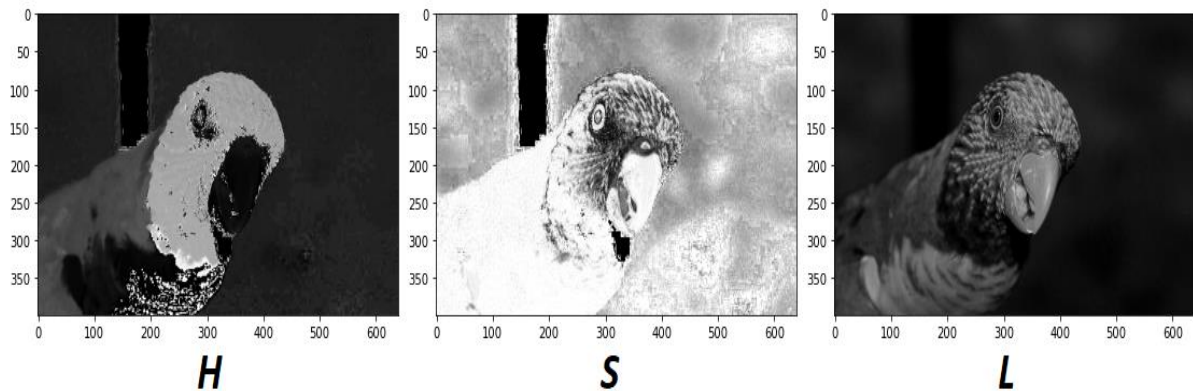
In the HSL model, the hue is computed in the same way as in the HSV model

$$H_{hsl} = H_{hsv} \quad (23)$$

The other values Saturation and Lightness, are computed as follows





$$L = \frac{C_{\max} + C_{\min}}{2} \quad (24)$$

$$S = \begin{cases} 0, & \Delta = 0 \\ \frac{\Delta}{1 - |2L - 1|}, & \Delta \neq 0 \end{cases} \quad (25)$$



**Figure 48** HSL Color Component

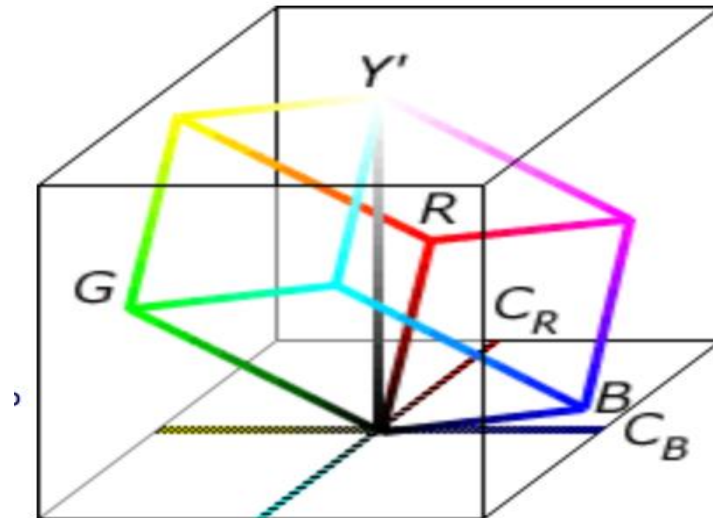
The table below show the variation between the RGB and HSL color space

Color	Color name	Hex	(R,G,B)	(H,S,L)
	Black	#000000	(0,0,0)	(0°,0%,0%)
	White	#FFFFFF	(255,255,255)	(0°,0%,100%)
	Red	#FF0000	(255,0,0)	(0°,100%,50%)
	Lime	#00FF00	(0,255,0)	(120°,100%,50%)
	Blue	#0000FF	(0,0,255)	(240°,100%,50%)
	Yellow	#FFFF00	(255,255,0)	(60°,100%,50%)
	Cyan	#00FFFF	(0,255,255)	(180°,100%,50%)
	Magenta	#FF00FF	(255,0,255)	(300°,100%,50%)

**Table 2** RGB/HSL Values

#### 2.1.2.4 TV Color Space YUV

YUV also is known as YCbCr in the word of computer color model is adopted by European TV systems Phase Alternating Line (PAL Standard) which is a color encoding system for analog television [42] and NTSC (National Television System Committee). The Y component determines the color brightness (referred to as luminance), while the U and V components determine the color itself (the chroma). Y is the luminance value it's the overall brightness of the pixel, Y ranges from 0 to 1 (or 0 to 255 in digital format), this effectively a grayscale value (Figure 50). while U or (CB) and V or (CR) range from -0.5 to 0.5 (or -128 to 127 in signed digital form) [43]. Coordinate representation of YUV color space has been shown in Figure 49



**Figure 49** YUV Color Space

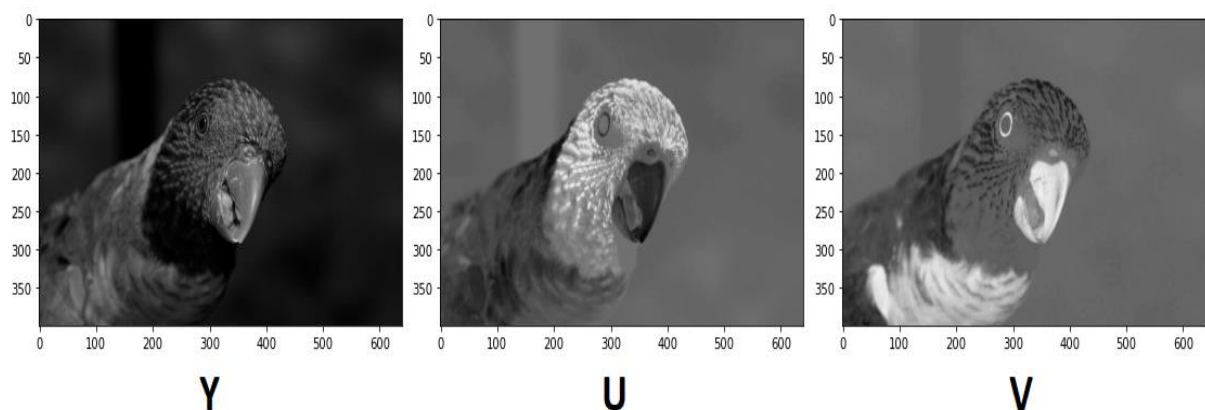
#### 2.1.2.4.1 Convert RGB to YUV [43]

The first equation gives conversion from RGB to YUV

$$\begin{aligned} Y &= 0.299 \times R + 0.587 \times G + 0.144 \times B \\ U &= -0.14713 \times R - 0.22472 \times G + 0.436 \times B \\ V &= 0.615 \times R - 0.51498 \times G + 0.10001 \times B \end{aligned} \quad (26)$$

The second equation gives the inverse function that convert from YUV to RGB

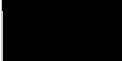







$$\begin{aligned} R &= 0.7492 \times Y - 0.50901 \times U + 1.1398 \times V \\ G &= 1.0836 \times Y - 0.22472 \times U - 0.5876 \times V \\ B &= 0.97086 \times Y + 1.9729 \times U - 0.000015 \times V \end{aligned} \quad (27)$$



**Figure 50** YUV Color Component



The table below show the variation between the RGB and YUV color space

Color	Color name	Hex	(R,G,B)	(Y,U,V)
	Black	#000000	(0,0,0)	(16,128,128)
	White	#FFFFFF	(255,255,255)	(235,128,128)
	Red	#FF0000	(255,0,0)	(82,90,240)
	Lime	#00FF00	(0,255,0)	(145,54 ,34)
	Blue	#0000FF	(0,0,255)	(41,240 ,110)
	Yellow	#FFFF00	(255,255,0)	(210,16 ,146)
	Cyan	#00FFFF	(0,255,255)	(170,166,16)
	Magenta	#FF00FF	(255,0,255)	(107,102,221)

**Table 3** RGB/YUV Values

### 2.1.2.5 XYZ and LUV Color Space [44]

RGB color space is linear space which is always used in computation of color optical flow. For the nonlinear character of the human vision, RGB color space appears obviously non-uniform effect. When an object is changed from one location to another in RGB color space, if only the original image location is changed and the distance remain unchanged, the difference of variation can be visually felt. In order to obtain the real uniform color space from human vision, CIE (the International Commission on Illumination )defines the LUV color space, which is derived from XYZ color space. The XYZ color space can be obtained from RGB color space by following linear transformation:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.430 & 0.342 & 0.178 \\ 0.222 & 0.707 & 0.071 \\ 0.020 & 0.130 & 0.939 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (28)$$

Further, two middle variables should be defined in the transition from CEI XYZ color space to CEI LUV color space

$$u' = \frac{4X}{X + 15Y + Z} \quad (28)$$



$$v' = \frac{9X}{X + 15Y + Z} \quad (29)$$

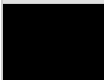







If the reference luminance of white light is considered, then  $X, Y, Z$  can be expressed as  $X_n, Y_n, Z_n$ , and then  $U$  and  $V$  computed as

$$U = 13L(u' - u'_n), V = 13L(v' - v'_n) \quad (30)$$

Where,  $L$  is as follow:

$$L = 166 \left( \frac{Y}{Y_n} \right)^{1/3} - 16, \frac{Y}{Y_n} < 0.08865 \quad (31)$$

to convert RGB to CIELUV, first we convert RGB to CIEXYZ and then we convert CIEXYZ to CIELUV

Color	Color name	Hex	(R,G,B)	(X,Y,Z)	(L,U,V)
	Black	#000000	(0,0,0)	(0, 0, 0)	(0, 0, 0)
	White	#FFFFFF	(255,255,255)	(95.05,100,108.89)	(100,0.00089, -0.017)
	Red	#FF0000	(255,0,0)	(41.24,21.26,1.93)	(53.23,175.053,37.75)
	Lime	#00FF00	(0,255,0)	(35.76,71.52,11.92)	(87.73, -83.07, 107.40)
	Blue	#0000FF	(0,0,255)	(18.05, 7.22 ,95.05)	(32.30, -9.39, -130.35)
	Yellow	#FFFF00	(255,255,0)	(77, 92.78, 13.85)	(97.13, 7.70, 106.78)
	Cyan	#00FFFF	(0,255,255)	(53.81,78.74,106.97)	(91.11, -70.47, -15.21)
	Magenta	#FF00FF	(255,0,255)	(59.29,28.48,96.98)	(60.31,84.07, -108.71)

**Table 4** RGB/XYZ/LUV Values

### 2.1.2 Deep Learning Approaches

The methods discussed above are applied to images in the input space, Traditional augmentation approaches are firmly limited, especially in tasks where the images follow strict standards, as in the case of medical datasets. The most used technique is called GAN

#### 2.1.2.1 GAN-Based Data Augmentation

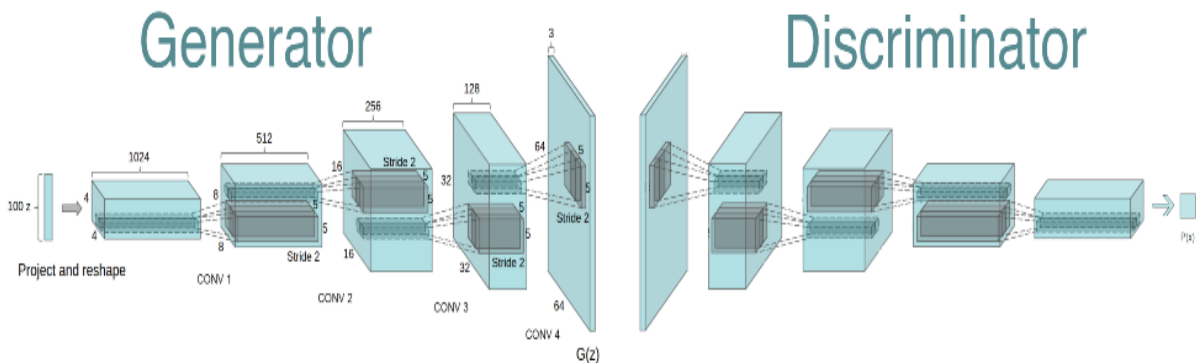
GAN refer to Generative Adversarial Network, the idea of GANs was conceived in 2014 by Ian Goodfellow, [45] GANs can learn about your data and learn to synthesize, generate never-before-seen data to augment the dataset it's considered as an approach of unsupervised learning and even semi-supervised learning (when the some of the data are labeled and rest are not), GANs allow us to generate unlabeled data from our labeled data. A basic GAN consist of two model that play against each other a generator and a discriminator, we can think of the generator as counterfeiter who tries to replicate the input data in order to produce fake data, on the other hand the discriminator is like a cop who need to be able to distinguishes real data which is form the dataset and the fake data which is produced by the generator [46] the loss of the discriminator is used as an objective function for the generator in the next time round. The idea is awesome creating a fake dataset from the original one will completely solve the problem of the lack of data, NVIDIA has create their own model based on GANs architecture which allow them to produce fake human faces as we can see in Figure 51. [47] It's hard to believe that those faces are fake and had been generated from a GAN model.



**Figure 51** Fake Faces Generated by StyleGAN

Building a GAN model consist of different steps, first we need to define the architecture of the GAN a neural network for example then we need to train the discriminator model to distinguish between real and fake data, the next step is to train the generator, we need to modify parameters of the generative model to maximize the loss of the discriminator then

we repeat the training of the generator and the discriminator over  $n$  epoch after every iteration the generator will get better at fooling the discriminator, finally the discriminator will not be able to tell the real images of the dataset from the generated ones by the generator once the training is complete we synthesize data from the generator and this can be used to augment our dataset or use it as is. There are many types of GANs let's take the most useful one which is the Deep Convolutional GAN (DCGAN) this type used CNNs in generator and discriminator



**Figure 52** GAN Generator and Discriminator

Let's go back to the example of generating images of faces like we see in NVIDIA's IA the discriminator will take input face image either from the generator or the dataset and output real if it believe the image is of an actual face or fake if it believe the face is not of a human, the generator will be given some data as an input and will have to come up with a face this is done through a D-Convolutional Neural Net.

### 2.1.2.2 Neural Style Transfer [48]

Neural style transfer is the flashiest demonstration of technique of deep learning capabilities. The general idea behind the Neural Style Transfer is to use CNN to transfer the style of a given painting to any image (Figure 53). Neural Style Transfer is probably best known for its artistic applications, but it also used on Data Augmentation, it's done by manipulating the sequential representation across a CNN such that the style of one image can be transferred to another while preserving the original one.



**Figure 53** Example of Neural Style Transfer

**Smart data augmentation [45]**

Smart augmentation is the latest technique, both neural style transfer and smart augmentation are new research field, the smart augmentation consist of creating a neural that learn how to generate augmented data during the training process in order to get more data. This technique allows us to learn augmentation that minimize the error of network. The goal of Smart Augmentation is to learn the best augmentation strategy for a given class of input data. It does this by learning to merge two or more samples in one class. This merged sample is then used to train a target network. The loss of the target network is used to inform the augmenter at the same time. This method might be used to generate more data that for the training process. This process often includes letting the network come up with unusual and unexpected highly performant augmentation strategies.

## CHAPTER 3

# EXPERIMENTATION AND RESULTS

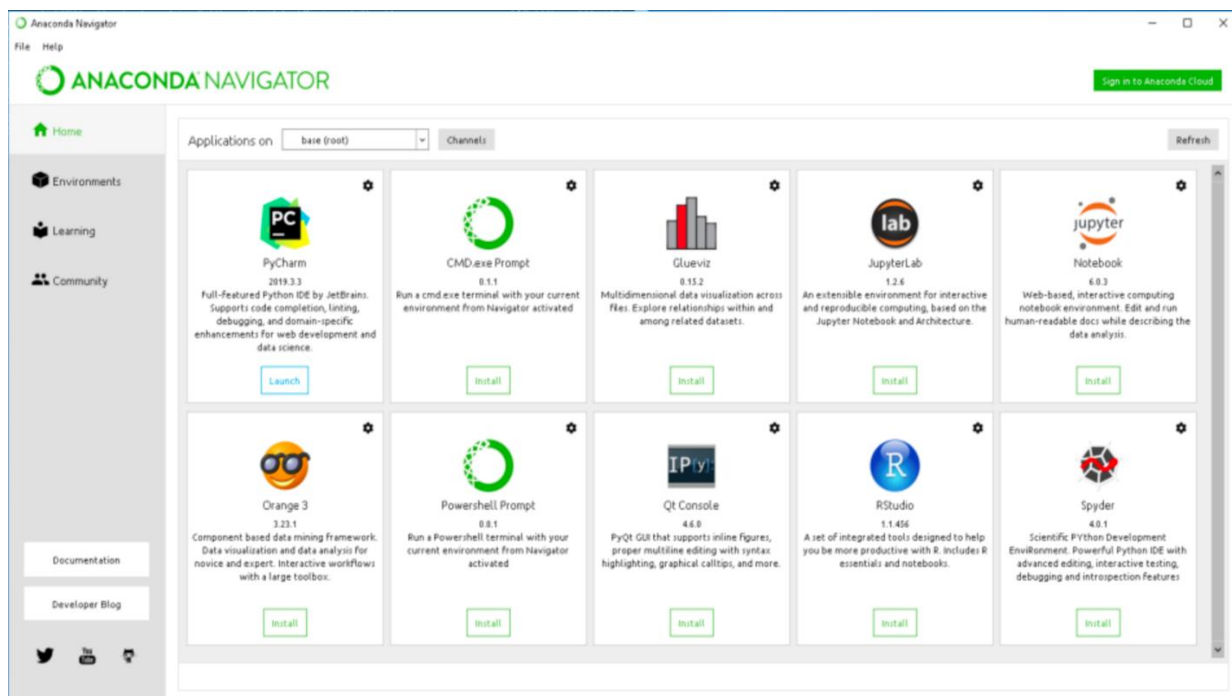
The main focus throughout this thesis has shown the impact of the quantity of the data on the performance of deep learning model in image classification and the effect of the data augmentation when facing the lack of dataset or overfitting problems.

In this chapter we will discuss the development tools used such as anaconda and Jupyter notebook and the main libraries as Keras and Tensorflow using python as a programming language then the Kaggle dataset used. The second part contain the implementation and discussion of result we obtain

### 3.1 Development Tools

#### 3.1.1 Anaconda [50]

Anaconda is a free and open source distribution of python and R programming language more than 20 million person use this technology, platform to solve the toughest problem [www.anaconda.com] specially for scientific computing such as machine learning application, data science and data processing, Anaconda aims to simplify deployment and package management, package version are managed by the package management conda (Figure 54), Anaconda distributions are suitable to Windows, Linux and even MacOS.



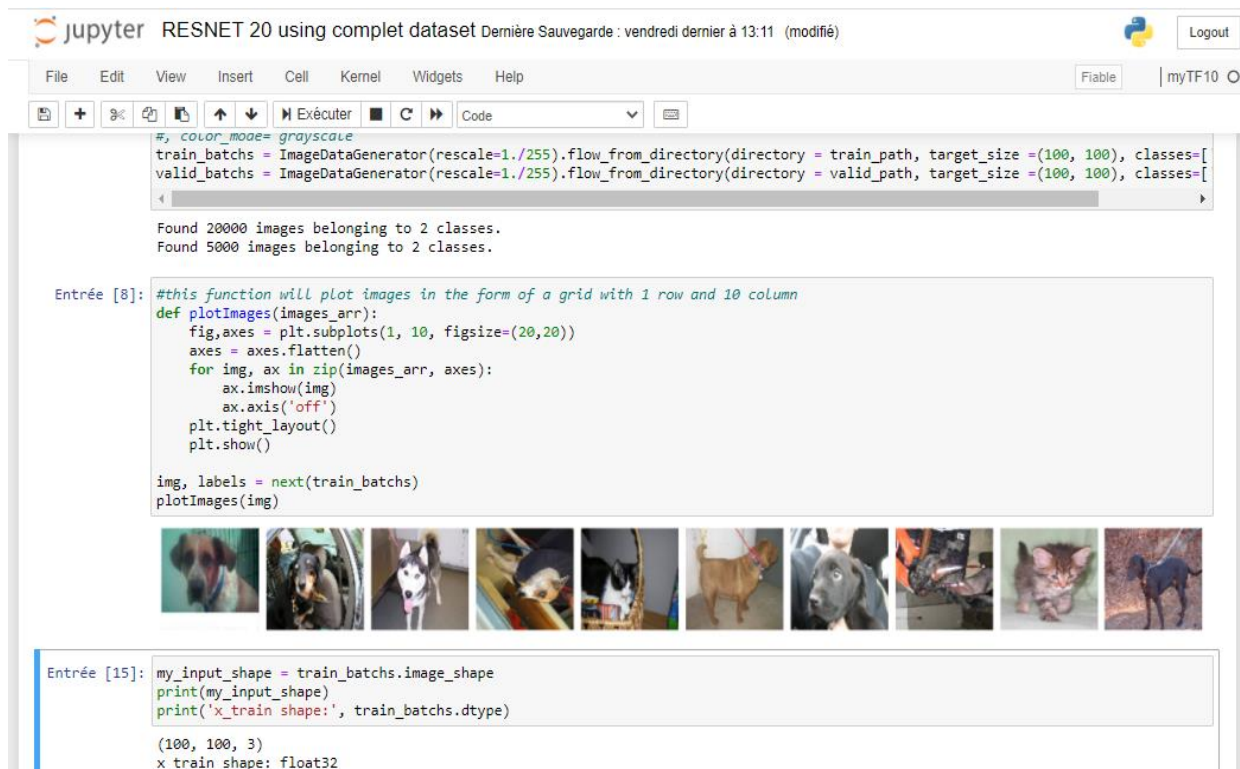
*Figure 54* Anaconda Navigator

### 3.1.2 Jupyter Notebook [51]

Jupyter notebook is an open source software and a service for interactive computing across nearly 40 programming language providing a web-based application suitable for capturing the whole computation process developing, executing code, documentation and communicating the results. The jupyter notebook combine two components

**A web application** which is a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output.

**Notebook documents:** a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects.



**Figure 55** Jupyter Notebook Interface

### 3.1.3 Python [52]

Python is an interpreted, high-level and general-purpose programming language created by Guido van Rossum and first released in 1991, its language constructs and object-oriented approach helps to write logical and clear code and there is no compilation step. Many machine learning applications written in python, it's become the most used language in the field of machine leaning, data analysis and data processing.



## 3.2 Libraries

### 3.2.1 Keras [53]

Keras is a deep-learning framework for Python that provides a convenient way to define and train almost any kind of deep-learning model. Keras was initially developed for researchers, with the aim of enabling fast experimentation. Keras allow the same code to run on CPU and GPU, it has also a user-friendly API that make deep learning programming easier, it has built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both. It supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, and so on. This means Keras is appropriate for building essentially any deep-learning model, from a generative adversarial network to a neural Turing machine.

### 3.2.2 Tensorflow [54]

Tensorflow was developed by google on the google brain project and it's become free and open source in 2015, this software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning and deep learning applications such as neural networks it is used for both research and production at Google.

### 3.2.3 Scikit-image

It's an open source library for image processing for python programming language [55], scikit-image library contains algorithm of geometric transformation, segmentation, and color space manipulation, analysis and filtering and more. Scikit-image was designed to interoperate with the python numerical and scientific libraries SciPy and NumPy.

### 3.2.4 Pillow [56]

Python Imaging Library abbreviated as PIL called Pillow in new version in a free open-source library for python programming language that used for opening manipulating and saving images, Pillow allow us to applicate many geometric transformations on images that we need in this thesis to generate more images data from the existing one.

### 3.2.5 Matplotlib [57]

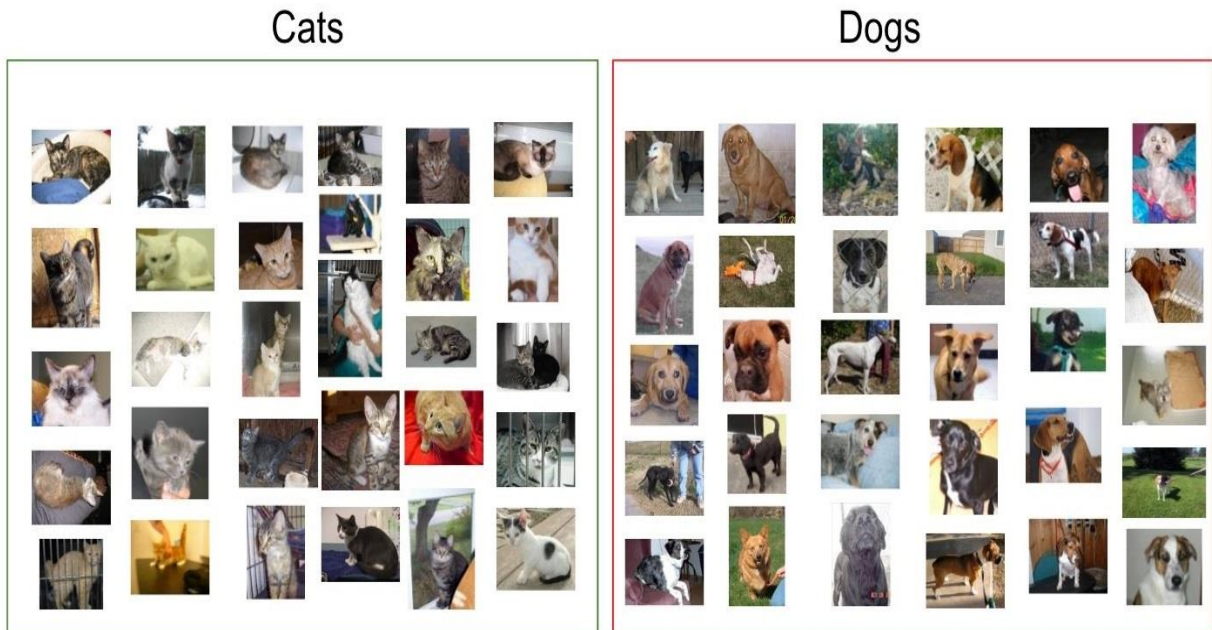
Matplotlib was originally written by John D. Hunter, since then it has an active development community, it's the most used plotting library for the python programmers. Matplotlib provides an object-oriented API for embedding plots into application using general-purpose GUI toolkits like QT and TKinter.

## 3.3 Kaggle Dataset

A subsidiary of Google LLC, is an online community of data scientists and machine learning practitioners. Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.

Kaggle got its start in 2010 by offering machine learning competitions and now also offers a public data platform, a cloud-based workbench for data science, and Artificial Intelligence education.

The chosen dataset for our study is one of the most available and used dataset for both beginners and even expert machine learning developers, we are clearly talk about dog and cat classification, I chose this dataset because it contains only 2 classes and it so easy to observe and evaluate those 2 class. Kaggle dataset provide us a complete dataset of 25000 labeled images divided on 2 class that mean 12500 image of dogs and 12500 images of cats for the test data it contains 12500 no labeled images [58]



**Figure 56** Sample of Cats and Dogs from Kaggle Dataset

Now we have 2 folders the first one contains cats' images and the other one contains dogs' images, as pretreatment is to add all this images to a NumPy Array of size 25000 element of size (100, 100, 3), 80% of this dataset will be used in training set and 20% used for the validation set.

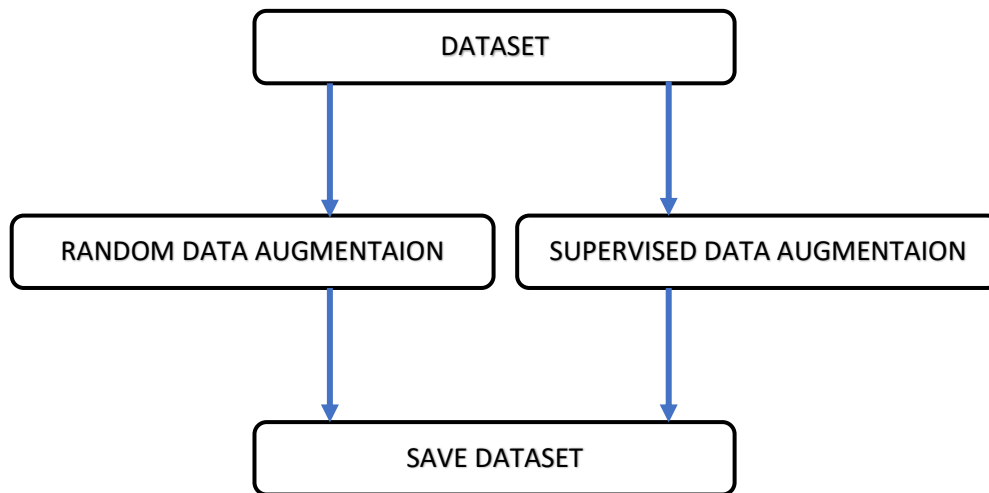
After fitting the complete dataset, we extracted 8% of dataset that means 2000 images and for the validation set we used 600 images.

Third we augment this 8% of data to become 16000 images that mean we applied 8 random augmentation for each image, then we did the same using supervised data augmentation, supervised means that we define the operation used for the data augmentation, the operation applied was mirror then 2 rotation (+20, -20) for both images the original and the flipped one and 2 horizontal shifting



**Figure 57** Data Augmentation Applied for Each Image





**Figure 58** The Process of Dataset Augmentation

### 3.4 implementation

For the implementation, ResNet-20 model was used on our different amount of data, Tensorflow and keras was used as a base for this project. We used the ImageDataGenerator class from keras library in order to apply the random geometric transformations on the Kaggle dataset to generate more data (augment our training dataset from 2000 image to 16000 image), then we did the same thing to the same dataset of 2000 this time instead of Keras we used Pillow library to apply a supervised data augmentation (we control the type of augmentation this time instead of applying a random operations) to generate the same number of samples 16000 divided on two class, keras library was also used to load and save training models the record of the history of the training to be able to draw chart line or confusion matrix using matplotlib library to compare result and the performance of each model and make prediction.

### 3.5 Experimentation

After we prepared four different dataset, the first one is the complete Kaggle dataset, the second one which contain 8% from the first dataset that means 2000 training samples, the third dataset is the second one augmented randomly from 2000 to 16000 training samples, and finally the fourth dataset generated by applying a supervised data augmentation to the second dataset it contain the same number of samples as the previous one.

We have trained the models into different architecture ResNet-20 and AlexNet, we set the number of epochs to 50 because of the limited computation resources. The low number of epochs won't impact the results we got since the purpose of this study is only to understand the impact of the data size on the performance of the CNN not to maximize the performance of the models used.

The models were trained using GPU NVIDIA GeForce 920MX with 4GB RAM with Keras backend. ResNet-20 took around 10 hours when using the complete dataset, and 6 to 8 hours when using supervised and random data augmentation

The AlexNet model was faster than the ResNet-20 it takes around 6 to 7 hours to train the model using the complete Kaggle dataset and around 4 hours to train the augmented dataset.

We store the weights of the CNN model for every epoch while training and record the accuracy and loss for each epoch to be able to evaluate and compare the results obtained.

### Discussion of the Results

From the experiment that we have conducted the results shown in table 5 was obtained, we observed that there are some changes in the model accuracy the best accuracy was obtained in the smallest dataset with 99.95% using ResNet-20 and 100% using AlexNet, otherwise the validation accuracy does not exceed 75.00% using ResNet-20 and reach 82.6% using AlexNet with the complete dataset , it's so clear that using the 8% of the dataset produce an overfitting because of the lack of training data.

		Data set			
		Complete dataset	8% of the dataset	Random data augmentation	Supervised data augmentation
ResNet-20	Accuracy	96.62%	<b>99.95%</b>	96.53%	97.32%
	Validation accuracy	<b>75.00%</b>	64.83%	70.67%	73.17%
AlexNet	Accuracy	98.11%	<b>100%</b>	97.72%	98.84%
	Validation accuracy	<b>82.60%</b>	72%	75%	74.83%

**Table 5** Comparison of Results for Different Dataset

The training accuracy can reach the highest value on training that refer to the model itself the model will overfit the data some point when using a limited amount of data as we can see in the second dataset, on the other hand the validation accuracy is higher when we use a large amount of data, Data Augmentation may raise the validation accuracy as we can in the previous table it raise the validation accuracy from 64.83% to 73.17% using ResNet-20, and from 72% to 75% using AlexNet, even small percentage may make difference.

The model performance of the model can be further understood by looking at the per class recognition accuracy of the CNN table 6 shows the results obtained by testing our models on 1000 images divided on two class

		Data set			
		Complete dataset	8% of the dataset	Random data augmentation	Supervised data augmentation
<b>ResNet-20</b>	<b>Cat</b>	73.00%	67.80%	65.20%	<b>78.00%</b>
	<b>Dog</b>	<b>74.60%</b>	62.20%	74.00%	63.60%
<b>AlexNet</b>	<b>Cat</b>	<b>90.20%</b>	75.40%	62.80%	82.80%
	<b>Dog</b>	77.00%	76.00%	<b>88.00%</b>	72.40%

**Table 6** Per Class Accuracy of ResNet-20 and AlexNet

The highest accuracy changed from a dataset to another, the class cat in ResNet-20 highest accuracy 78% was obtained using the Supervised Data Augmentation, the class dog highest accuracy reaches 74.6% using the complete dataset. Meanwhile using AlexNet gives different results 90.20% for cat class using the complete dataset as a highest accuracy and 88% as the highest accuracy for the second class obtained when using the Random Data Augmentation.

The table 7 and Table 8 shows the graph obtained using ResNet-20 and AlexNet respectively we can clearly see that the model has the same behaviors all the training loss is going down while the test loss still going up and that's an indicator of overfitting.

### CHAPTER 3 EXPERIMENTATION AND RESULTS

Table 7 ResNet-20 Results

	accuracy	Loss	Validation accuracy	Validation Loss
Complete dataset				
8% of the dataset				
Random DA				
Supervised DA				

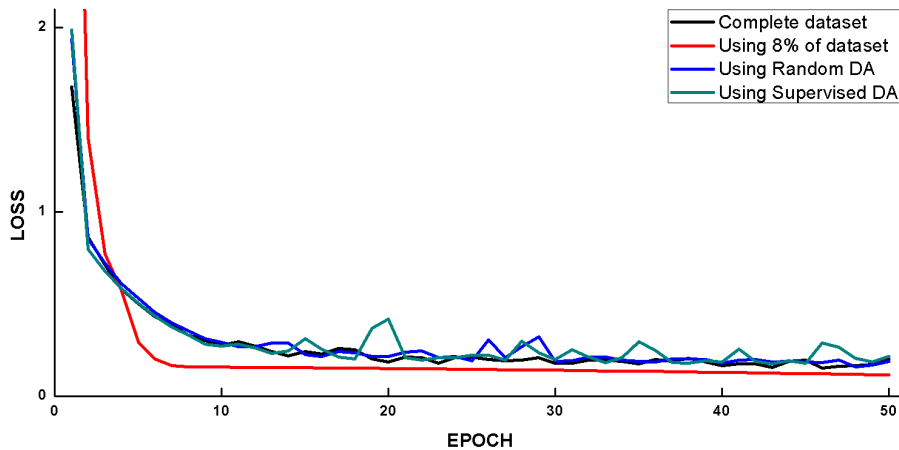
### CHAPTER 3 EXPERIMENTATION AND RESULTS

**Table 8** AlexNet Results

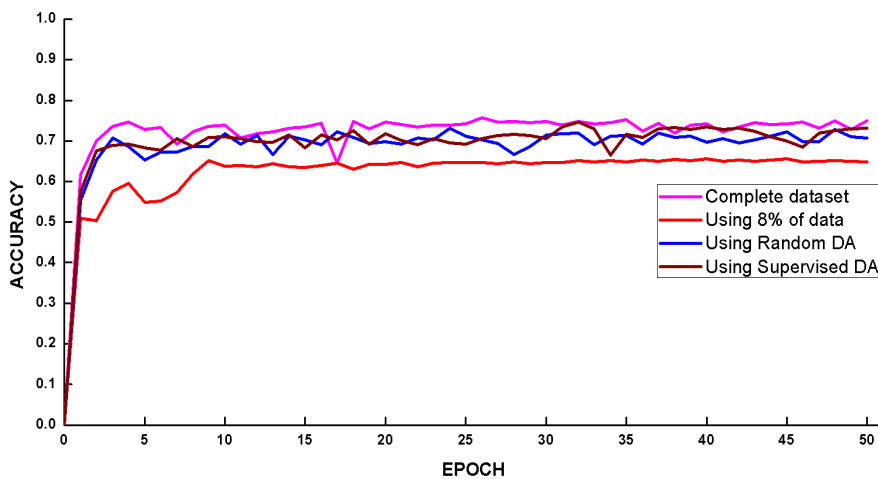
	<b>accuracy</b>	<b>Loss</b>	<b>Validation accuracy</b>	<b>Validation Loss</b>
<b>Complete dataset</b>				
<b>8% of the dataset</b>				
<b>Random DA</b>				
<b>Supervised DA</b>				

### Comparing the Results Using ResNet-20

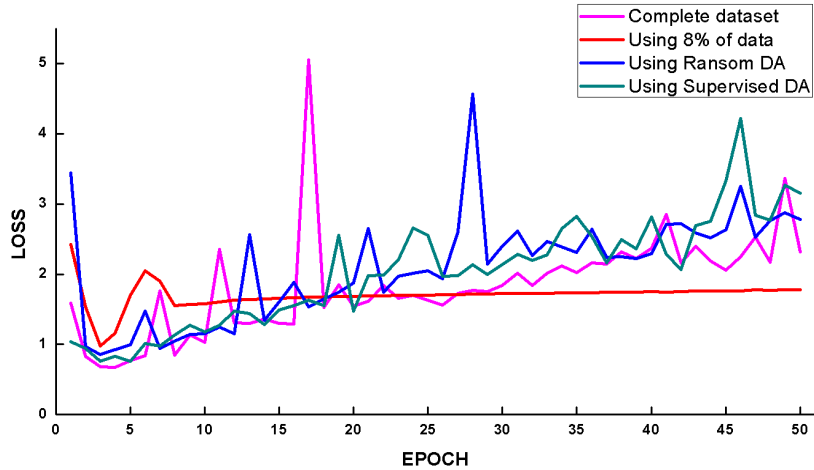
Looking at the plot obtained using ResNet-20 on different dataset, we can see that the model training loss have the same behavior (Figure 59), the model trained on the complete dataset and the augmented one are less overfit (Figure 60) (Figure 61) comparing to the plot when we use only 8% of the dataset, random data augmentation and supervised data augmentation shows a good concurrence, the supervised data augmentation almost reach the same value as the curve when using the complete dataset.



*Figure 59* ResNet-20 Training Loss



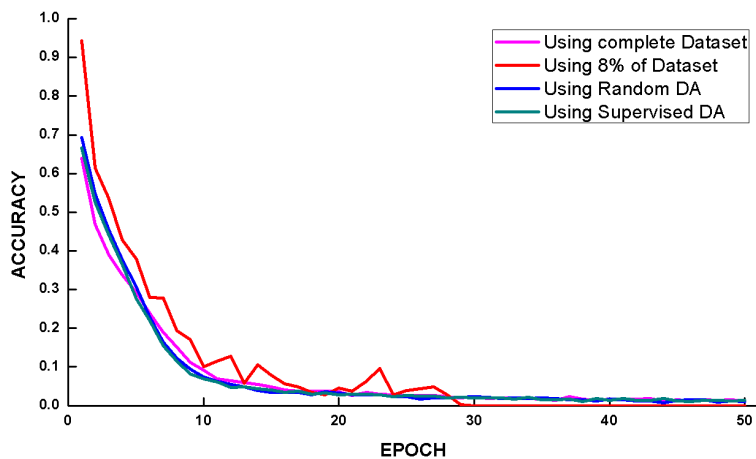
*Figure 60* ResNet-20 Validation Accuracy



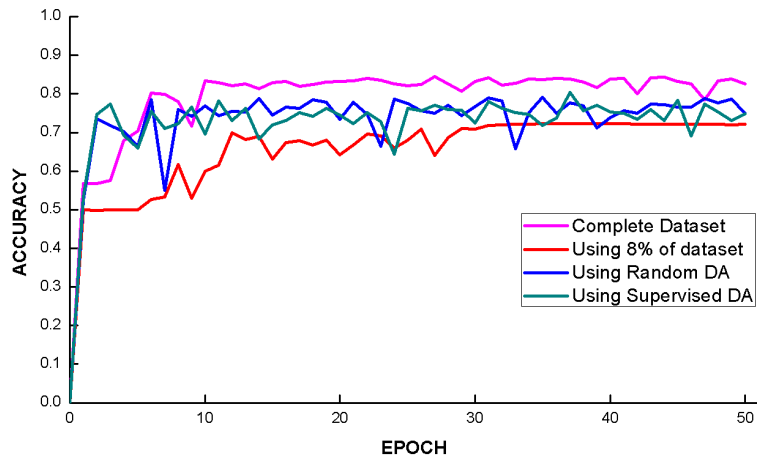
**Figure 61** ResNet-20 Validation Loss

### Comparing the Results Using AlexNet

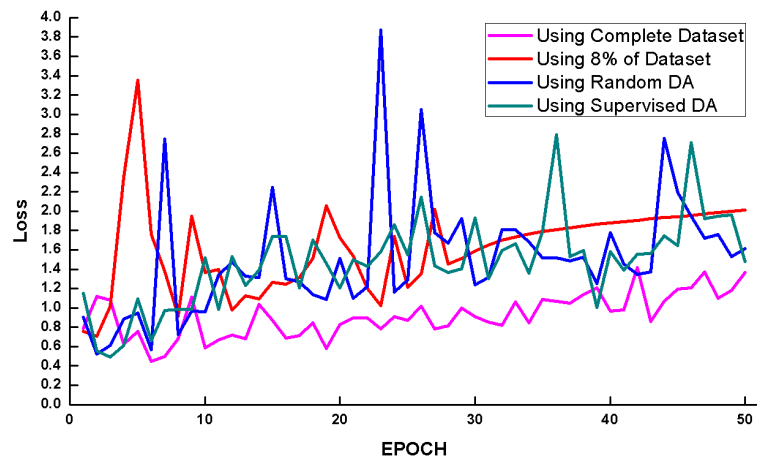
The model trained on the complete dataset is less overfit than the other models followed by the other models using Keras random data augmentation and the supervised data augmentation, the model trained on the 8% of Kaggle dataset show the highest overfit as we can see in Figure 63 and Figure 64 meanwhile the training loss reach the minimum value cause the model overfit the data.



**Figure 62** AlexNet Training Loss



**Figure 63** AlexNet Validation Accuracy



**Figure 64** AlexNet Validation Loss



## CONCLUSION

Using the complete dataset and a part of it has shown a major change in accuracy. the complete dataset overrides the partial one with 11.83% using ResNet-20 and with 10.8% using AlexNet. The large difference of the amount of data leads to a big accuracy gap between the 2 datasets, meanwhile data augmentation based on image manipulation shows that it is a good way to deal with the lack of data problem, using Random data augmentation on the partial dataset increase the validation accuracy by 5.84% using ResNet-20 and 3% using AlexNet. Supervised data augmentation is also showing a great improvement by increasing the validation accuracy by 8.34% using ResNet-20 and 2.83% using AlexNet. The results prove that There is a positive relation between the amount of data and the accuracy of the model "More training data means more accuracy".

Data augmentation showed that it's a good alternative to handle with the lack of data and overfitting problems in deep learning. Even small percentage in accuracy may make the difference.

Using data augmentation with a supervised way may increase the model's accuracy more than using a random operation, that open the door to another research including what are the transformation that may increase the accuracy more ... etc.

So, we may see in the future models that training on small dataset and reach the maximum accuracy using different data augmentation techniques.

## Bibliography

- [1] A. Fawzi, H. Samulowitz, D. Turaga, and P. Frossard, "Adaptive data augmentation for image classification," in *2016 IEEE International Conference on Image Processing (ICIP)*, Phoenix, AZ, USA, Sep. 2016, pp. 3688–3692, doi: 10.1109/ICIP.2016.7533048.
- [2] J. Arun Pandian, G. Geetharamani, and B. Annette, "Data Augmentation on Plant Leaf Disease Image Dataset Using Image Manipulation and Deep Learning Techniques," in *2019 IEEE 9th International Conference on Advanced Computing (IACC)*, Tiruchirappalli, India, Dec. 2019, pp. 199–204, doi: 10.1109/IACC48062.2019.8971580.
- [3] F. Chollet, "Deep learning with Python". Shelter Island, NY: Manning Publications Co,2018.
- [4] H. Chen, "Machine Learning for Information Retrieval: Neural Networks, Symbolic Learning, and Genetic Algorithms" University of Arizona, Management Information Department.
- [5] "Memo" Functions and Machine Learning by DONALD MICHIE Experimental Programming Unit, Department of Machine Intelligence and Perception, University of Edinburgh (Reprinted from Nature, Vol. 218, No. 5136, pp. 19-22, April 6, 1968).
- [6] M. Arif Wani, F. Ahmed Baht, S. Afzal, A. Iqbal Khan "Advances in Deep Learning" ISSN 2197-6503, Studies in Big Data 2020.
- [7] Z. Li, B. Ko, H. Choi "Pseudo-Labeling Using Gaussian Process for Semi-supervised Deep Learning", "[IEEE 2010 IEEE 10th International Conference on Data Mining (ICDM) - Sydney, Australia" 2010 page 767-772.
- [8] Robotics and Automation Laboratory (RAL), UNIVERSITY OF TORONTO Department of Mechanical and Industrial Engineering <https://ral.mie.utoronto.ca/research/ai-domains-of-interest-for-intelligent-robotics/>.
- [9] I. Goodfellow, Y. Bengio and A. Courville the MIT Press, Cambridge, England 2006 Massachusetts Institute of Technology.
- [10] R. Seigneuric and I. Bichindaritz "Decoding artificial intelligence and machine learning concepts for cancer research application" "Decoding artificial intelligence and machine learning concepts for cancer research application".
- [11] Y. Bo-Suk, H. Tian, Y. Zhong-Jun "Fault Diagnosis System of Induction Motors Using Feature Extraction, Feature Selection and Classification Algorithm" page 734.
- [12] Geoffrey E. Hinton (auth.), J. W. de Bakker, A. J. Nijman, P. C. Treleaven "PARLE Parallel Architectures and Languages Europe": Volume I: "Parallel Architectures Eindhoven", The Netherlands, June 15–19, 1987.
- [13] N. Buduma, *amentals of deep learning: designing next-generation machine intelligence algorithms*. Sebastopol, CA: O'Reilly Media, 2017.
- [14] F. Bre, Juan M. Gimenez, Victor D. Fachinotti "Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks"
- [15] S. Pattanayak "Pro Deep Learning with TensorFlow" 'A Mathematical Approach to Advanced Artificial Intelligence in Python' 2017.

- [16] V. Singh Bawa, V. Kumar "Linearized sigmoidal activation: A novel activation function with tractable non-linear characteristics to boost representation capability" Expert System with Applications (ESWA) 2018 journal homepage: [www.elsevier.com/locate/eswa](http://www.elsevier.com/locate/eswa).
- [17] W. Ballard "Hands-On Deep Learning for Images with TensorFlow" July 2018.
- [18] J. Patterson, A. Gibson "Deep Learning A Practitioner Approach", O'Reilly 2017.
- [19] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by backpropagating errors." *Cognitive Modeling* 5.3 (1988).
- [20] Handwritten Digit Recognition with a Back-Propagation Network, Y LeCun, and others, 1989, published at Neural Information Processing Systems (NIPS) conference.
- [21] S. Raschka, V. Marjalili, "Python Machine Learning with python, scikit-learn, and TensorFlow", 2<sup>nd</sup> edition, UK 2017.
- [22] I. Gogul, V.S ethiesh Kumar, (2017)'Flower Species Recognition System using Convolutional Neural Network and Transfer Learning', 4<sup>TH</sup> International Conference on Signal Processing, Communication and Networking (ICSCN-2017), March 16-18, Chennai, INDIA.
- [23] M. A. Wani, "Advances in deep learning". Singapore: Springer, 2020
- [24] Michael Nielsen "Neural Networks and Deep Learning" The original online book can be found at <http://neuralnetworksanddeeplearning.com>.
- [25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", arXiv e-prints, p. arXiv:1512.03385, 2015.
- [26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.
- [27] Shi, S. Zhou, X. Liu, Q. Zhang, M. Lu, and T. Wang, "Stacked deep polynomial network-based representation learning for tumor classification with small ultrasound image dataset," *Neurocomputing*, vol. 194, pp. 87–94, 2016.
- [28] M. Farid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger and H. Greenspan, Member, IEEE "GAN-Based Synthetic Medical Image Augmentation for increased CNN Performance in Liver Lesion Classification" arXiv: 1803.01229v1 [cs.CV] 3 Mars 2018.
- [29] A. Mikolajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," in *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, Swinoujście, May 2018, pp. 117–122, doi: 10.1109/IIPHDW.2018.8388338.
- [30] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice, Oct. 2017, pp. 843–852, doi: 10.1109/ICCV.2017.97.
- [31] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," in *Proc. of European Conference on Computer Vision (ECCV2014)*, 2014, pp. 818–833.

- [32] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," Technical report, University of Toronto, pp. 1–60, 2009.
- [33] D. Giardino, M. Matta, F. Silvestri, S. Spanò, and V. Trobiani, "FPGA Implementation of Hand-written Number Recognition Based on CNN," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 9, no. 1, p. 167, Feb. 2019, doi: 10.18517/ijaseit.9.1.6948.
- [34] P. Pawara, E. Okafor, L. Schomaker, and M. Wiering, "Data Augmentation for Plant Classification," in *Advanced Concepts for Intelligent Vision Systems*, vol. 10617, J. Blanc-Talon, R. Penne, W. Philips, D. Popescu, and P. Scheunders, Eds. Cham: Springer International Publishing, 2017, pp. 615–626.
- [35] R. Takahashi, T. Matsubara, and K. Uehara, "RICAP: Random Image Cropping and Patching Data Augmentation for Deep CNNs," p. 13.
- [36] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random Erasing Data Augmentation," *ArXiv170804896 Cs*, Nov. 2017, Accessed: Sep. 21, 2020. [Online]. Available: <http://arxiv.org/abs/1708.04896>.
- [37] H. M Amine "Deep Learning for Image Classification in Different Color Space", Master Thesis University Ibn Khaldoun Computer Science Department July 2019.
- [38] W. Burger and M. J. Burge, *Principles of Digital Image Processing: Advanced Methods*. London: Springer London, 2013.
- [39] M. A. Garduno-Ramon, J. I. Sanchez-Gomez, L. A. Morales Hernandez, J. P. Benitez-Rangel, R. A. Osornio-Rios "Methodology for automatic detection of trees and shrubs in aerial pictures from UAS "17 August 2016 p. 6.
- [40] S. Ioniță and D. Țurcanu-Caruțiu, "Intelligent Image Processing and Optical Means for Archeological Artifacts Examination," in *Advanced Methods and New Materials for Cultural Heritage Preservation*, D. Turcanu-Carutiu and R.-M. Ion, Eds. IntechOpen, 2019.
- [41] R. Pan, W. Gao, and J. Liu, "Skew Rectification for Yarn-dyed Fabric via FFT in HSL Color Space," in *2009 WRI World Congress on Software Engineering*, Xiamen, China, 2009, pp. 481–485, doi: 10.1109/WCSE.2009.149.
- [42] L. Ming, Y. Jie, S. Zhong-yi "Support vector regression based color image restoration in YUV color space", "Journal of Shanghai Jiao tong University (Science)", Chinese Electronic Periodical Services, 2010 page 31—35.
- [43] M. S. Devi and A. Mandowara, "Extended performance comparison of pixel window size for colorization of grayscale images using YUV color space," in *2012 Nirma University International Conference on Engineering (NUiCONE)*, Ahmedabad, Gujarat, India, Dec. 2012, pp. 1–5, doi: 10.1109/NUiCONE.2012.6493197.
- [44] X. Xiang, Y. Peng, X. Xiang, and L. Zhang, "A method of optical flow computation based on LUV color space," in *2009 International Conference on Test and Measurement*, Hong Kong, Hong Kong, Dec. 2009, pp. 378–381, doi: 10.1109/ICTM.2009.5413027.
- [45] Ian J. Goodfellow, J. Pouget-Abadiey, M. Mirza, B. Xu, D. Warde-Farley, S. Ozairz, A. Courville, Y. Bengio "Generative Adversarial Nets" 2014
- [46] F. Henrique, C. Aranha "Data Augmentation Using GANs" *Proceedings of Machine Learning Research XXX:1-16*, 2019

- [47] T. Karras, S. Laine, and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," IEEE Xplore p. 10.
- [48] L. A. Gatys, A. S. Ecker, and M. Bethge, "A Neural Algorithm of Artistic Style," *ArXiv150806576 Cs Q-Bio*, Sep. 2015, Accessed: Sep. 22, 2020. [Online]. Available: <http://arxiv.org/abs/1508.06576>.
- [49] J. Lemley, S. Bazrafkan, and P. Corcoran, "Smart Augmentation - Learning an Optimal Data Augmentation Strategy," *IEEE Access*, vol. 5, pp. 5858–5869, 2017, doi: 10.1109/ACCESS.2017.2696121.
- [50] <https://docs.anaconda.com/> "Anaconda Documentation". Retrieved 14 September 2020.
- [51] <https://jupyter-notebook.readthedocs.io/en/stable/notebook.html> "Jupyter Notebook Documentation". Retrieved 14 September 2020.
- [52] <https://www.python.org/doc/essays/blurb/> "What is Python? Executive Summary". Retrieved 14 September 2020
- [53] <https://keras.io/> Keras. Retrieved 14 September 2020.
- [54] <https://www.tensorflow.org/> Tensorflow. Retrieved 15 September 2020.
- [55] S. van der Walt, JL Schönberger, J Nunez-Iglesias, F. Boulogne; J.D. Warner, N. Yager, E. Gouillart, T. Yu, the scikit-image contributors (2014). "scikit-image: image processing in Python" *arXiv:1407.6245*.
- [56] <https://pillow.readthedocs.io/en/stable/#> Pillow. Retrieved 15 September 2020.
- [57] S. Tosi, Matplotlib for python developers: build remarkable publication quality plots the easy way. Birmingham: Packt Publ, 2009.
- [58] dataset used link: <https://www.kaggle.com/c/dogs-vs-cats/data> Retrieved 18 September 2020.