الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي والبحث العلمي

UNIVERSITÉ BADJI MOKHTAR - ANNABA

BADJI MOKHTAR – ANNABA UNIVERSITY

جامعة باجي مختار – عنابـــة

Faculté : Science d'ingéniorat

Département : Electronique

Domaine : Sciences et Technologies

Filière : Electronique

Spécialité : Electronique des systèmes embarqués

Mémoire

Présenté en vue de l'obtention du Diplôme de Master

Thème:

Computer Vision and Machine Learning On RPI for hand gesture recognition

Présenté par : *Guebla Chems Eddine*

Encadrant : *Fezari Mohamed* *Grade : Professeur*

Université :Badji Mokhtar-Annaba

Jury de Soutenance :

Hamdi Rachid	Professeur	Badji Mokhtar Annaba	Président
Mokhneche Azzouz	Maître de conference(A)	Badji Mokhtar Annaba	Examinateur
Fezari Mohamed	Professeur	Badji Mokhtar Annaba	Encadrant

Année Universitaire : 2019/2020

Contents

Acknowledgment

Abstract

List of Figures

List of Algorithms

List of Abbreviations

I Introduction

I.1 General introduction..1

I.2 Stat of art...2

II Artificial Intelligence **3**

II.1 Introduction **4**

II.1.1 Modern history of AI...4

II.1.2 Definition of AI 5

II.1.3 AI on computer vision...5

II.2 Artificial Neural Networks **7**

II.2.1 The neuron... 8

II.2.2 ANN structure.. 8

II.3 Machine Learning **13**

II.3.1 What is Machine Learning ?...13

II.3.2 How does Machine Learning Work?.................................. 14

II.3.3 Types of Machine Learning...15

II.4 Deep Learning **17**

II.4.1 What is Deep Learning?.. 18

II.4.2 Deep Learning for visual recognition..................................24

II.5 Convolutional Neural Networks **33**

II.5.1 Definition... 34

II.5.2 Model architecture... 35

II.5.3 3d CNN - vs - Recurrent Neural Networks..........................41

III Problem statement and working algorithm **43**

III .1 Problem statement.. 43

III .2 Image Recognition Algorithms...44

III .2.1 CNNs and Their Role in Image Recognition...........................46

III .2.2 Effectiveness and Limitations of CNNs.. 46
III .3 Some algorithms for special procedures and functions............................ .47

IV Results and Discussion **52**
IV.1 System Design 52
IV.1.1 System Architecture..52
IV.1.2 Digram...53
IV.2 Implementation...54
IV.1.1 Overview on RPI platforme...54
IV.1.2 MNIST dataset...55
IV.1.3 Data CollectionPre-Processing..56

IV.3 Results and Discussion 57
IV.3.1 Presentation of GUI 57
IV.3.2 Notes 59

V Conclusion
V.1 Conclusion 65
V.2 Future Work..65

Webography 69

Bibliography 72

Appendix

Abstract

In this project we will introduce a **Python**-based, **deep learning** hand gesture recognition model , implemented on a **Raspberry Pi 3** model B electronique board using a "machine learning" algorithm.

The model works in real-time and can recognize 25 different hand gestures from a simple RGB webcam stream. It uses a simple laptop camera or an external camera connected to the embedded system .With a simple graphic user (**GUI**) interface based on **Qt** platforme ; a free software tool for designing and building graphical user interface, In order to access the application.

The objective of our work is to detect the movement of gestures and their meaning with a basic level of hardware component like camera and an electronic board ,and an interface to access the application Instead of using further technology .

Key words : Artificial intelligence , computer vision , hand gestures ,machine learning , deep learning.

Résumé

Dans ce projet, nous présenterons un modèle "deep learning" de reconnaissance des gestes basé sur Python, implémenté sur une carte électronique Raspberry Pi 3 model B à l'aide de "machine learning" algorithme.

Le modèle fonctionne en temps réel et peut reconnaître 25 gestes de main différents à partir d'un simple flux de webcam RGB. il utilise une simple caméra portable ou une caméra externe connectée au système embarqué .Avec une interface graphique simple (GUI) basée sur Qt plateforme; un outil logiciel gratuit pour concevoir et construire une interface utilisateur graphique, afin d'accéder à l'application.

L'objectif de notre travail est de détecter le mouvement des gestes et leur signification avec un niveau de base de composant matériel comme une caméra et une carte électronique, et une interface pour accéder à l'application au lieu d'utiliser d'autres technologies.

mots clés : Intelligence artificielle , vision par ordinateur , gestes de la main,reconnaissance , machine learning , deep learning.

الملخص

في هذا المشروع ، سنقدم نموذج التعرف على إيماءات اليد معتمدين على تقنية "deep learning" و استنادًا على لغة البرمجة Python ، والذي تم تنفيذه على بطاقة إلكترونية Raspberry Pi 3 model B باستخدام خوارزمية ""machine learning"

يعمل النموذج في الوقت الفعلي ويمكنه التعرف على 25 إيماءة يدوية مختلفة من دفق كاميرا RGB بسيطة. يستخدم كاميرا الكمبيوتر المحمول أو كاميرا خارجية متصلة بالنظام المضمن ، مع واجهة رسومية بسيطة (GUI) تعتمد على منصة Qt؛وهي أداة برمجية مجانية لتصميم وبناء واجهة مستخدم رسومية بهدف الوصول إلى التطبيق.

الهدف من عملنا هو الكشف عن حركة الإيماءات ومعناها بمستوى بسيط من المكونات مثل كاميرا ولوحة إلكترونية,وواجهة للوصول إلى التطبيق بدلاً من استخدام المزيد من التقنيات.

الكلمات المفتاحية : الذكاء الاصطناعي , الرؤية الحاسوبية , التعرف على حركات اليد , التعلم الآلي , التعلم العميق.

List of Figures

Chapitre I

1.American Sign Language. 1

2.Hand gesture classification using python 2

3. :A human computer interface(PC , Kinect and gloves). 2

Chapitre II

1.Pixel representation 6

2. A cartoon drawing of a biological neuron (a) and its mathematical model (b).. 8

3. The basic structure of an ANN 8

4. A mathematical representation of a node. 9

5. Functionality steps of a single node. 9

6. Forward propagation. 12

7. Adjustment of the weights using gradient descent algorithm 12

8.Backpropagation. 12

9.Basics steps of machine learning 14

10.Supervised learning process 15

11. From Vision-Based Classification of Skin Cancer 15

12.Supervised learning process 16

13.Collection of different genes. 16

14.Gradient descent (simplified). 22

15.Normal Gradient Descent limit. 22

16.Example of Handwritten Digits From the MNIST Dataset. 25

17.Example of Image Classification With Localization of a Dog from VOC 2012. 25

18.Example of Image Classification With Localization of Multiple Chairs From VOC 2012. . . . 26

19.Example of Object Detection. 27

20.Example of Object Segmentation 27

21.Example of Neural Style Transfer 28

22.Examples of Photo Colorization 29

23. Example of Photo Inpainting. 29

24. Example of the Results From Different Super-Resolution Techniques. 30
25.Example of Styling Zebras and Horses. 31
26.A CNN sequence to classify handwritten digits 33
27.4x4x3 RGB Image. 33
28.Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature 34
29.Convolution operation on a MxNx3 image matrix with a 3x3x3 Kernel. 35
30.Three nonlinear activation functions adopted by CNNs. 36
31.Types of Pooling. 37
32.Fully Connected Layer. 37

Chapitre III

33.Presentation of a semantic segmentation problem. 43

Chapitre IV

34.System Architecture presentation.. 52
35.Hand gesture detection diagram.. 53
36:Raspberry Pi platforme and Camera Module. 54
37.Raspberry Pi3 model B with camera. 55
38.Dashboard :welcome. 57
39.Dashboard :Task 1 , create new gesture. 57
40.Dashboard :Task2 ,scan gesture. 58
41.Dashboard :Task3 ,create sequence of characters. 58
42.Dashboard :Task4 ,export file. 59
43.Sample image of the letter “C” from the training dataset. 59
44.Outputs from the first hidden layer. 60
45.Outputs from the fourth hidden layer. 60
46.Confusion matrix of the CNN’s output of the test data. 60

List of Algorithms

Forward propagation 20

Backpropagation 21

Gradient descent 22

Stochastic gradient descent 23

Logistic Regression. 47

Decision Tree Algorithm. 47

Random Forest 48

Naive Forest Classifier 48

K-Nearest Neighbor(KNN) 49

Artificial Neural Networks and Deep Neural Networks 49

List of Abbreviations

AI	:	Artificial Intelligence
ANN	:	Artificial Neural Networks
ASL	:	American Sign Language
API	:	An application Program Interface
CNN	:	Convolutional Neural Networks
CV	:	Computer Vision
DNN	:	Deep Neural Networks
DL	:	Deep Learning
ML	:	Machine Learning
MNIST	:	Modified National Institute of Standards and Technology database
RNN	:	Recurrent Neural Networks
ReLU	:	Rectified Linear Unit
RPI	:	Raspberry Pi

Chapter I

Introduction

Chapter II

Artificial Intelligence

Chapter III

Problem statement and working algorithm

Chapter IV

Results and Discussion

Chapter V

Conclusion

CHAPTER I: Introduction

I.1 General Introduction

Some of the major problems faced by a person who is unable to speak is they cannot express their emotion as freely in this world. Utilize that voice recognition and voice search systems in smartphones [1]. Audio results cannot be retrieved. They are not able to utilize (Artificial Intelligence/personal Butler) like google assistance, or Apple's SIRI etc [2]. because all those apps are based on voice controlling.

For this, the thesis is structured in four (4) chapters:
In this chapter, we presented a general idea about the project and the application and the reason behind it .

In the second chapter, we take a deep look at the Artificial Intelligence field ,from the history of this field to its definition passing by Artificial Neural Networks structure, Machine Learning technique to Deep Learning algorithm,and ending with Convolutional Neural Networks (CNN) .

In the third chapter, we started to present the problem statement and the interest of the image recognition algorithms and some other algorithms for special procedures .

In the fourth chapter ,having an idea about the application's programme ,the result of this programme and a brief discussion about it .

Finally, in the fifth chapter, This continuation of memory is followed by a general conclusion highlighting the strong points of this work of end of studies master_2 , and other future work that may be related to our project .

I.2 state of art

Hand gesture is one of the methods used in sign language for non-verbal communication. It is most commonly used by deaf & dumb people who have hearing or talking disorders to communicate among themselves or with normal people [3]. Various sign language systems have been developed by many manufacturers around the world but they are neither flexible nor cost-effective for the end users [6].

One of the solutions to communicate with the deaf-mute people is by using the services of a sign language interpreter. But the usage of sign language interpreters could be expensive [3]. Cost-effective solution is required so that the deaf-mute and normal people can communicate normally and easily [3].
Our strategy involves implementing such an application which detects pre-defined American Sign Language (ASL) through hand gestures. For the detection of movement of gesture, we would use a basic level of hardware component like camera and interfacing is required. Our application would be a comprehensive User-friendly Based system built on PyQt5 module.

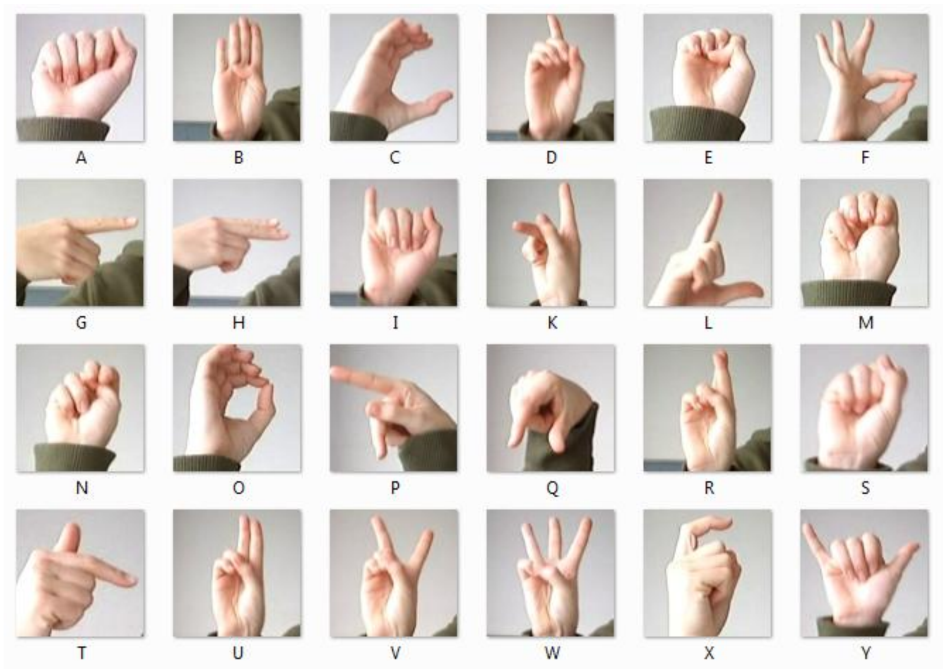


Figure 1 :American Sign Language

Instead of using technology like gloves or Kinect, we are trying to solve this problem using state of the art computer vision and machine learning algorithms [6].

Our application will consist of two core modules one is that simply detects the gesture and displays appropriate alphabet. The second is after a certain amount of interval period the scanned frame would be stored into a buffer so that a string of characters could be generated forming a meaningful word.

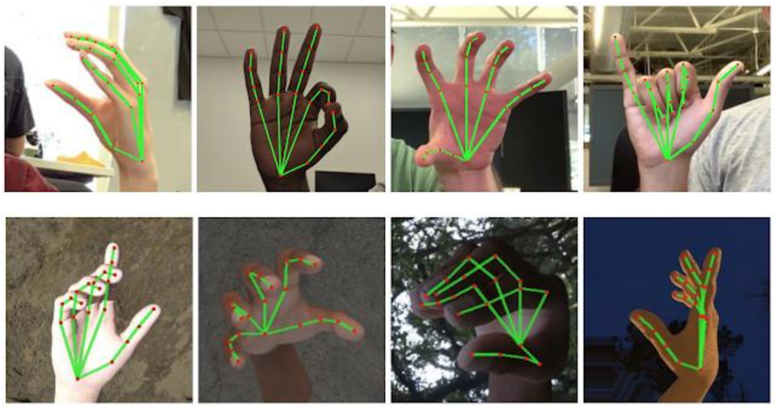


Figure 2 :Hand gesture classification using python.

Additionally, an add-on facility for the user would be available where a user can build their own custom-based gesture for a special character like period (.) or any delimiter so that a user could form a whole bunch of sentences enhancing this into paragraph and likewise. Whatever the predicted outcome was, it would be stored into a **.txt file.**

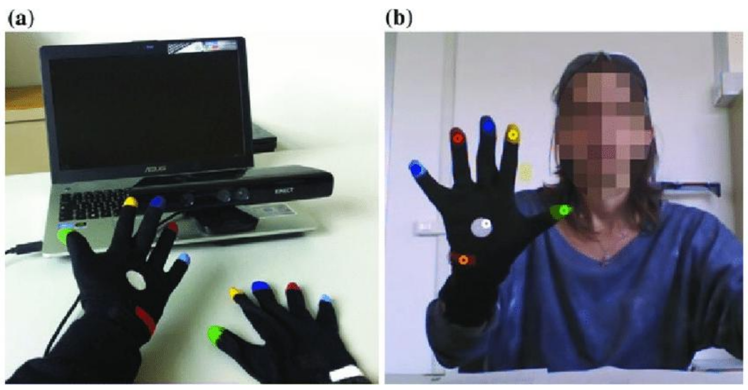


Figure 3 :A human computer interface(PC , Kinect and gloves).

CHAPTER II: Artificial Intelligence

*II.*1 Introduction

When programmable computers were first conceived, people wondered whether such machines might become intelligent, over a hundred years before one was built (Lovelace, 1842). Today, artificial intelligence (AI) is a thriving field with many practical applications and active research topics. We look to intelligent software to automate routine labor, understand speech or images, make diagnoses in medicine and support basic scientific research.

Artificial intelligence is one of the most controversial yet trendy topics nowadays. Everyone has high hopes about the evolution of AI—some are intimidated by this type of technology and some even fear it.

This chapter focuses on the main achievements of AI in modern history and expressing more closely what AI really is and the relation between AI and computer vision, and the fields and subfields of AI from neural networks, passing to machine learning and deep learning, and CNN/RNN.

II.1.1 Modern history of AI

John MacCarthy invented the word Artificial Intelligence when he held the first academic conference on the subject in the campus of Dartmouth College in 1956. During the 1960's the first article of AI was published under the name "Computers and Thought" and the computer mouse was invented by Doug Engelbart. The first ever international conference about AI was held in 1969 in Washington DC. In the 1970's a famous Scottish robot called Freddy was built, could spot and compile models with eye technology. INTERNIST-program was also invented in the 1970's the program could provide medical diagnoses based on what information it receives.

In the 1980's neural networks became widely used and "The Society of Mind" a theoretical description of the collective mind, was published by Marvin Minsky. In the 1990's AI took significant developments in areas such as machine learning, virtual reality and in games. A chess program built on AI won the chess world championship and the first autonomous robotics machinery system "Sojourner" was deployed on the surface of Mars by NASA.
Interactive robot pets (a.k.a "smart toys") became commercially available, Stanford's autonomous vehicle, Stanley, won DARPA Grand Challenge race and The Nomad robot explored the remote regions of Antarctica looking for meteorite samples during the early years of the 21st century. (Tekoäly 2018)

AI is one of the most mysterious subjects in computer science although it has been studied for decades. The subject is very vague and ranges from machines capable of thinking to search algorithms used to play board games.

CHAPTER II: Artificial Intelligence

Algorithms, machine learning algorithms, and integrating statistical analysis into understanding the world have been the main advances of AI in the past 60 years. AI has not had rapid progress over the decades and significant AI breakthroughs have been guaranteed in “10 years” for the past 60 years. In the field of AI expectations always outran the reality. (University of Washington 2006).

II.1.2 Definition of AI

1. Charniak and McDermott [1]: Artificial intelligence is the study of mental faculties through the use of computational models.
2. Shapiro [2]: Artificial intelligence is a field of science and engineering concerned with the computational understanding of what is commonly called intelligent behavior, and with the creation of artifacts that exhibit such behavior.
3. Rich and Knight [3]: The study of how to make computers do things at which, at the moment, people are better.

4. In a nutshell and much more simple words Artificial intelligence is a part of computing science that focuses on creating intelligent machines and programs. The purpose of artificial intelligence is to try to mimic human consciousness and perform tasks such as human beings. In practice it means the ability of a machine or program to think and learn.

II.1.3 AI and computer vision

“ If we want machines to think, we need to teach them to see**. ”**

-Fei Fei Li, Director of Stanford AI Lab and Stanford Vision Lab

Artificial Intelligence has been witnessing a monumental growth in bridging the gap between the capabilities of humans and machines. Researchers and enthusiasts alike, work on numerous aspects of the field to make amazing things happen. One of many such areas is the domain of Computer Vision.
Computer vision is a brache or field of artificial intelligence that trains computers to interpret and understand the visual world. Machines can accurately identify and locate objects then react to what they "see" using digital images from cameras , videos ,and deep learning models.

CHAPTER II: Artificial Intelligence

An overview of computer vision

Starting in the late 1950s and early 1960s, the goal of image analysis was to mimic human vision systems and to ask computers what they see. Prior to this, image analysis had been completed manually using x-rays, MPIs or hi-res space photography. Nasa's map of the moon took the lead with digital image processing, but wasn't fully accepted until 1969.

As computer vision evolved, programming algorithms were created to solve individual challenges. Machines became better at doing the job of vision recognition with repetition. Over the years, there has been a huge improvement of deep learning techniques and technology. We now have the ability to program supercomputers to train themselves, self-improve over time and provide capabilities to businesses as online applications.
Think of computer vision as working with millions of calculations in order to recognize patterns and to have the same accuracy as the human eye. Patterns can be seen physically or can be observed mathematically by applying algorithms.

The Breakdown of Computer Vision

Images are broken down into pixels, which are considered to be the elements of the picture or the smallest unit of information that make up the picture.
Computer vision is not just about converting a picture into pixels and then trying to make sense of what's in the picture through those pixels. You have to understand the bigger picture of how to extract information from those pixels and interpret what they represent.

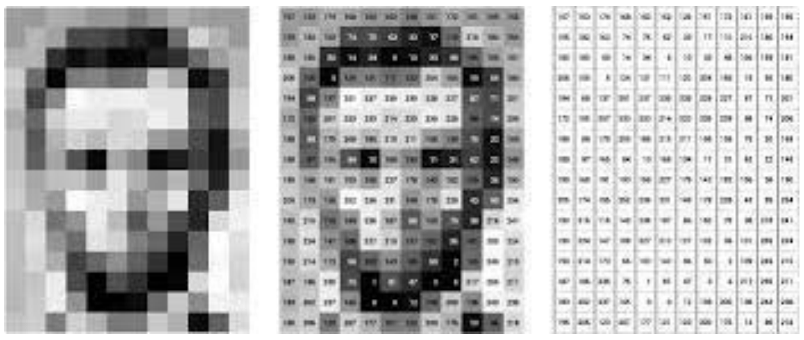


Figure 1: Pixel representation

We will discuss some of computer vision problems with further explanation at **chapter** (**2.3.4**) .

Who Uses Computer Vision?

1. **Amazon** unveiled 18 AmazonGo stores where shoppers can bypass lines and pay for items right away. With computer vision, cameras are used to let employees know when

something was taken off the shelves. It can also identify returned items or removed items from a shopping cart. Your Amazon prime account will be charged one you finish filling your "virtual basket."

2. Computer vision in **retail stores** can also improve security. Tracking each person inside the store at all times makes sure each shopper pays for the merchandise.
3. **Facebook** uses facial recognition ("DeepFace") when automatically tagging photos that are posted to your profile. After negative feedback from many audiences due to privacy, Facebook only allows the recognition to opt into it.

1. **Military | Space** — Countries across the globe are embedding AI into weapons, transportation, target recognition, healthcare in the field, simulation training, and other systems used on land, air, sea, and space. AI systems based on these platforms are less reliant on human input because they enhance performance while requiring less maintenance.
   With current systems, AI decreases cyber-attacks and can protect networks, computers, programs, and data from any unauthorized access.

1. **Cars** — Computer vision is a hot topic in the car industry. Companies like Tesla and Google are building self-driving cars. Cars today have Adaptive/Dynamic Cruise Control that has the ability to maintain a safe distance from the vehicles ahead.

II.2 Artificial Neural Networks

Artificial neural networks are one of the main tools used in machine learning. As the "neural" part of their name suggests, they are brain-inspired systems which are intended to replicate the way that we humans learn [5].

II.2.1 The neuron

The basic building block of a neural network is a neuron, An artificial neural network (ANN) is a system that is based on the biological neural network, such as the brain. The brain has approximately 100 billion neurons, which communicate through electro-chemical signals. The neurons are connected through junctions called synapses. Each neuron receives thousands of connections with other neurons, constantly receiving incoming signals to reach the cell body. If the resulting sum of the signals surpasses a certain threshold, a response is sent through the axon. The ANN attempts to recreate the computational mirror of the biological neural network, although it is not comparable since the number and complexity of neurons and the use in a biological neural network is many times more than those in an artificial neural network [6].

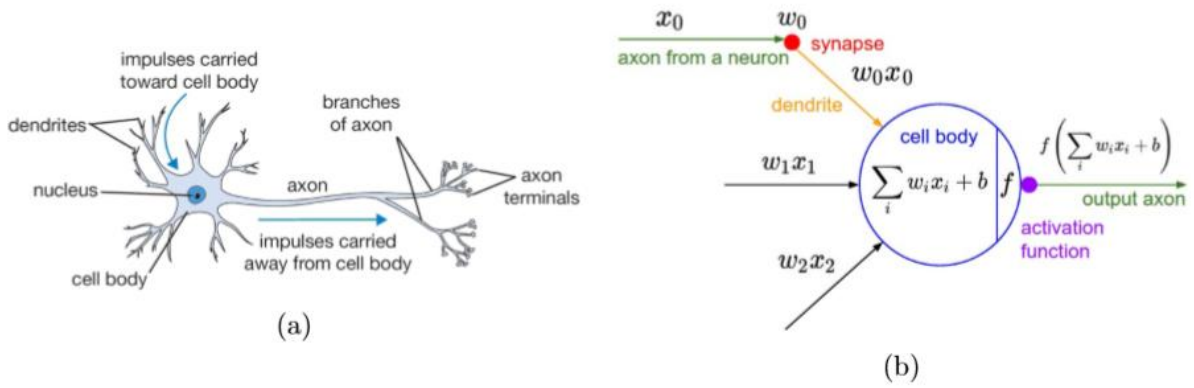


Figure 2: A cartoon drawing of a biological neuron (a) and its mathematical model (b).

II.2.2 ANN structure

Artificial Neural Networks (ANN) are multi-layer fully-connected neural nets that look like the figure below. They consist of an input layer, multiple hidden layers, and an output layer. Every node in one layer is connected to every other node in the next layer. We make the network deeper by increasing the number of hidden layers [4].

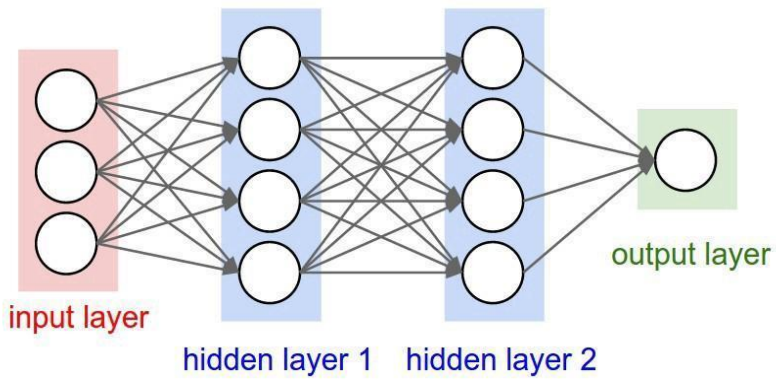


Figure 3: The basic structure of an ANN.

If we zoom in to one of the hidden or output nodes, what we will encounter is the figure below.

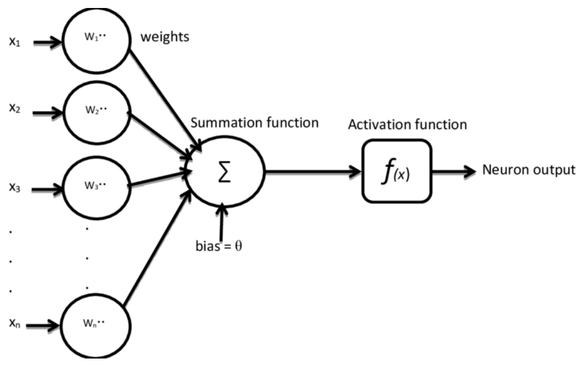


Figure 4: A mathematical representation of a node.

A given node takes the weighted sum of its inputs, and passes it through a non-linear activation function. This is the output of the node, which then becomes the input of another node in the next layer. The signal flows from left to right, and the final output is calculated by performing this procedure for all the nodes. Training this deep neural network means learning the weights associated with all the edges.

The equation for a given node looks as follows. The weighted sum of its inputs passed through a non-linear activation function. It can be represented as a vector dot product, where n is the number of inputs for the node.

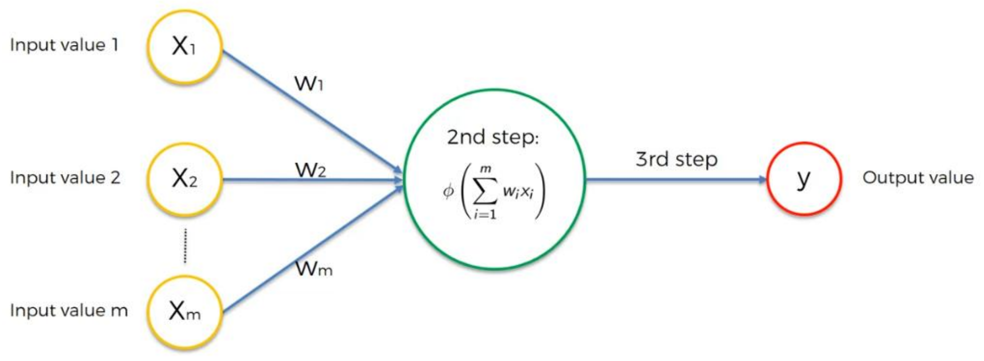
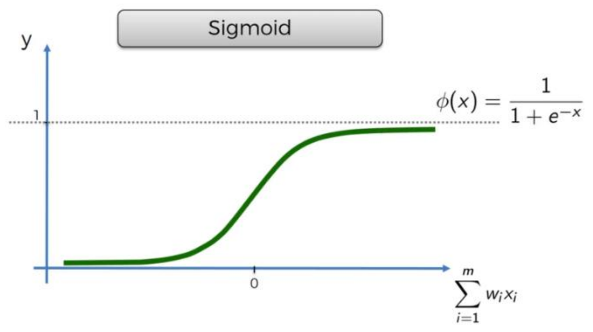


Figure 5: Functionality steps of a single node

Transfer (Activation) Functions or Non Linearity

The transfer function translates the input signals to output signals. The capacity of the neural networks to approximate any functions is directly the result of the non-linear activation functions. Every kind of activation function takes a vector and performs a certain fixed pointwise operation on it. There are three main activation functions.

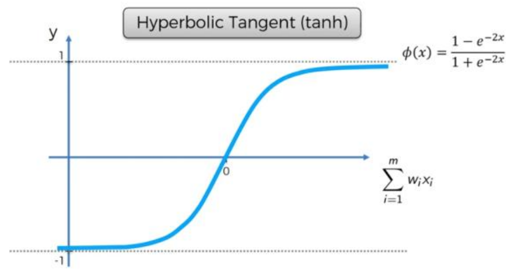
Sigmoid: The Sigmoid nonlinearity has the following mathematical form



y = σ(x) = 1/(1 + exp−x)

It takes a real value and squashes it between 0 and 1. However, when the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero. Thus, the backpropagation algorithm fails at modifying its parameters and the parameters of the preceding neural layers.

Hyperbolic Tangent: The TanH nonlinearity has the following mathematical form

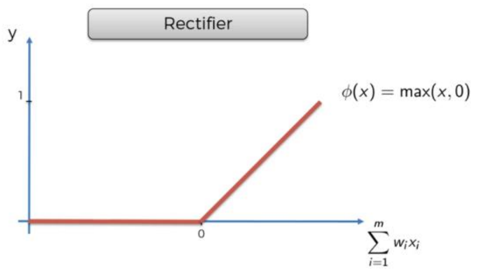


y = 2σ(2x)−1

It squashes a real-valued number between -1 and 1. However it has the same drawback as the sigmoid.

Rectified Linear Unit: The ReLU has the following mathematical form, for more information about the role of this function check

y = max(0,x)



The hidden layer neurons follow all the same functionality steps:

1. Receive a real number as input from another neuron.
2. Multiply it by a weight (W) and add a bias (b) to it. One neuron has one pair of W and (b) associated to each of the neurons that give an input to it.
3. Apply a nonlinear operation to the result to break the linearity. Here we can typically see functions such as Rectified Linear Unit (ReLU) or sigmoid function.
4. Send the current value to all the neurons connected to its output.

The output layer is interpreted as a one-hot sequence; the neuron with the higher output value is interpreted as a 1 and all the others as 0. So, the decision of the classification is the one associated with that neuron .

So far we have described the forward pass, meaning given an input and weights how the output is computed. After the training is complete, we only run the forward pass to make the predictions. But we first need to train our model to actually learn the weights, and the training procedure works as follows:

1. Randomly initialize the weights for all the nodes. There are smart initialization methods which we will explore in another article.
2. For every training example, perform a **forward pass** using the current weights, and calculate the output of each node going from left to right. The final output is the value of the last node.

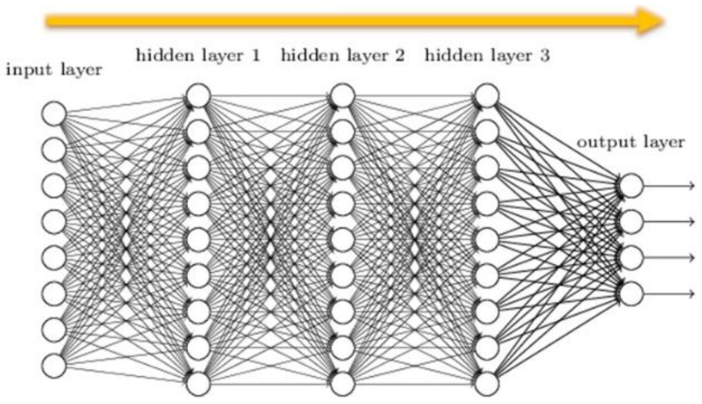


Figure 6: Forward propagation

1. Compare the final output with the actual target in the training data, and measure the error using a loss function.

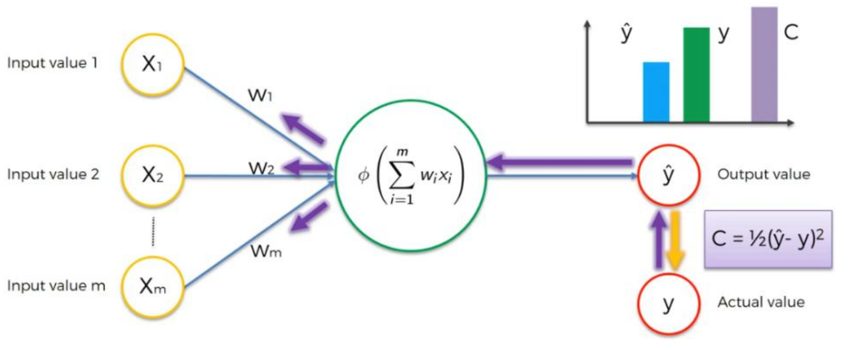


Figure 7: Adjustment of the weights using gradient descent algorithm.

1. Perform a **backwards pass** from right to left and propagate the error to every individual node using backpropagation. Calculate each weight's contribution to the error, and adjust the weights accordingly using gradient descent. Propagate the error gradients back starting from the last layer.

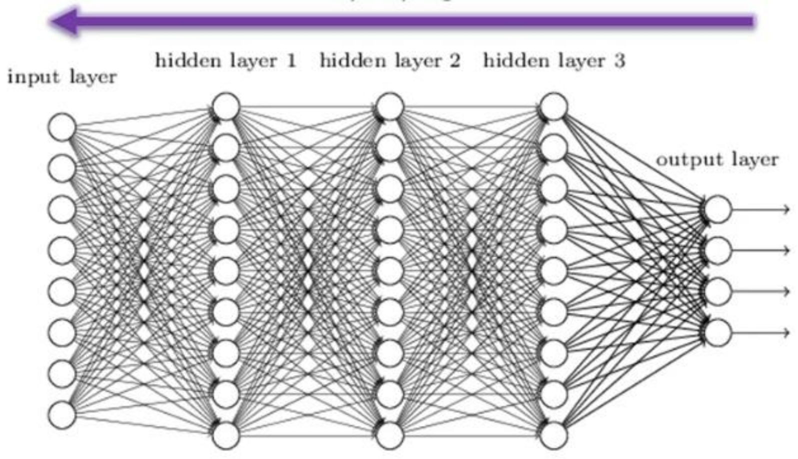


Figure 8: Backpropagation.

Backpropagation with **gradient descent** is literally the "magic" behind deep learning models. It's a rather long topic and involves some calculus, so we won't go into the specifics in this applied deep learning series. For a detailed explanation of gradient descent refer [7]. A basic overview of backpropagation is available [8]. For a detailed mathematical treatment refer [9] and [10]. And for more advanced optimization algorithms refer [11].

II.3 Machine Learning

II.3.1 What is Machine Learning?

1. Two definitions of Machine Learning are offered. **Arthur Samuel** [1] described it as: "the field of study that gives computers the ability to learn without being explicitly programmed." This is an older, informal definition.
2. **Tom Mitchell** [2] provides a more modern definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [3] .

Well, **Machine Learning** in nutshell is a concept which allows the machine to learn from examples and experience, and that too without being explicitly programmed. So instead of you writing the code, what you do is you feed data to the generic algorithm, and the algorithm/ machine builds the logic based on the given data [4] .

Example : playing checkers.
E = the experience of playing many games of checkers
T = the task of playing checkers.
P = the probability that the program will win the next game.

II.3.2 How does machine learning work ?

Machine Learning algorithm is trained using a training data set to create a model. When new input data is introduced to the ML algorithm, it makes a prediction on the basis of the model.

The prediction is evaluated for accuracy and if the accuracy is acceptable, the Machine Learning algorithm is deployed. If the accuracy is not acceptable, the Machine Learning algorithm is trained again and again with an augmented training data set.
This is just a very high-level example as there are many factors and other steps involved.

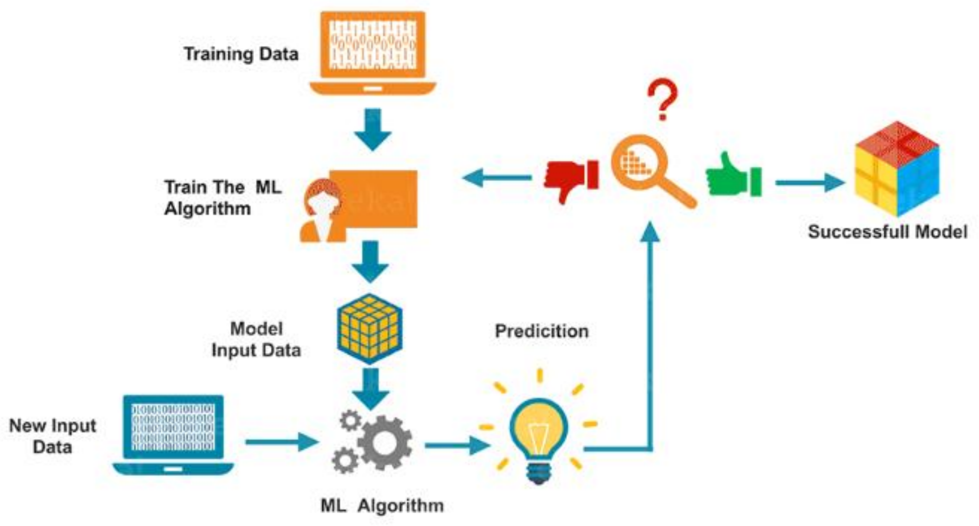


Figure 9: Basics steps of machine learning.

II.3.3 Types of Machine Learning

Machine learning is sub-categorized to three types:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

In general, any machine learning problem can be assigned to one of two broad classifications: **Supervised learning** and **Unsupervised learning**.

A. Supervised Learning

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output. Supervised learning problems are categorized into "regression" and "classification" problems.

A.1 classification model

A type of machine learning model for distinguishing among two or more discrete classes. For example, a natural language processing classification model could determine whether an input sentence was in French, Spanish, or Italian. Compare with a regression model.

A.2 Regression model

A type of model that outputs continuous (typically, floating-point) values. Compare with classification models, which output discrete values.

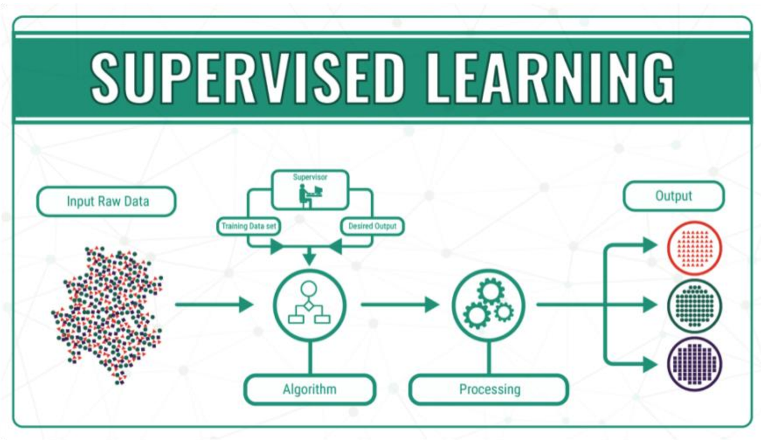


Figure 10: Supervised learning process .

Example 1:

1. **A regression model** predicts continuous values. For example, regression models make predictions that answer questions like the following:
    1. What is the value of a house in California?
    2. What is the probability that a user will click on this ad?
2. **A classification model** predicts discrete values. For example, classification models make predictions that answer questions like the following:
    1. Is a given email message spam or not spam?
    2. Is this an image of a dog, a cat, or a hamster?

Example 2:

1. **Regression** - Given a picture of a person, we have to predict their age on the basis of the given picture
2. **Classification** - Given a patient with a tumor, we have to predict whether the tumor is malignant or benign.

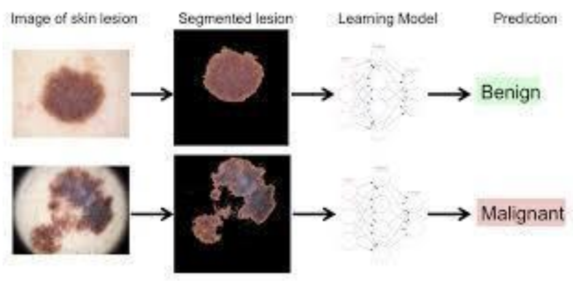


Figure 11: from Vision-Based Classification of Skin Cancer .

B. Unsupervised Learning

Unsupervised learning allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables.

We can derive this structure by clustering the data based on relationships among the variables in the data.

With unsupervised learning there is no feedback based on the prediction results.

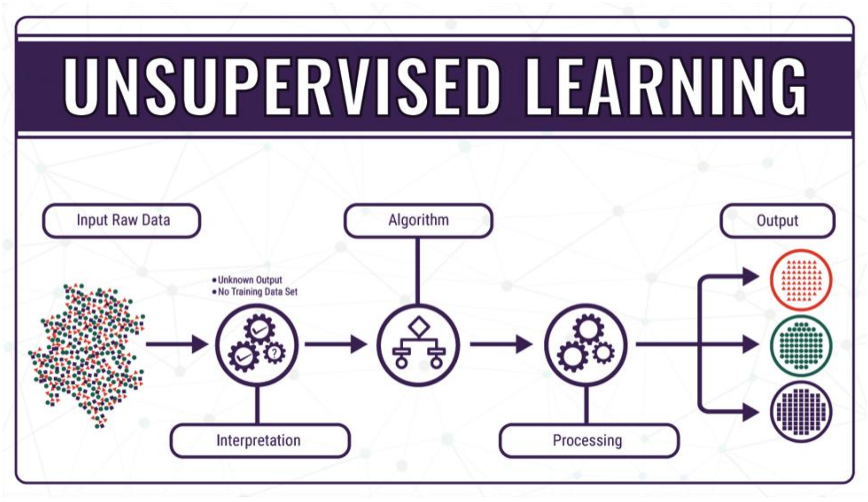
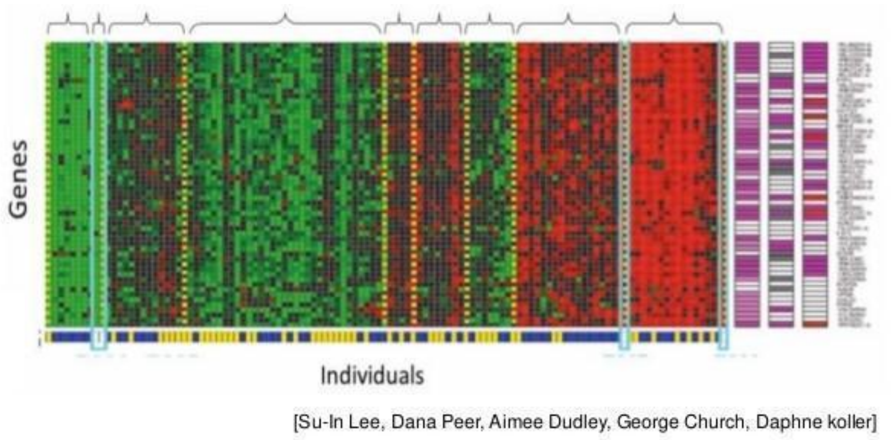


Figure 12: Unsupervised learning process .

Example:

1. **Clustering**: Take a collection of 1,000,000 different genes, and find a way to automatically group these genes into groups that are somehow similar or related by different variables, such as lifespan, location, roles, and so on.



1. **Non-clustering**: The "Cocktail Party Algorithm", allows you to find structure in a chaotic environment. (i.e. identifying individual voices and music from a mesh of sounds at a cocktail party).

II.4 Deep Learning

In the early days of artificial intelligence, the field rapidly tackled and solved problems that are intellectually difficult for human beings but relatively straightforward for computers-problems that can be described by a list of formal, mathematical rules. The true challenge to artificial intelligence proved to be solving the tasks that are easy for people to perform but hard for people to describe formally- problems that we solve intuitively, that feel automatic, like recognizing spoken words or faces in images.

This chapter is about a solution to these more intuitive problems. This solution is to allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined through its relation to simpler concepts. By gathering knowledge from experience, this approach avoids the need for human operators to formally specify all the knowledge that the computer needs.

The hierarchy of concepts enables the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI deep learning.
Deep Learning is a new area of Machine Learning research, which has been introduced with the objective of moving Machine Learning closer to one of its original goals: Artificial Intelligence.

II.4.1 What is Deep Learning ?

Definition

Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network [12].

So, to better understand this concept let's answer some common questions.

How does it work ?

At a very basic level, deep learning is a machine learning technique. It teaches a computer to filter inputs through layers to learn how to predict and classify information. Observations can be in the form of images, text, or sound.

The inspiration for deep learning is the way that the human brain filters information. Its purpose is to mimic how the human brain works to create some real magic.

Neurons by themselves are kind of useless. But when you have lots of them, they work together to create some serious magic. That's the idea behind a deep learning algorithm! You get input from observation and you put your input into one layer. That layer creates an output which in turn becomes the input for the next layer, and so on. This happens over and over until your final output signal !
The neuron (node) gets a signal or signals (input values), which pass through the neuron. That neuron delivers the output signal.

Think of the input layer as your senses: the things you see, smell, and feel, for example. These are independent variables for one single observation. This information is broken down into numbers and the bits of binary data that a computer can use. You'll need to either standardize or normalize these variables so that they're within the same range.
They use many layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output of the previous layer for its input. What they learn forms a hierarchy of concepts. In this hierarchy, each level learns to transform its input data into a more and more abstract and composite representation.

That means that for an image, for example, the input might be a matrix of pixels. The first layer might encode the edges and compose the pixels. The next layer might compose an arrangement of edges. The next layer might encode a nose and eyes. The next layer might recognize that the image contains a face, and so on.

What happens inside the neuron ?

The input node takes in information in a numerical form. The information is presented as an activation value where each node is given a number. The higher the number, the greater the activation.
Based on the connection strength (weights) and transfer function, the activation value passes to the next node. Each of the nodes sums the activation values that it receives (it calculates the **weighted sum)** and modifies that sum based on its transfer function. Next, it applies an activation function. An activation function is a function that's applied to this particular neuron. From that, the neuron understands if it needs to pass along a signal or not.
Each of the synapses gets assigned weights, which are crucial to **Artificial Neural Networks** (ANNs). Weights are how ANNs learn. By adjusting the weights, the ANN decides to what extent signals get passed along. When you're training your network, you're deciding how the weights are adjusted.The activation runs through the network until it reaches the output nodes. The output nodes then give us the information in a way that we can understand. Your network will use a cost function to compare the output and the actual expected output. The model

performance is evaluated by the cost function. It's expressed as the difference between the actual value and the predicted value. There are many different cost functions you can use, you're looking at what the error you have in your network is. You're working to minimize loss function. (In essence, the lower the loss function, the closer it is to your desired output). The information goes back, and the neural network begins to learn with the goal of minimizing the cost function by tweaking the weights. This process is called **backpropagation**.

In **forward propagation**, information is entered into the input layer and propagates forward through the network to get our output values. We compare the values to our expected results. Next, we calculate the errors and propagate the info backward. This allows us to train the network and update the weights. (Backpropagation allows us to adjust all the weights simultaneously.) During this process, because of the way the algorithm is structured, you're able to adjust all of the weights simultaneously. This allows you to see which part of the error each of your weights in the neural network is responsible for.

When you've adjusted the weights to the optimal level, you're ready to proceed to the testing phase .

How does an artificial neural network learn ?

There are two different approaches to get a program to do what you want. First, there's the specifically guided and hard-programmed approach. You tell the program exactly what you want it to do. Then there are **neural networks**. In neural networks, you tell your network the inputs and what you want for the outputs, and then you let it learn on its own.
By allowing the network to learn on its own, you can avoid the necessity of entering in all of the rules. You can create the architecture and then let it go and learn. Once it's trained up, you can give it a new image and it will be able to distinguish output.

Feedforward and feedback networks

- A **feedforward network** is a network that contains inputs, outputs, and hidden layers. The signals can only travel in one direction (forward). Input data passes into a layer where calculations are performed. Each processing element computes based upon the weighted sum of its inputs. The new values become the new input values that feed the next layer (feed-forward). This continues through all the layers and determines the output. Feedforward networks are often used in, for example, data mining.

- A **feedback network** (for example, a recurrent neural network) has feedback paths. This means that they can have signals traveling in both directions using loops. All possible connections between neurons are allowed. Since loops are present in this type of network, it becomes a non-linear dynamic system which changes continuously until it reaches a state of equilibrium. Feedback networks are often used in optimization problems where the network looks for the best arrangement of interconnected factors.

What is a weighted sum ?

Inputs to a neuron can either be features from a training set or outputs from the neurons of a previous layer. Each connection between two neurons has a unique synapse with a unique weight attached. If you want to get from one neuron to the next, you have to travel along the synapse and pay the "toll" (weight). The neuron then applies an activation function to the sum of the weighted inputs from each incoming synapse. It passes the result on to all the neurons in the next layer. When we talk about updating weights in a network, we're talking about adjusting the weights on these synapses.

A neuron's input is the sum of weighted outputs from all the neurons in the previous layer. Each input is multiplied by the weight associated with the synapse connecting the input to the current neuron. If there are 3 inputs or neurons in the previous layer, each neuron in the current layer will have 3 distinct weights: one for each synapse.

In a nutshell, the activation function of a node defines the output of that node.

The activation function (or transfer function) translates the input signals to output signals. It maps the output values on a range like 0 to 1 or -1 to 1. It's an abstraction that represents the rate of action potential firing in the cell. It's a number that represents the likelihood that the cell will fire. At its simplest, the function is binary: yes (the neuron fires) or no (the neuron doesn't fire). The output can be either 0 or 1 (on/off or yes/no), or it can be anywhere in a range.
If you were using a function that maps a range between 0 and 1 to determine the likelihood that an image is a cat, for example, an output of 0.9 would show a 90% probability that your image is, in fact, a cat.

What is an activation function?
In a nutshell, the activation function of a node defines the output of that node.
Return to **chapter** (**2.2**).

How do you adjust the weights ?

You could use a brute force approach to adjust the weights and test thousands of different combinations. But even with the most simple neural network that has only five input values and a single hidden layer, you'll wind up with 10^{75} possible combinations.
Running this on the world's fastest supercomputer would take longer than the universe has existed so far.

Enter gradient descent

But if you go with gradient descent, you can look at the angle of the slope of the weights and find out if it's positive or negative in order to continue to slope downhill to find the best weights on your quest to reach the global minimum.

If you go with gradient descent, you can look at the angle of the slope of the weights and find out if it's positive or negative. This allows you to continue to slope downhill to find the best weights on your quest to reach the global minimum.

Gradient descent is an algorithm for finding the minimum of a function. The analogy you'll see over and over is that of someone stuck on top of a mountain and trying to get down (find the minima). There's heavy fog making it impossible to see the path, so she uses gradient descent to get down to the bottom of the mountain. She looks at the steepness of the hill where she is and proceeds down in the direction of the steepest descent. You should assume that the steepness isn't immediately obvious. Luckily, she has a tool that can measure steepness!..Unfortunately, this tool takes forever.
She wants to use it as infrequently as she can to get down the mountain before dark. The real difficulty is choosing how often she wants to use her tool so she doesn't go off track.

In this analogy, the person is the algorithm. The steepness of the hill is the slope of the error surface at that point. The direction she goes is the gradient of the error surface at that point. The tool she's using is differentiation (the slope of the error surface can be calculated by taking the derivative of the squared error function at that point). The rate at which she travels before taking another measurement is the learning rate of the algorithm. It's not a perfect analogy, but it gives you a good sense of what gradient descent is all about. The machine is learning the gradient, or direction, that the model should take to reduce errors.

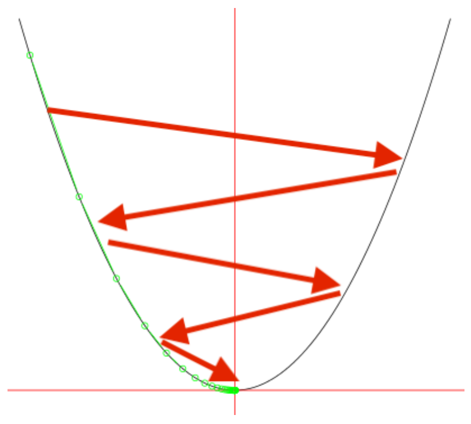


Figure 14: Gradient descent (simplified).

Gradient descent requires the cost function to be convex, but what if it isn't?

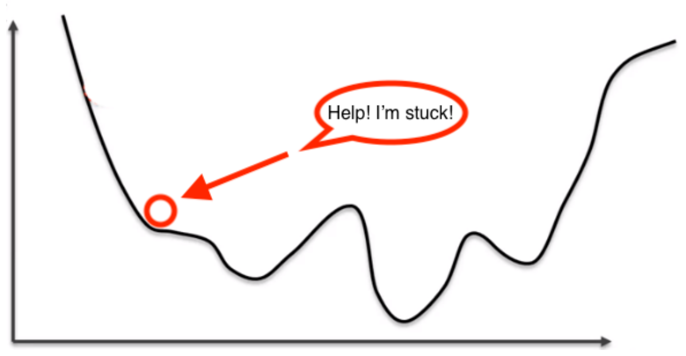


Figure 15: Normal Gradient Descent limit.

Normal gradient descent will get stuck at a local minimum rather than a global minimum, resulting in a subpar network. In normal gradient descent, we take all our rows and plug them into the same neural network, take a look at the weights, and then adjust them. This is called batch gradient descent. In s**tochastic gradient descent**, we take the rows one by one, run the neural network, look at the cost functions, adjust the weights, and then move to the next row. Essentially, you're adjusting the weights for each row.

Stochastic gradient descent has much higher fluctuations, which allows you to find the global minimum. It's called "stochastic" because samples are shuffled randomly, instead of as a single group or as they appear in the training set. It looks like it might be slower, but it's actually faster because it doesn't have to load all the data into memory and wait while the data is all run together.

The main pro for batch gradient descent is that it's a deterministic algorithm.
This means that if you have the same starting weights, every time you run the network you will get the same results. Stochastic gradient descent is always working at random. (You can also run mini-batch gradient descent where you set a number of rows, run that many rows at a time, and then update your weights.)

Many improvements on the basic stochastic gradient descent algorithm have been proposed and used, including implicit updates (ISGD), momentum method, averaged stochastic gradient descent, adaptive gradient algorithm (AdaGrad), root mean square propagation (RMSProp), adaptive moment estimation (Adam), and more.

So here's a quick walkthrough of training an artificial neural network with stochastic gradient descent:

1. Randomly initiate weights to small numbers close to 0
2. Input the first observation of your dataset into the input layer, with each feature in one input node.

3. **Forward propagation** — from left to right, the neurons are activated in a way that each neuron's activation is limited by the weights. You propagate the activations until you get the predicted result.
4. Compare the predicted result to the actual result and measure the generated error.
5. **Backpropagation** — from right to left, the error is back propagated. The weights are updated according to how much they are responsible for the error. (The learning rate decides how much we update the weights.)
6. **Reinforcement learning** (repeat steps 1–5 and update the weights after each observation) or batch learning (repeat steps 1–5, but update the weights only after a batch of observations).
7. When the whole training set has passed through the ANN, that is one epoch. Repeat with more epochs.

II.4.2 Deep learning for visual recognition

The field of computer vision is shifting from statistical methods to deep learning neural network methods.
There are still many challenging problems to solve in computer vision. Nevertheless, deep learning methods are achieving state-of-the-art results on some specific problems.
It is not just the performance of deep learning models on benchmark problems that is most interesting; it is the fact that a single model can learn meaning from images and performs vision tasks, obviating the need for a pipeline of specialized and hand-crafted methods.
In this chapter, we will discover nine interesting computer vision tasks where deep learning methods are achieving some headway [13].
Each example provides a description of the problem, an example, and references to papers that demonstrate the methods and results.

Overview

Some of the computer vision problems where deep learning has been used:

A. Image Classification
B. Image Classification With Localization
C. Object Detection
D. Object Segmentation
E. Image Style Transfer
F. Image Colorization
G. Image Reconstruction
H. Image Super-Resolution
I. Image Synthesis
J. Other Problems

A. Image Classification

Image classification involves assigning a label to an entire image or photograph.
This problem is also referred to as "object classification" and perhaps more generally as "image recognition," although this latter task may apply to a much broader set of tasks related to classifying the content of images.

Some examples of image classification include:

1. Labeling an x-ray as cancer or not (binary classification).
2. Classifying a handwritten digit (multiclass classification).
3. Assigning a name to a photograph of a face (multiclass classification).

A popular example of image classification used as a benchmark problem is the MNIST dataset [14].
A popular real-world version of classifying photos of digits is The Street View House Numbers (SVHN) dataset [15].
For state-of-the-art results and relevant papers on these and other image classification tasks, see: what is the class of this image [16].

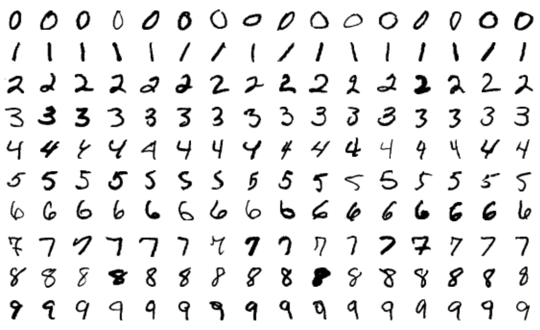


Figure 16:Example of Handwritten Digits From the MNIST Dataset.

B. Image Classification With Localization

Image classification with localization involves assigning a class label to an image and showing the location of the object in the image by a bounding box (drawing a box around the object).
This is a more challenging version of image classification.
Some examples of image classification with localization include:

1. Labeling an x-ray as cancer or not and drawing a box around the cancerous region.
2. Classifying photographs of animals and drawing a box around the animal in each scene.

A classical dataset for image classification with localization is the PASCAL
Visual Object Classes datasets [17] , or PASCAL VOC for short (e.g. VOC 2012) [18]. These are datasets used in computer vision challenges over many years.

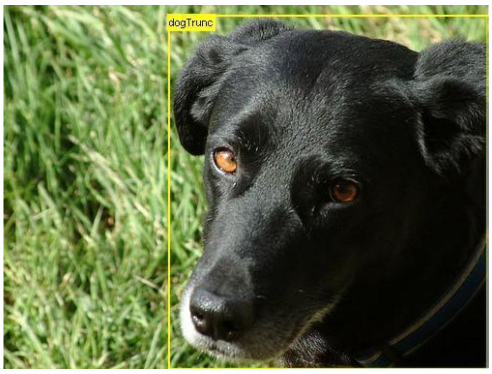


Figure 17: Example of Image Classification With Localization of a Dog from VOC 2012.

The task may involve adding bounding boxes around multiple examples of the same object in the image. As such, this task may sometimes be referred to as "object detection."

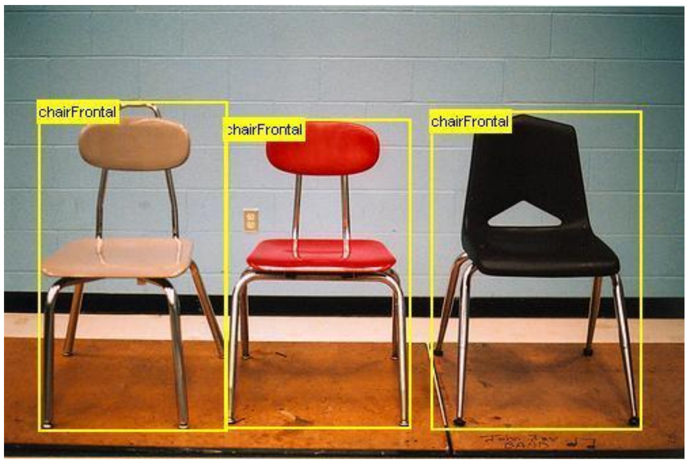


Figure 18: Example of Image Classification With Localization of Multiple Chairs From VOC 2012.

C. Object Detection

Object detection is the task of image classification with localization, although an image may contain multiple objects that require localization and classification.
This is a more challenging task than simple image classification or image classification with localization, as often there are multiple objects in the image of different types.
Often, techniques developed for image classification with localization are used and demonstrated for object detection. Some examples of object detection include:

1. Drawing a bounding box and labeling each object in a street scene.
2. Drawing a bounding box and labeling each object in an indoor photograph.
3. Drawing a bounding box and labeling each object in a landscape.

The PASCAL Visual Object Classes datasets, or PASCAL VOC for short (e.g. VOC 2012), is a common dataset for object detection [17][18].

Another dataset for multiple computer vision tasks is Microsoft's Common Objects in Context Dataset, often referred to as MS COCO [19] .

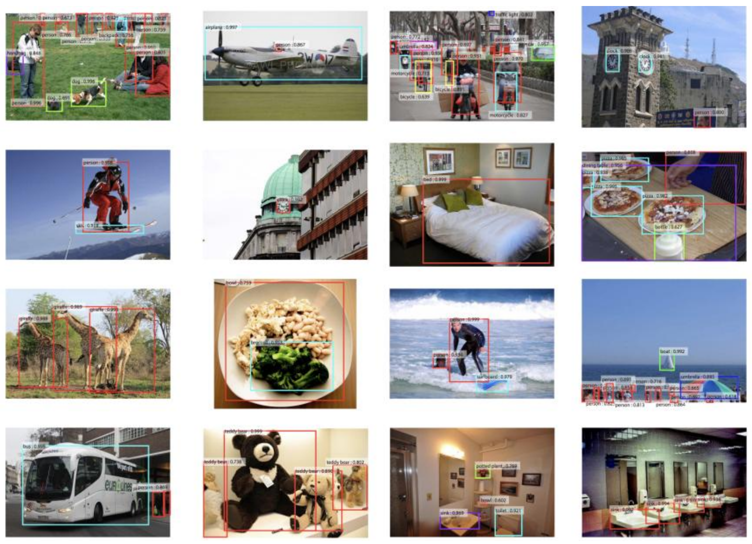


Figure 19: Example of Object Detection With Faster R-CNN on the MS COCO Dataset

D. Object Segmentation

Object segmentation, or semantic segmentation, is the task of object detection where a line is drawn around each object detected in the image. Image segmentation is a more general problem of spitting an image into segments.
Object detection is also sometimes referred to as object segmentation.

Unlike object detection that involves using a bounding box to identify objects, object segmentation identifies the specific pixels in the image that belong to the object. It is like a fine-grained localization.

More generally, "image segmentation" might refer to segmenting all pixels in an image into different categories of objects.
Again, the VOC 2012 and MS COCO datasets can be used for object segmentation [18][19].

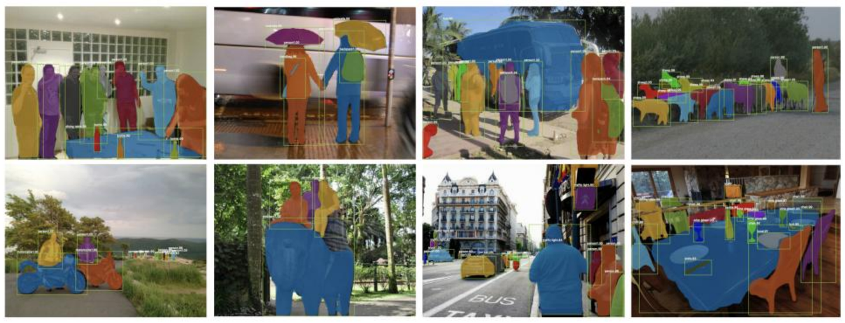


Figure 20: Example of Object Segmentation on the COCO Dataset Taken from "Mask R-CNN".

E. Style Transfer

Style transfer or neural style transfer is the task of learning style from one or more images and applying that style to a new image.
This task can be thought of as a type of photo filter or transform that may not have an objective evaluation.
Examples include applying the style of specific famous artworks (e.g. by Pablo Picasso or Vincent van Gogh) to new photographs.
Datasets often involve using famous artworks that are in the public domain and photographs from standard computer vision datasets.
Some papers include:A Neural Algorithm of Artistic Style, 2015 [20] and Image Style Transfer Using Convolutional Neural Networks, 2016 [21] .

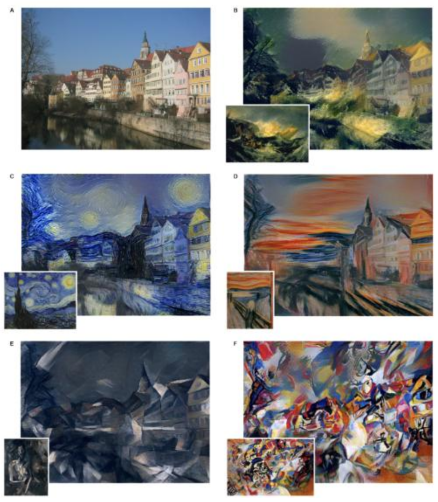


Figure 21: Example of Neural Style Transfer From Famous Artworks to a Photograph Taken from "A Neural Algorithm of Artistic Style".

F. Image Colorization

Image colorization or neural colorization involves converting a grayscale image to a full color image.
This task can be thought of as a type of photo filter or transform that may not have an objective evaluation.
Examples include colorizing old black and white photographs and movies.
Datasets often involve using existing photo datasets and creating grayscale versions of photos that models must learn to colorize.
Some papers include: Colorful Image Colorization, 2016 [22].

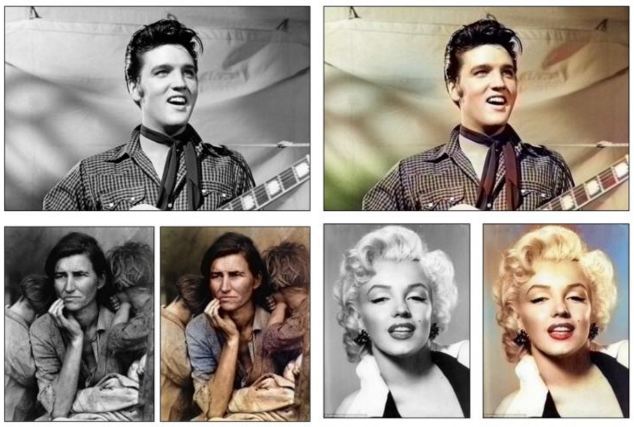


Figure 22: Examples of Photo ColorizationTaken from "Colorful Image Colorization".

G. Image Reconstruction

Image reconstruction and image inpainting is the task of filling in missing or corrupt parts of an image.
This task can be thought of as a type of photo filter or transform that may not have an objective evaluation.
Examples include reconstructing old, damaged black and white photographs and movies (e.g. photo restoration).
Datasets often involve using existing photo datasets and creating corrupted versions of photos that models must learn to repair.
Some papers include : Pixel Recurrent Neural Networks, 2016 [23].

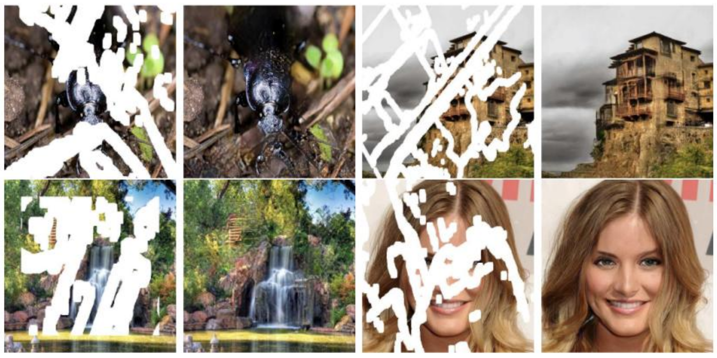


Figure 23: Example of Photo Inpainting.Taken from "Image Inpainting for Irregular Holes Using Partial Convolutions"".

H. Image Super-Resolution

Image super-resolution is the task of generating a new version of an image with a higher resolution and detail than the original image.
Often models developed for image super-resolution can be used for image restoration and inpainting as they solve related problems.

Datasets often involve using existing photo datasets and creating down-scaled versions of photos for which models must learn to create super-resolution versions.
Some papers include: Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network, 2017 [24] .

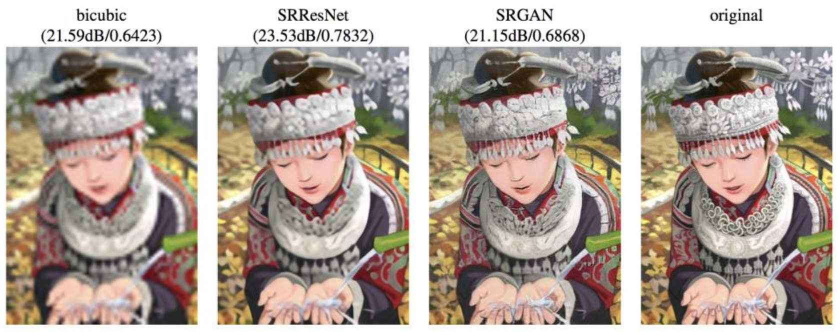


Figure 24: Example of the Results From Different Super-Resolution Techniques.Taken from "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network".

I. Image synthesis

Is the task of generating targeted modifications of existing images or entirely new images.
This is a very broad area that is rapidly advancing.
It may include small modifications of image and video (e.g. image-to-image translations), such as:

1. Changing the style of an object in a scene.
2. Adding an object to a scene.
3. Adding a face to a scene.

It may also include generating entirely new images, such as:

1. Generating faces.
2. Generating bathrooms.
3. Generating clothes.

Some papers include: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015 [25] .

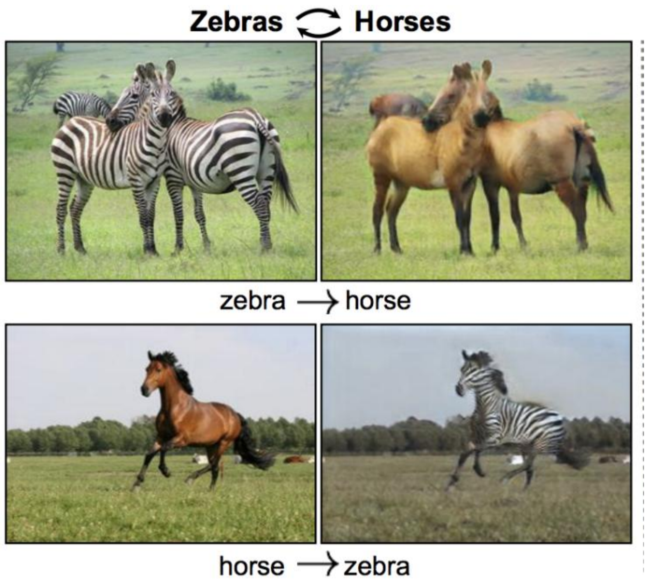


Figure 25: Example of Styling Zebras and Horses.Taken from "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks".

J. Other Problems

There are other important and interesting problems that I did not cover because they are not purely computer vision tasks.
Notable examples image to text and text to image:

1. Image Captioning: Generating a textual description of an image.
   Show and Tell: A Neural Image Caption Generator, 2014.
2. Image Describing: Generating a textual description of each object in an image.
   Deep Visual-Semantic Alignments for Generating Image Descriptions, 2015.
3. Text to Image: Synthesizing an image based on a textual description.
   AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks, 2017

II.5 Convolutional Neural Networks

The advancements in Computer Vision with Deep Learning has been constructed and perfected with time, primarily over one particular algorithm — a Convolutional Neural Network.

There is a strong resurging interest in neural-network-based learning because of its superior performance in many speech and image/video understanding applications nowadays. The recent success of deep neural networks (DNN) [1] is due to the availability of a large amount of labeled training data (e.g. the ImageNet) and more efficient computing hardware. It is called deep learning since we often observe performance improvement when adding more layers.
The resulting networks and extracted features are called deep networks and deep features, respectively. There are two common neural network architectures: the convolutional neural

networks (CNNs) [2] and the recurrent neural networks (RNNs). CNNs are used to recognize visual patterns directly from pixel images with variability. RNNs are designed to recognize patterns in time series composed of symbols or audio/speech waveforms. Both CNNs and RNNs are special types of multilayer neural networks. They are trained with the back-propagation algorithm. We will focus on CNNs in this work.

Although deep learning tends to outperform classical pattern recognition methods experimentally, its superior performance is somehow difficult to explain. Without a good understanding of deep learning, we can only have a set of empirical rules and intuitions. There has been a large amount of recent efforts devoted to the understanding of CNNs. Examples include scattering networks [3, 4, 5], tensor analysis [6], generative modeling [7], relevance propagation [8], Taylor decomposition [9], etc. Another popular topic along this line is on the visualization of filter responses at various layers [10, 11, 12]. It is worthwhile to point out that the CNN is a special form of the feedforward neural network (FNN), also known as the multi-layer perceptron (MLP), trained with back-propagation. It was proved in [13] that FNNs are capable of approximating any measurable function to any desired accuracy. In short, FNNs are universal approximators. The success of CNNs in various applications today is a reflection of the universal approximation capability of FNNs. Despite this theoretical foundation, the internal operational mechanism of CNNs remains mysterious.

II.5.1 Definition

A **Convolutional Neural Network** (**ConvNet/CNN**) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

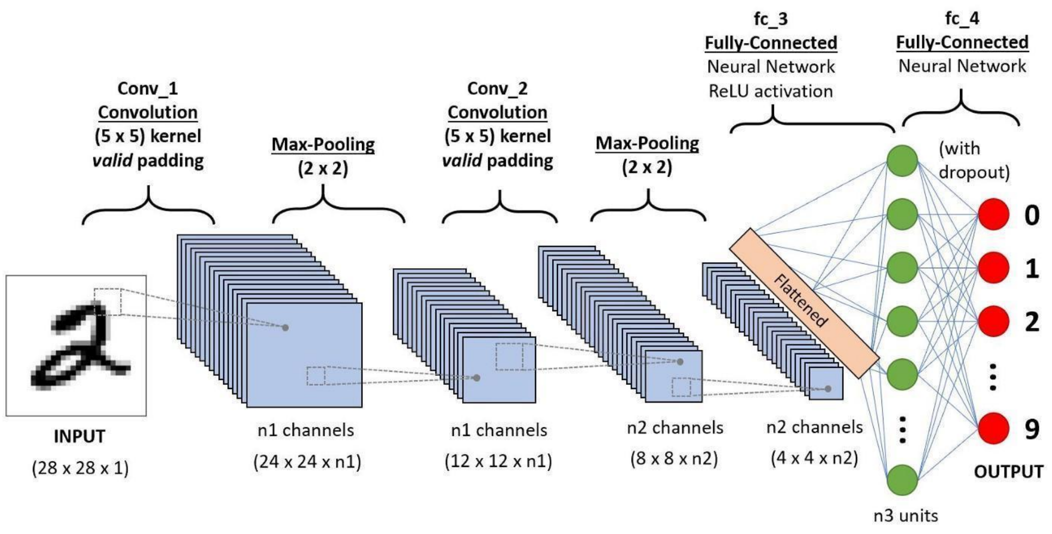


Figure 26: A CNN sequence to classify handwritten digits.

A **ConvNet** is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

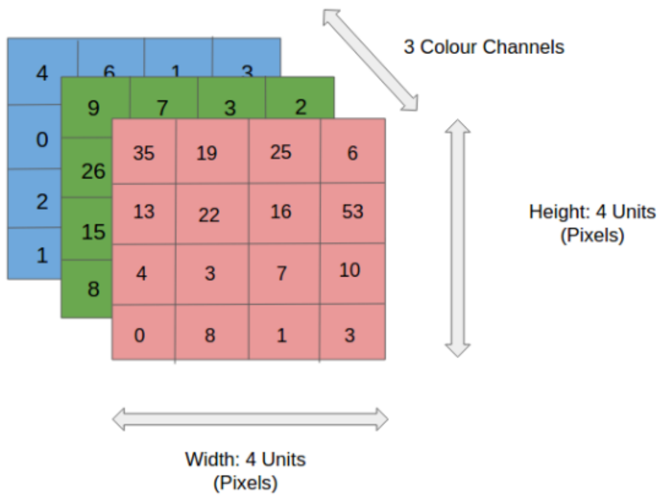


Figure 27: 4x4x3 RGB Image

In the figure, we have an RGB image which has been separated by its three color planes — Red, Green, and Blue. There are a number of such color spaces in which images exist — Grayscale, RGB, HSV, CMYK, etc.You can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320).

The role of the **ConvNet** is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.

II.5.2 Model architecture

Convolution Layer — The Kernel

Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB),
In the above demonstration, the green section resembles our 5x5x1 input image, I. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K, represented in the color yellow. We have selected K as a 3x3x1 matrix.

Kernel/Filter, K =

1 0 1

0 1 0

1 0 1

The Kernel shifts 9 times because of Stride Length = 1 (Non-Strided), every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering.
The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

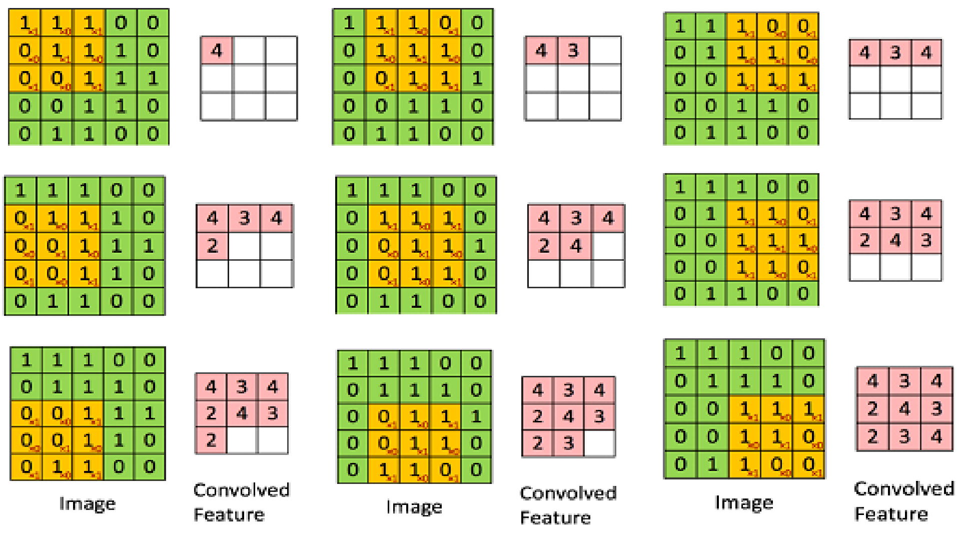


Figure 28: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

In the case of images with multiple channels (e.g. RGB), the Kernel has the same depth as that of the input image. Matrix Multiplication is performed between Kn and In stack ([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.

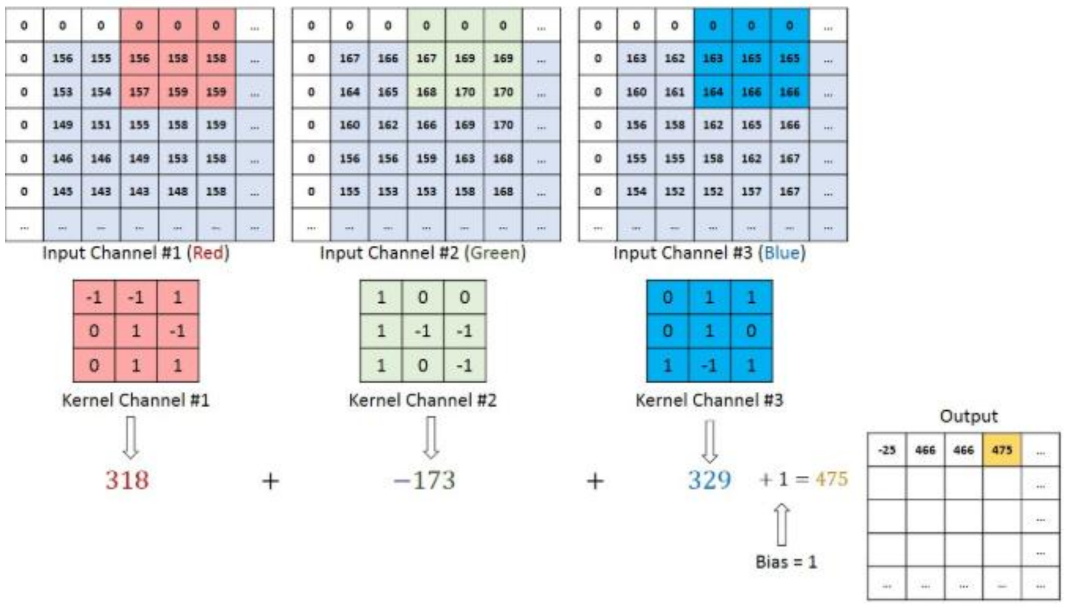


Figure 29: Convolution operation on a MxNx3 image matrix with a 3x3x3 Kernel

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. **ConvNets** need not be limited to only one Convolutional Layer. Conventionally, the first **ConvLayer** is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

Why Nonlinear Activation?

Generally speaking, CNNs attempt to learn the relationship between the input and the output and store the learned experience in their filter weights. One challenge to understand CNNs is the role played by the nonlinear activation unit after the convolutional operation. We will drop the pooling operation in the discussion below since it mainly provides a spatial dimension reduction technique and its role is not as critical. The adoption of nonlinear activation in neural networks can be dated back to the early work of McCulloch and Pitts [16], where the output of the nonlinear activation function is set to 1 or ´1 if the input value is positive or non-positive, respectively. A geometrical interpretation of the McCullochPitts neural model was given in [17].

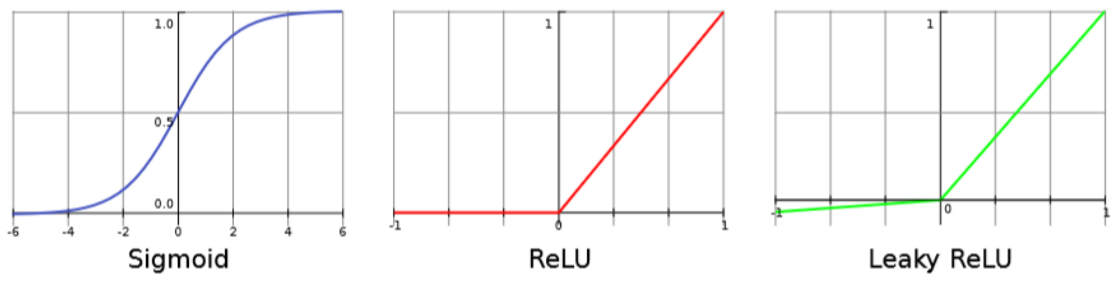


Figure 30: Three nonlinear activation functions adopted by CNNs: the sigmoid function (left), the ReLU (middle) and the Leaky ReLU (right).

In the recent literature, three activation functions are commonly used by CNNs. They are the sigmoid function, the rectified linear unit (ReLU) and the parameterized ReLU (PReLU) as shown in Fig. 2. The PReLU is also known as the leaky ReLU. All of them play a clipping-like operation. The sigmoid clips the input into an interval between 0 and 1. The ReLU clips negative values to zero while keeping positive values unchanged. The leaky ReLU has a role similar to the ReLU but it maps larger negative values to smaller ones by reducing the slope of the mapping function. It is observed experimentally that, if the nonlinear operation is removed, the system performance drops by a large margin. Each convolutional layer is specified by its filter weights which are determined in the training stage by an iterative update process. That is, they are first initialized and then adjusted by backpropagation to minimize a cost function. All weights are then fixed in the testing stage. These weights play the role of "system memory".

Pooling Layer

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training the model.

There are two types of Pooling: **Max Pooling** and **Average Pooling**. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.

Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that Max Pooling performs a lot better than Average Pooling.

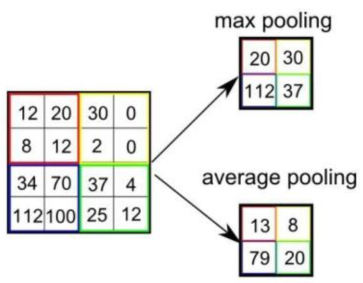


Figure 31: Types of Pooling

The Convolutional Layer and the Pooling Layer, together form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power.

After going through the above process, we have successfully enabled the model to understand the features. Moving on, we are going to flatten the final output and feed it to a regular Neural Network for classification purposes.

Fully Connected Layer (FC Layer)

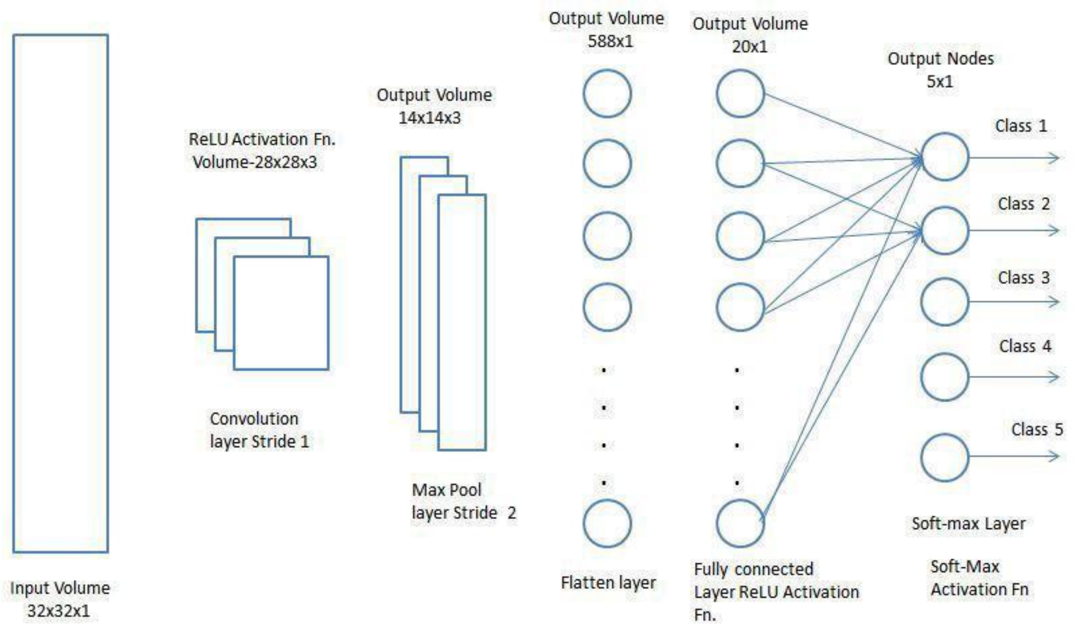


Figure 32 :Fully Connected Layer

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the **Softmax Classification** technique.

There are various architectures of CNNs available which have been key in building algorithms which power and shall power AI as a whole in the foreseeable future. Some of them have been listed below :

1. LeNet [14] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–232
2. AlexNet
3. VGGNet
4. GoogLeNet
5. ResNet
6. ZFNet

II.5.3 3D CNN - vs - Recurrent Neural Networks

On the one hand, Recurrent Neural Networks (RNN) can be defined as Neural Networks with memory. The idea behind RNNs is to make use of sequential information. In a traditional neural network, we assume that all inputs and outputs are independent of each other.

The working method of a RNN can be seen as if you add a hidden short time memory between layers of Neural Network. In this way, with the next item of the sequence, you do not only feed the network with it but also with the result of the previous item so now the training takes into account the relation between both.
RNNs shine when working with speech or text recognition because they are not limited to a fixed number of inputs in a sequence.

On the other hand, knowing that we can fix the number of frames per sequence, we have the 3D-CNN which are, as its names indicates, a 3-dimensional version of ordinary CNN where the Kernels and the input and output volumes have on extra dimension (time dimension) and the convolution is performed through width, height and time.
This fact makes 3D-CNN really well-suited for spatiotemporal feature learning. Compared to ordinary CNN, 3D-CNN has the ability to model temporal information better owing to 3D convolution and 3D pooling operations. Despite the fact that ordinary Convolutional Layers could also work with video sequences (if you treat the frames as channels) they would give an image as output so you would lose the temporal information. In 3D-CNN the channels and the temporal axis are treated separately meaning that with a volume input you give a volume output, avoiding to lose the temporal information.

CHAPTER III: Problem statement and working algorithm

Gesture recognition has been a very interesting problem in the Computer Vision community for a long time. This is particularly due to the fact that segmentation of foreground objects from a cluttered background is a challenging problem in real-time. The most obvious reason is because of the semantic gap involved when a human looks at an image and a computer looking at the same image. Humans can easily figure out what's in an image but for a computer, images are just 3-dimensional matrices. It is because of this, computer vision problems remain a challenge. Look at the image below.

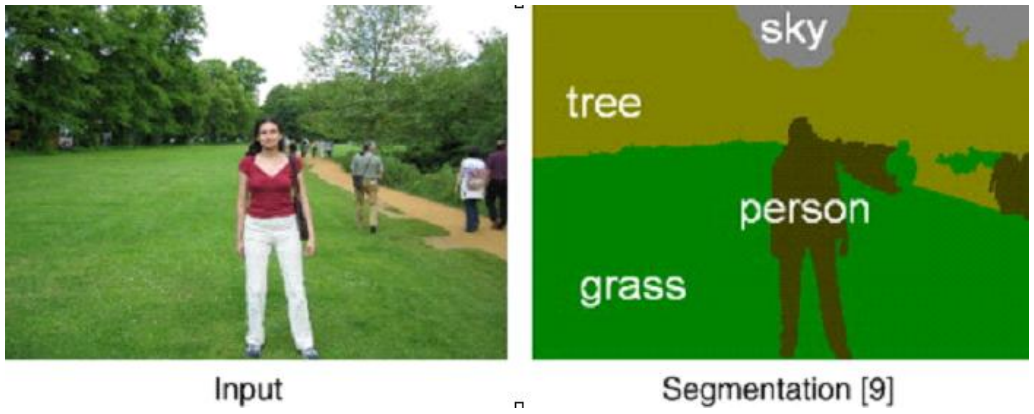


Figure 33 : Presentation of a semantic segmentation problem.

This image describes the semantic segmentation problem where the objective is to find different regions in an image and tag its corresponding labels. In this case, "sky", "person", "tree" and "grass". A quick Google search will give you the necessary links to learn more about this research topic. As this is a very difficult problem to solve, we will restrict our focus to nicely segment one foreground object from a live video sequence.

III .1 Problem statement

We are going to recognize hand gestures from a video sequence. To recognize these gestures from a live video sequence, we first need to take out the hand region alone removing all the unwanted portions in the video sequence. After segmenting the hand region, we then count

the fingers shown in the video sequence instruct a robot based on the finger count. Thus, the entire problem could be solved using 2 simple steps :

1.Find and segment the hand region from the video sequence.

2.Count the number of fingers from the segmented hand region in the video sequence.

III .2 Image Recognition Algorithms

One type of image recognition algorithm is an image classifier. It takes an image (or part of an image) as an input and predicts what the image contains. The output is a class label, such as dog, cat or table . In our case the output can be one of the 24 letters of the alphabet determined by the hand's gesture. The algorithm needs to be trained to learn and distinguish between classes.

In a simple case, to create a classification algorithm that can identify images with dogs, you'll train a neural network with thousands of images of the letter alphabet, and thousands of images of backgrounds without dogs. The algorithm will learn to extract the features that identify a "dog" object and correctly classify images that contain dogs. While most image recognition algorithms are classifiers, other algorithms can be used to perform more complex activities. For example, a Recurrent Neural Network can be used to automatically write captions describing the content of an image.

Once training images are prepared, you'll need a system that can process them and use them to make a prediction on new, unknown images. That system is an artificial neural network. Neural network image recognition algorithms can classify just about anything, from text to images, audio files, and videos (see our in-depth article on classification and neural networks).

CHAPTER III: Problem statement and working algorithm

Image Data Pre-Processing Steps for Neural Networks

Neural network image recognition algorithms [37] rely on the quality of the dataset – the images used to train and test the model. Here are a few important parameters and considerations for image data preparation.

1. Image size—higher quality images give the model more information but require more neural network nodes and more computing power to process.
2. The number of images—the more data you feed to a model, the more accurate it will be, but ensure the training set represents the real population.
3. The number of channels—grayscale images have 2 channels (black and white) and color images typically have 3 color channels (Red, Green, Blue / RGB), with colors represented in the range [0,255].
4. Aspect ratio—ensure the images have the same aspect ratio and size. Typically, neural network models assume a square shape input image.
5. Image scaling—once all images are squared you can scale each image. There are many up-scaling and down-scaling techniques, which are available as functions in deep learning libraries.
6. Mean, standard deviation of input data—you can look at the 'mean image' by calculating the mean values for each pixel, in all training examples, to obtain information on the underlying structure in the images.
7. Normalizing image inputs—ensures that all input parameters (pixels in this case) have a uniform data distribution. This makes convergence speedier when you train the network. You can conduct data normalization by subtracting the mean from each pixel and then dividing the outcome by the standard deviation.
8. Dimensionality reduction—you can decide to collapse the RGB channels into a gray-scale channel. You may want to reduce other dimensions if you intend to make the neural network invariant to that dimension or to make training less computationally intensive.
9. Data augmentation—involves augmenting the existing data-set, with perturbed types of current images, including scaling and rotating. You do this to expose the neural

network to a variety of variations. This way this neural network is less likely to identify unwanted characteristics in the data-set.

III .2.1 Convolutional Neural Networks and Their Role in Image Recognition

In a Convolutional Neural Network (CNN) the neurons in one layer don’t connect to all the neurons in the next layer. Rather, a convolutional neural network uses a three-dimensional structure, where each set of neurons analyzes a specific region or “feature” of the image. CNNs filters connections by proximity (pixels are only analyzed in relation to pixels nearby), making the training process computationally achievable. In a CNN each group of neurons focuses on one part of the image. For example, in a hand image, one group of neurons might identify the fingers, another the palm, etc. There may be several stages of segmentation in which the neural network image recognition algorithm analyzes smaller parts of the images, for example, within the fingers, the hand’s index,thumb , middle, etc. The final output is a vector of probabilities, which predicts, for each feature in the image, how likely it is to belong to a class or category.

III .2.2 Effectiveness and Limitations of Convolutional Neural Networks

A CNN architecture makes it possible to predict objects and faces in images using industry benchmark datasets with up to 95% accuracy, greater than human capabilities which stand at 94% accuracy. Even so, convolutional neural networks have their limitations:

1. **Require high processing power**. Models are typically trained on high-cost machines with specialized Graphical Processing Units (GPUs).
2. **Can fail when images are rotated or tilted**, or when an image has the features of the desired object, but not in the correct order or position, for example, a face with the nose and mouth switched around. A new architecture called CAPSNet has emerged to address this limitation.

III .4 Some algo for special procedures and functions

To understand classification with neural networks, it’s essential to learn how other classification algorithms work, and their unique strengths. For many problems, a neural network may be unsuitable or “overkill”. For others, it might be the only solution.

A. Logistic Regression

Classifier type : Binary

How It Works : Analyzes a set of data points with one or more independent variables (input variables, which may affect the outcome) and finds the best fitting model to describe the data points, using the logistic regression equation:

$$Y_i = \frac{1}{1-e^{-X_i'\beta}} + \varepsilon_i \text{ where } -\infty < x < \infty , y = 0,1$$

Strengths : Simple to implement and understand, very effective for problems in which the set of input variables is well known and closely correlated with the outcome.

Weaknesses : Less effective when some of the input variables are not known, or when there are complex relationships between the input variables.

B. Decision Tree Algorithm

Classifier type : Multiclass

How it works : Uses a tree structure with a set of “if-then” rules to classify data points. The rules are learned sequentially from the training data. The tree is constructed top-down; attributes at the top of the tree have a larger impact on the classification decision. The training process continues until it meets a termination condition.

Strengths : Able to model complex decision processes, very intuitive interpretation of results.

Weaknesses : Can very easily overfit the data, by over-growing a tree with branches that reflect outliers in the data set. A way to deal with overfitting is pruning the model, either by preventing it from growing superfluous branches (pre-pruning), or removing them after the tree is grown (post-pruning).

C. Random Forest Algorithm

Classifier type *:* Multiclass

How it works : A more advanced version of the decision tree, which addresses overfitting by growing a large number of trees with random variations, then selecting and aggregating the best-performing decision trees. The "forest" is an ensemble of decision trees, typically done using a technique called "bagging".

Strengths : Provides the strengths of the decision tree algorithm, and is very effective at preventing overfitting and thus much more accurate, even compared to a decision tree with extensive manual pruning.

Weaknesses : Not intuitive, difficult to understand why the model generates a specific outcome.

D. Naive Bayes Classifier

Classifier type : Multiclass

How it works : A probability-based classifier based on the Bayes algorithm. According to the concept of dependent probability, it calculates the probability that each of the features of a data point (the input variables) exists in each of the target classes. It then selects the category for which the probabilities are maximal. The model is based on an assumption (which is often not true) that the features are conditionally independent.

Strengths : *Simple to implement and computationally light—the algorithm is linear and does not involve iterative calculations. Although its assumptions are not valid in most cases, Naive Bayes is surprisingly accurate for a large set of problems, scalable to very large data sets, and is used for many NLP models. Can also be used to construct multi-layer decision trees, with a Bayes classifier at every node.*

Weaknesses : Very sensitive to the set of categories selected, which must be exhaustive. Problems where categories may be overlapping or there are unknown categories can dramatically reduce accuracy.

E. k-Nearest Neighbor (KNN)

Classifier type : Multiclass

How it works : Classifies each data point by analyzing its nearest neighbors from the training set. The current data point is assigned the class most commonly found among its neighbors. The algorithm is non-parametric (makes no assumptions on the underlying data) and uses lazy learning (does not pre-train, all training data is used during classification).

Strengths : Very simple to implement and understand, and highly effective for many classification problems, especially with low dimensionality (small number of features or input variables).

Weaknesses : KNN's accuracy is not comparable to supervised learning methods. Not suitable for high dimensionality problems. Computationally intensive, especially with a large training set.

F. Artificial Neural Networks and Deep Neural Networks

Classifier type : Either binary or multiclass

How it works : Artificial neural networks are built of simple elements called neurons, which take in a real value, multiply it by a weight, and run it through a non-linear activation function. By constructing multiple layers of neurons, each of which receives part of the input variables, and then passes on its results to the next layers, the network can learn very complex functions. Theoretically, a neural network is capable of learning the shape of just any function, given enough computational power.

Strengths : Very effective for high dimensionality problems, able to deal with complex relations between variables, non-exhaustive category sets and complex functions relating input to output variables. Powerful tuning options to prevent over- and under-fitting.

Weaknesses *:*Theoretically complex, difficult to implement (although deep learning frameworks are readily available that do the work for you). Non-intuitive and requires expertise to tune. In some cases a large training set is required to be effective.

CHAPITRE IV: Results and Discussion

IV.1 System Design

IV.1.1 System Architecture

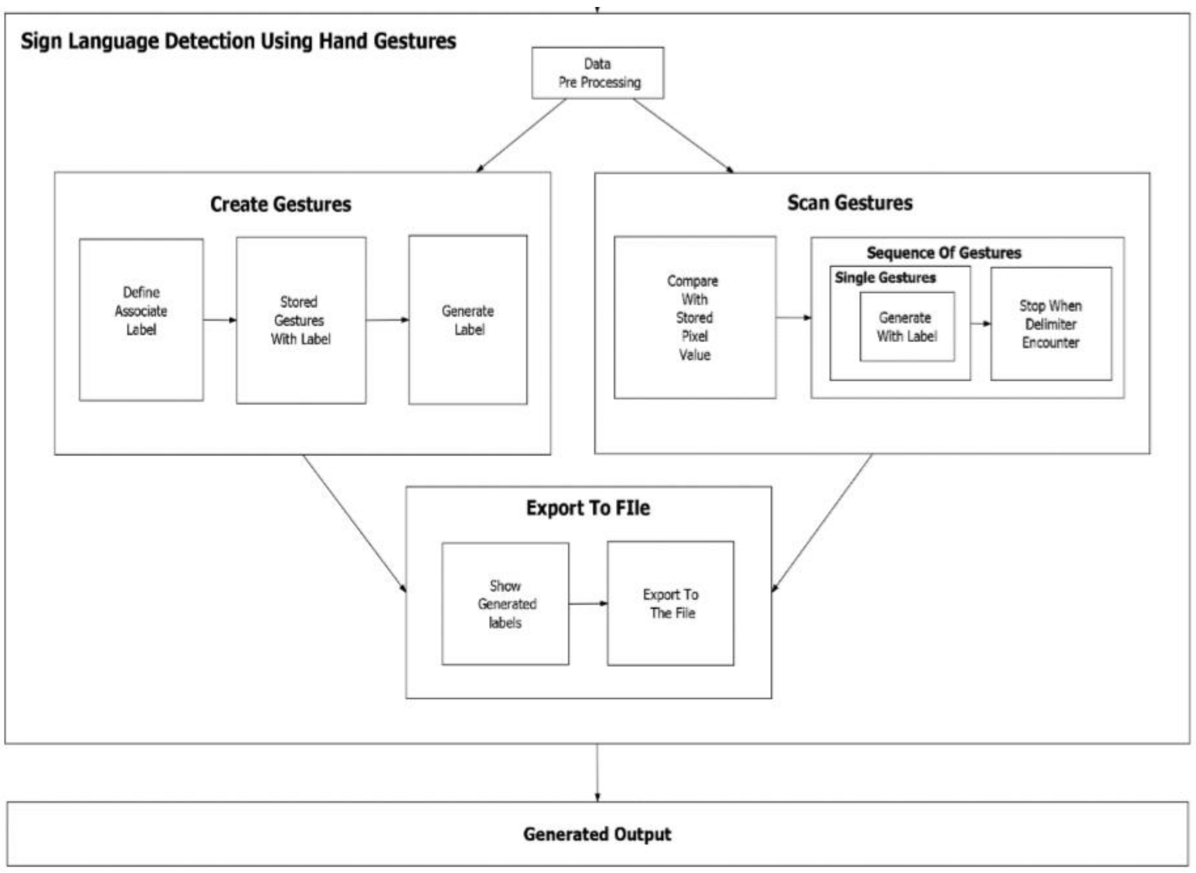


Figure 34 : System Architecture presentation.

IV.1.2 Diagram

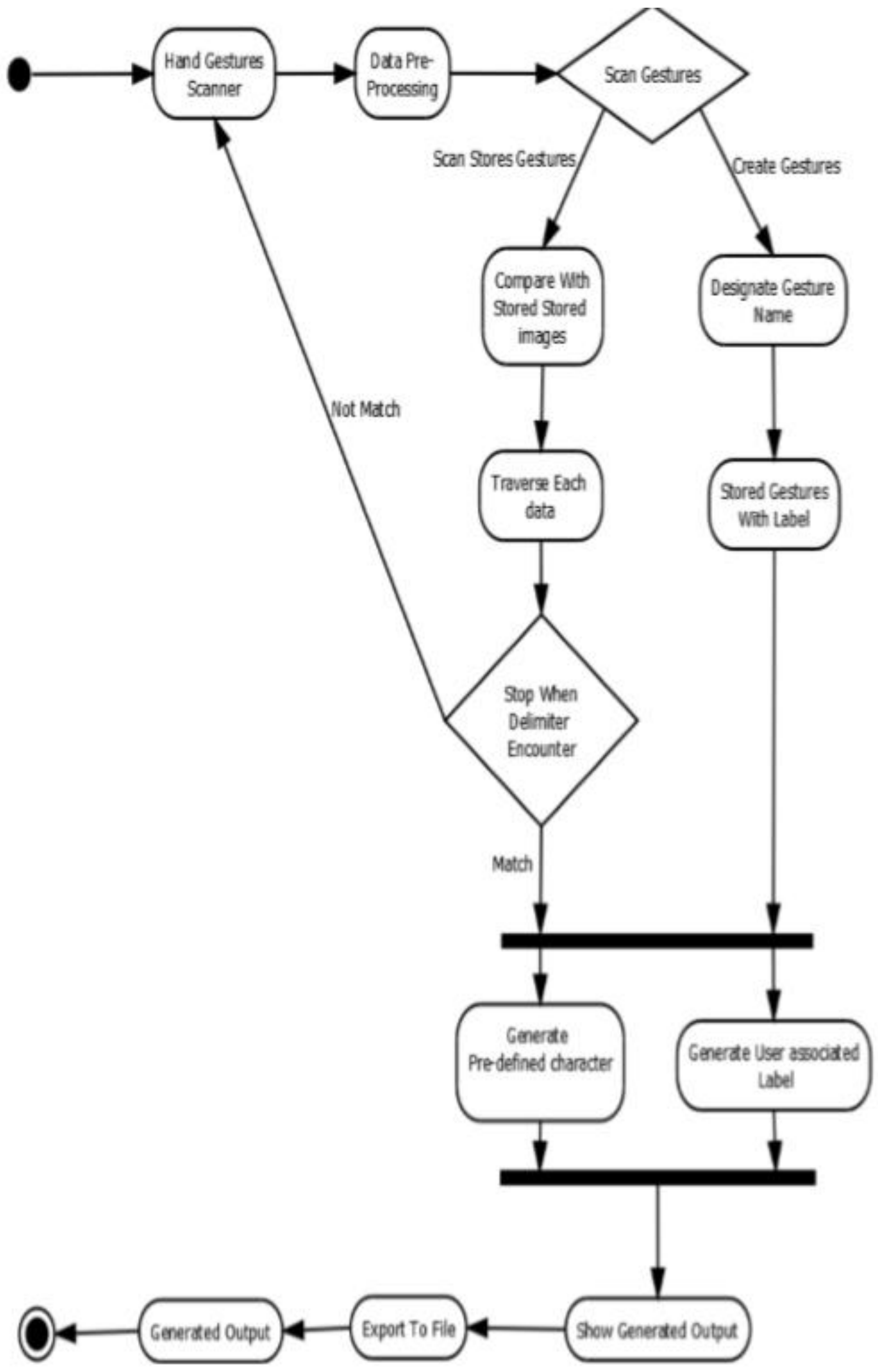


Figure 35 : Hand gesture detection diagram.

IV.2 Implementation

IV.1.1 Overview to RPI platforme

Raspberry Pi is one of the popular single-board computers of our generation. All the major image processing and computer vision algorithms and operations can be implemented easily with **OpenCV** on **Raspberry P**i. and **Python 3 as** a programming language, to create and design some computer vision projects.The Raspberry Pi 3 Model B is the earliest model of the third-generation Raspberry Pi. It replaced the Raspberry Pi 2 Model B in February 2016.

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet
- 40-pin extended GPIO
- 4 USB 2 ports
- 4 Pole stereo output and composite video port
- Full size HDMI
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- Micro SD port for loading your operating system and storing data
- Upgraded switched Micro USB power source up to 2.5AAll current models of Raspberry Pi have a port for connecting the Camera Module.

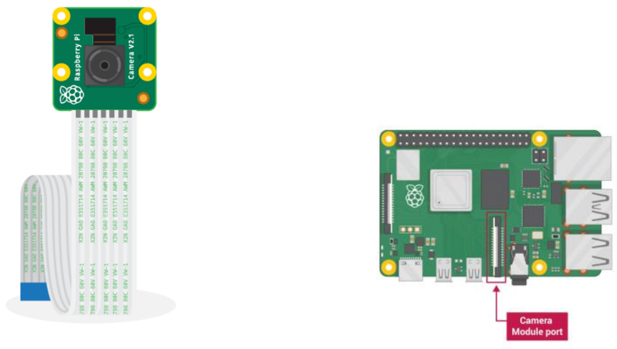


Figure 36 :Raspberry Pi platforme and Camera Module

Figure 37 :Raspberry Pi3 model B with camera

IV.1.2 MNIST dataset

The original MNIST (Modified National Institute of Standards and Technology database) image dataset of handwritten digits is a popular benchmark for image-based machine learning methods but researchers have renewed efforts to update it and develop drop-in replacements that are more challenging for computer vision and original for real-world applications. As noted in one recent replacement called the Fashion-MNIST dataset, the Zalando researchers quoted the startling claim that "Most pairs of MNIST digits (784 total pixels per sample) can be distinguished pretty well by just one pixel". To stimulate the community to develop more drop-in replacements, the Sign Language MNIST is presented here and follows the same CSV format with labels and pixel values in single rows. The American Sign Language letter database of hand gestures represents a multi-class problem with 24 classes of letters (excluding J and Z which require motion).

The dataset format is patterned to match closely with the classic MNIST. Each training and test case represents a label (0-25) as a one-to-one map for each alphabetic letter A-Z (and no cases for 9=J or 25=Z because of gesture motions). The training data (27,455 cases) and test data (7172 cases) are approximately half the size of the standard MNIST but otherwise similar with a header row of label, pixel1,pixel2....pixel784 which represent a single 28x28 pixel image with grayscale values between 0-255. The original hand gesture image data represented multiple users repeating the gesture against different backgrounds. The Sign Language MNIST data came from greatly extending the small number (1704) of the color images included as not cropped around the hand region of interest. To create new data, an image pipeline was used based on ImageMagick and included cropping to hands-only, gray-scaling, resizing, and then creating at least 50+ variations to enlarge the quantity. The modification and expansion strategy was filters ('Mitchell', 'Robidoux', 'Catrom', 'Spline', 'Hermite'), along with 5% random pixelation, +/- 15% brightness/contrast, and finally 3 degrees rotation. Because of the tiny size of the images, these modifications effectively alter the resolution and class separation in interesting, controllable ways.

IV.1.3 Data Collection & Pre-Processing

Using the Sign Language MNIST dataset from Kaggle, we evaluated models to classify hand gestures for each letter of the alphabet. Due to the motion involved in the letters J and Z, these letters were not included in the dataset. However, the data includes approximately 35,000 28x28 pixel images of the remaining 24 letters of the alphabet. Similar to the original MNIST hand drawn images, the data contains an array of grayscale values for the 784 pixels in each image. One of these images is shown below.
**35,000 image of all the letters (the remaining 24), each letter was represented by 28*28 pixel image (784 pixels).

IV.2 Results and Discussion

IV.3.1 Presentation of GUI

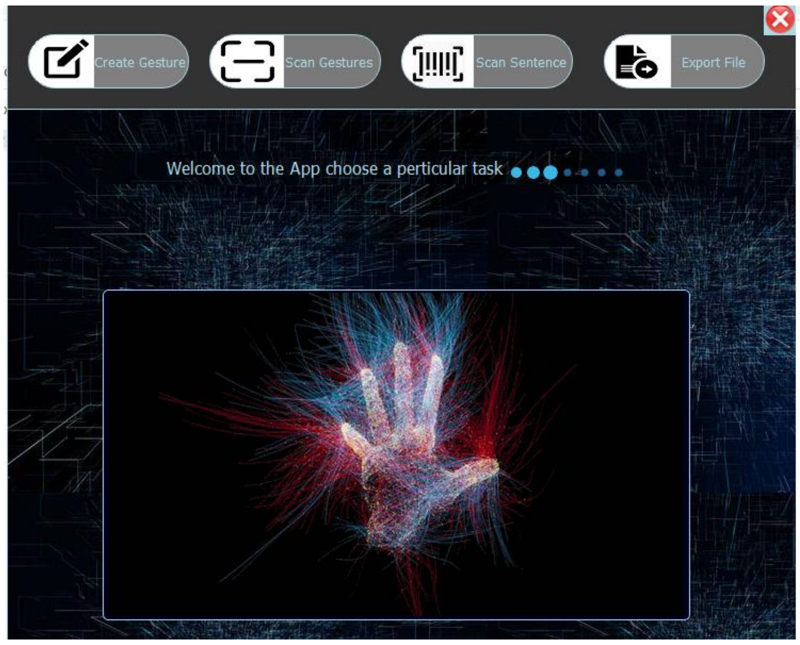


Figure 38 : Dashboard :welcome

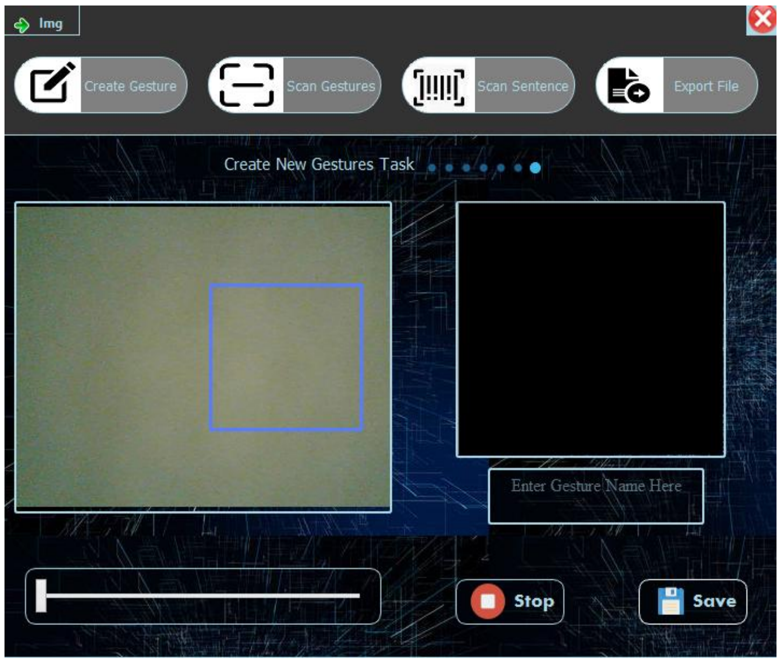


Figure 39 : Task 1 , create new gesture

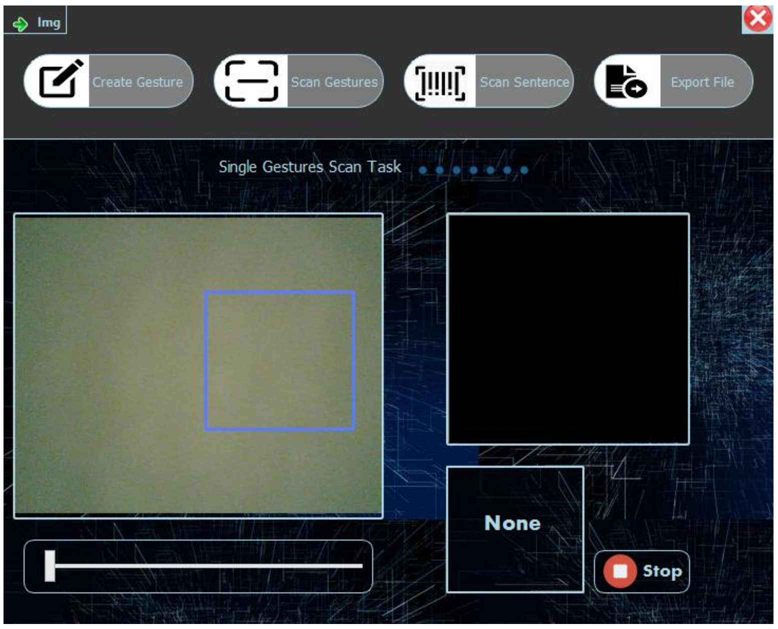


Figure 40 ::Task2 ,scan gesture

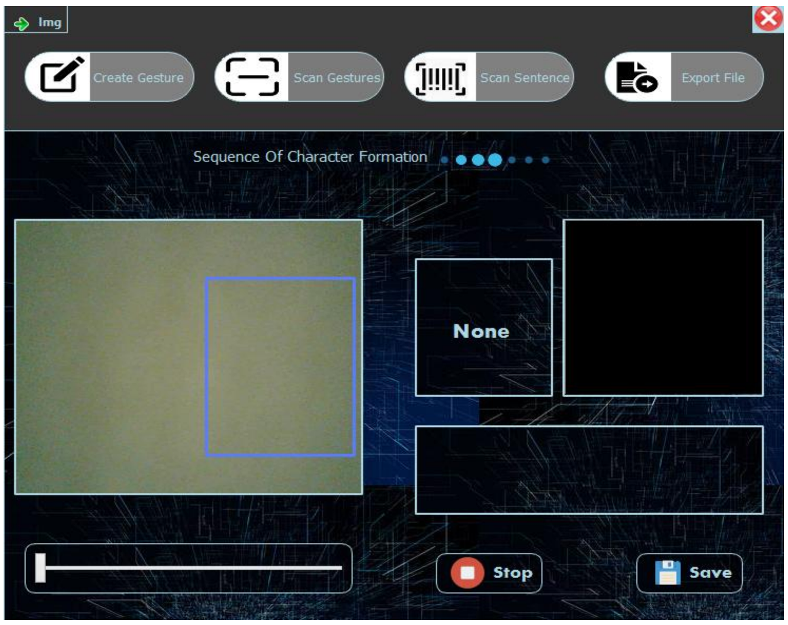


Figure 41 :Task3 ,create sequence of characters

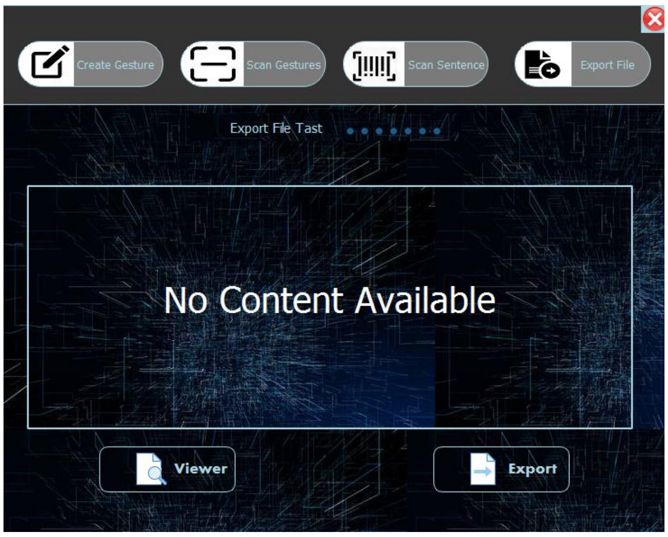


Figure 42 : Task4,Export file

IV.3.2 Notes

Performance

The proposed model for the still images is able to identify the static signs with an accuracy of 94.33%. Based on our analysis of the dynamic signs.

The training and validation datasets used to build and optimize the model contained 80% of the original data. The remaining 20% (~7,000 samples) was reserved for model testing. When this test data was input to the model, it achieved 94.33% accuracy. To further understand the strengths and weaknesses of this model, we created a confusion matrix.

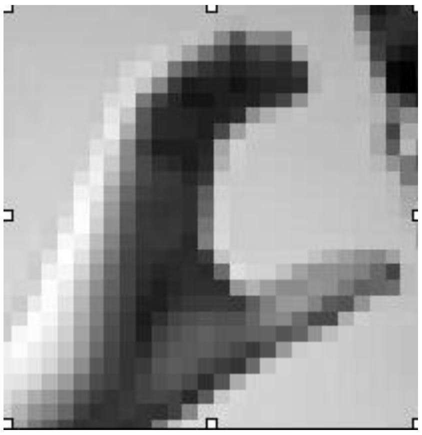


Figure 43 : Sample image of the letter "C" from the training dataset.

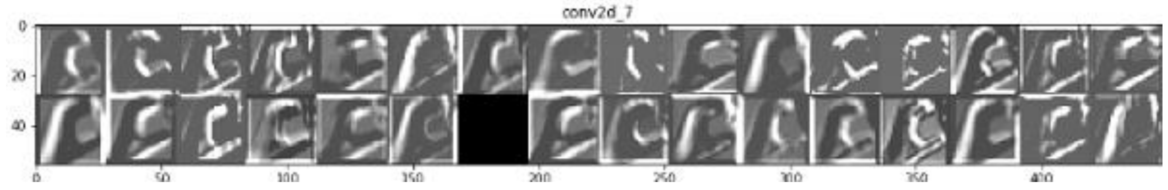


Figure 44 : Outputs from the first hidden layer.

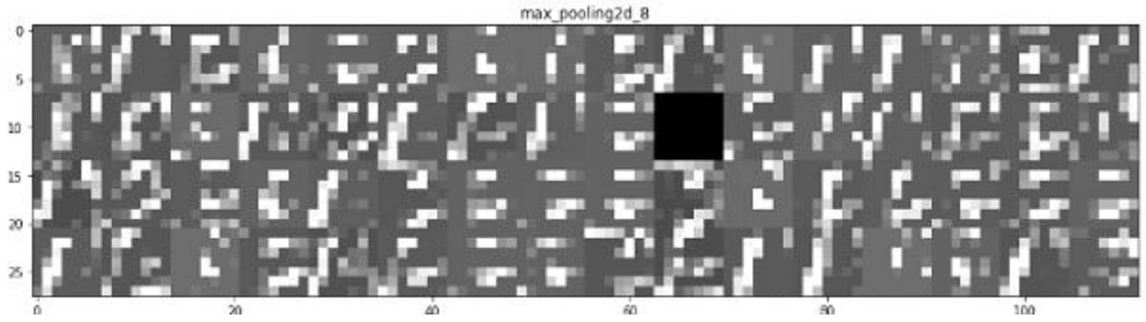
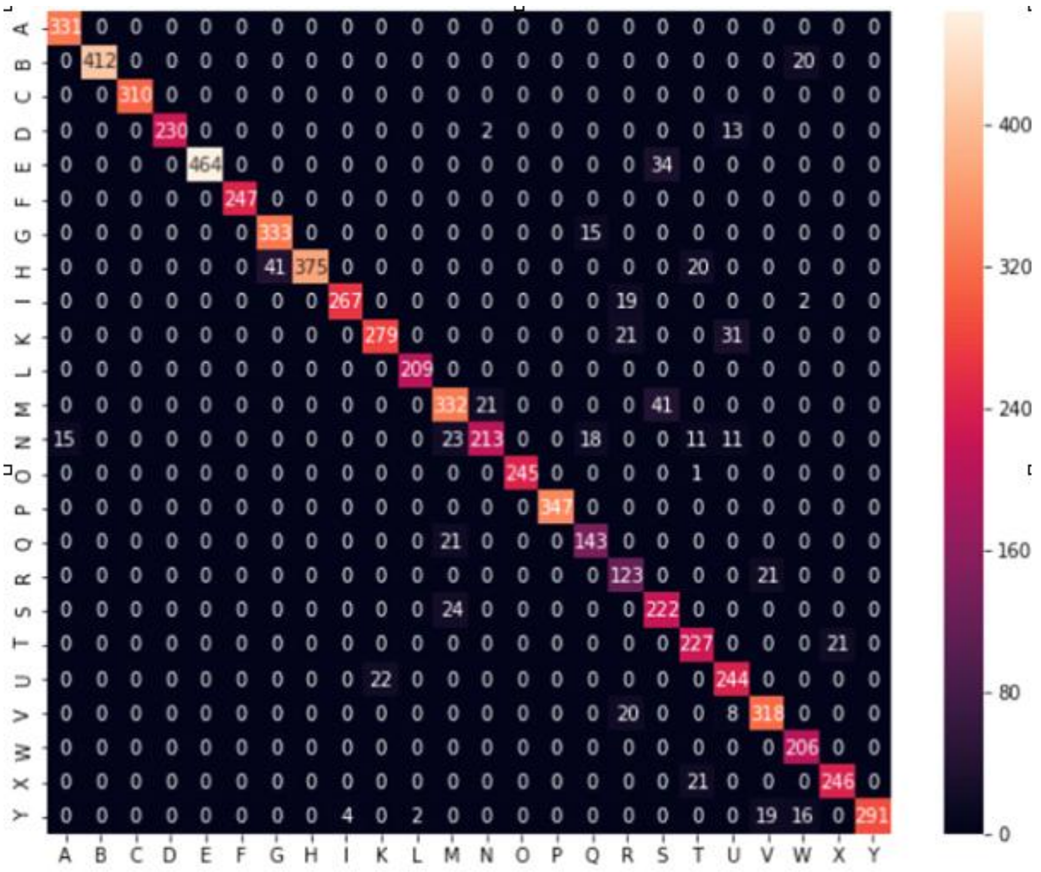


Figure 45 :: Outputs from the fourth hidden layer.



	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
A	331	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	412	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0
C	0	0	310	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	230	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	13	0	0	0	0
E	0	0	0	0	464	0	0	0	0	0	0	0	0	0	0	0	0	34	0	0	0	0	0	0
F	0	0	0	0	0	247	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	333	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	41	375	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0
I	0	0	0	0	0	0	0	0	267	0	0	0	0	0	0	0	19	0	0	0	0	2	0	0
K	0	0	0	0	0	0	0	0	0	279	0	0	0	0	0	0	21	0	0	31	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	209	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	332	21	0	0	0	0	41	0	0	0	0	0	0
N	15	0	0	0	0	0	0	0	0	0	0	23	213	0	0	18	0	0	11	11	0	0	0	0
O	0	0	0	0	0	0	0	0	0	0	0	0	0	245	0	0	0	0	1	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	347	0	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0	21	0	0	0	143	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	123	0	0	0	21	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	24	0	0	0	0	0	222	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	227	0	0	0	21	0
U	0	0	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	0	0	244	0	0	0	0
V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	8	318	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	206	0	0
X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21	0	0	0	246	0
Y	0	0	0	0	0	0	0	0	4	0	2	0	0	0	0	0	0	0	0	0	19	16	0	291

Figure 46 : Confusion matrix of the CNN's output of the test data.

Minor Drawbacks of CNN

1. A Convolutional neural network is significantly slower due to an operation such as maxpool.
2. If the CNN has several layers then the training process takes a lot of time if the computer doesn't consist of a good GPU.
3. A ConvNet requires a large Dataset to process and train the neural network.
4. If the images contain some degree of tilt or rotation the CNNs usually have difficulty in classifying the image.
5. CNNs do not have coordinate frames which are a basic component of human vision.

V.1 Conclusion

From this project we have tried to overshadow some of the major problems faced by the disabled persons in terms of talking. We found out the root cause of why they can't express themselves more freely. The result that we got was the other side of the audience is not able to interpret what these persons are trying to say or what is the message that they want to convey.

Thereby this application serves the person who wants to learn and talk in sign languages. With this application a person will quickly adapt various gestures and their meaning as per **ASL standards**. They can quickly learn what alphabet is assigned to which gesture. Add-on to this custom gesture facility is also provided along with sentence formation. A user need not be a literate person if they know the action of the gesture, they can quickly form the gesture and the appropriate assigned character will be shown onto the screen.

Concerning the implementation, we have used the **TensorFlow** framework, with **keras API**. And for the user feasibility complete front-end is designed using **PyQT5**. Appropriate user-friendly messages are prompted as per the user actions along with what gesture means which character window. Additionally, an export to file module is also provided with TTS (Text-To-Speech) assistance meaning whatever the sentence was formed a user will be able to listen to it and then quickly export along with observing what gesture he/she made during the sentence formation.

V.1 Future Work

1. It can be integrated with various search engines and texting applications such as google, WhatsApp. So that even the illiterate people could be able to chat with other persons, or query something from the web just with the help of gestures.

2. This project is working on image current; further development can lead to detecting the motion of video sequence and assigning it to a meaningful sentence with TTS assistance.

Webography

[1] Shobhit Agarwal, "What are some problems faced by deaf and dumb people while using todays common tech like phones and PCs", 2017 [Online]. Available: https://www.quora.com/What-are-some-problems-faced-by-deaf-and-dumb-people-whileusing-todays-common-tech-like-phones-and-PCs, [Accessed April 06, 2019].

[2] NIDCD, "american sign language", 2017 [Online]. Available:

https://www.nidcd.nih.gov/health/american-sign-language, [Accessed April 06, 2019].

[3] Suharjito MT, "Sign Language Recognition Application Systems for Deaf-Mute People A Review Based on Input-Process-Output", 2017 [Online]. Available: https://www.academia.edu/35314119/Sign_Language_Recognition_Application_Syste ms_for_Deaf-Mute_People_A_Review_Based_on_Input-Process-Output [Accessed April 06, 2019].

[4] Sanil Jain and K.V.Sameer Raja, "Indian Sign Language Character Recognition" , [Online]. Available: https://cse.iitk.ac.in/users/cs365/2015/_submissions/vinsam/report.pdf [Accessed April 06, 2019].

[5]https://www.researchgate.net/publication/220693385_Introduction_to_Artificial_Intelligence
[6] https://dl.acm.org/doi/10.5555/1074100.1074138

[7] https://www.amazon.com/Artificial-Intelligence-Elaine-Rich/dp/0070522634

[8] https://towardsdatascience.com/an-overview-of-computer-vision-1f75c2ab1b66

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

[10] https://www.saedsayad.com/artificial_neural_network.htm

[11]https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

[12] https://iamtrask.github.io/2015/07/27/python-network-part2/

[13] https://ml.berkeley.edu/blog/2017/02/04/tutorial-3/

[14] https://cs231n.github.io/optimization-2/

[15] http://neuralnetworksanddeeplearning.com/chap2.html

[16] https://ruder.io/optimizing-gradient-descent/index.html

[17] https://en.wikipedia.org/wiki/Arthur_Samuel

[18] https://en.wikipedia.org/wiki/Tom_M._Mitchell

[19] https://www.coursera.org/learn/machine-learning/lecture/zcAuT/welcome-to-machine-learning

[20] https://www.edureka.co/blog/what-is-machine-learning/

[22] https://www.investopedia.com/terms/d/deep-learning.asp

[23] https://machinelearningmastery.com/applications-of-deep-learning-for-computer-vision/

[24]https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/

[25] http://ufldl.stanford.edu/housenumbers/

[26] http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

[27] http://host.robots.ox.ac.uk:8080/pascal/VOC/

[28] http://host.robots.ox.ac.uk/pascal/VOC/voc2012/

[29] http://cocodataset.org/

[30] https://arxiv.org/abs/1508.06576

[31] https://ieeexplore.ieee.org/document/7780634

[32] https://arxiv.org/abs/1603.08511

[33] https://arxiv.org/abs/1601.06759

[34] https://arxiv.org/abs/1609.04802

[35] https://arxiv.org/abs/1511.06434

[36] https://missinglink.ai/guides/computer-vision/neural-networks-image-recognition-methods-best-practices-applications/

[37] https://missinglink.ai/guides/neural-network-concepts/classification-neural-networks-neural-network-right-choice/

Bibliography

[21] Deep Learning By Ian Goodfellow, Yoshua Bengio, Aaron Courville

[36] Understanding Convolutional Neural Networks with A Mathematical Model

C.-C. Jay Kuo ,arXiv:1609.04112v2 [cs.CV] 2 Nov 2016

[37] B. H. Juang, Deep neural networks–a developmental perspective, APSIPA Transactions on Signal and Information Processing 5 (2016) e7.

[38] Y. LeCun, Y. Bengio, G. E. Hinton, Deep learning, Nature 521 (2015) 436–444.

[39] S. Mallat, Group invariant scattering, Communications on Pure and Applied Mathematics 65 (10) (2012) 1331–1398.

[40] J. Bruna, S. Mallat, Invariant scattering convolution networks, IEEE transactions on pattern analysis and machine intelligence 35 (8) (2013) 1872–1886.

[41] T. Wiatowski, H. Bolcskei, A mathematical theory of deep convolutional neural networks for feature extraction, arXiv preprint arXiv:1512.06293.

[42] N. Cohen, O. Sharir, A. Shashua, On the expressive power of deep learning: a tensor analysis, arXiv preprint arXiv:1509.05009 556.

[43] J. Dai, Y. Lu, Y.-N. Wu, Generative modeling of convolutional neural networks, arXiv preprint arXiv:1412.6296.

[44] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Mu¨ller, W. Samek, On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation, PloS one 10 (7) (2015) e0130140.

[45] G. Montavon, S. Bach, A. Binder, W. Samek, K.-R. Mu¨ller, Explaining nonlinear classification decisions with deep Taylor decomposition, arXiv preprint arXiv:1512.02479.

[46] K. Simonyan, A. Vedaldi, A. Zisserman, Deep inside convolutional networks: visualising image classification models and saliency maps, arXiv preprint arXiv:1312.6034.

[47] M. D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: European

Conference on Computer Vision, Springer, 2014, pp. 818–833.

[48] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, A. Torralba, Object detectors emerge in deep scene CNNs, arXiv preprint arXiv:1412.6856.

[49] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural networks 2 (5) (1989) 359–366.

Appendix

The whole program is divided into three modules :

1. **CNN model.py** : *Building the CNN structure.*
2. **Dashboard.py** : *The main application.*
3. **My_function.py** : *Contains all the necessary functions used.*

CNN model.py

```
# Part 1 - Building the CNN

# importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv3D
from keras.layers import MaxPool3D
from keras.layers import Flatten
from keras.layers import Dense, Dropout
from keras import optimizers

# Initialing the CNN
classifier = Sequential()

# Step 1 - The first layer: Convolution Layer
classifier.add(Conv3D(32,(3,3,3), input_shape=(64,64, 64, 3), activation='relu',border_mode='same'))
# Step 2 - The second layer:Pooling Layer
classifier.add(MaxPool3D(pool_size=(2, 2, 2),strides=(2, 2, 2)))

# Adding second convolution layer
classifier.add(Conv3D(32, (3,3,3), activation='relu'))
classifier.add(MaxPool3D(pool_size=(2, 2,2)))

# Adding 3rd Convolution Layer
classifier.add(Conv3D(64,(3,3,3), activation='relu'))
classifier.add(MaxPool3D(pool_size=(2, 2, 2)))

# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full Connection
# Add the hidden and the output layer
classifier.add(Dense(256, activation='relu'))
classifier.add(Dropout(0.5))
classifier.add(Dense(28, activation='softmax'))

# Compiling The CNN
classifier.compile(
        optimizer='adam',  # Stochastic gradient descent optimizer
```

```
          loss='categorical_crossentropy',
          metrics=['accuracy'])
# Part 2 Fitting the CNN to the image //Image Preprocessing//
# Import the class that allows us to use the specific function
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
     rescale=1./255,
     shear_range=0.2,
     zoom_range=0.2,
     horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)
# The train_set preprocessing section
training_set = train_datagen.flow_from_directory(
     'mydata/training_set',
     target_size=(64, 64,64),
     batch_size=128,
     class_mode='categorical')

# The test_set preprocessing section
test_set = test_datagen.flow_from_directory(
     'mydata/test_set',
     target_size=(64, 64,64),
     batch_size=128,
     class_mode='categorical')
# Fit our CNN to the test_set and also test its performance on the test_set
model = classifier.fit_generator(
     training_set,
     steps_per_epoch=800,
     epochs=25,
     validation_data=test_set,
     validation_steps=6500
    )
# Saving the model
import h5py
classifier.save('My_Trained_model2 .h5 ')
print(model.history.keys())
import matplotlib.pyplot as plt

# summarize history for accuracy

plt.plot(model.history['accuracy'])
plt.plot(model.history['val_accuracy'])
plt.title('model accuracy ')
plt.ylabel('accuracy ')  # show the error or skill of the model on the y-axis
plt.xlabel('epoch ')     # show the epochs along the x-axis as time
plt.legend(['train', 'test'], loc='down right')
plt.show()
```

```
# summarize history for loss
plt.plot(model.history['loss'])
plt.plot(model.history['val_loss'])
plt.title('model loss ')
plt.ylabel('loss ')
plt.xlabel('epoch ')
plt.legend(['train', 'test'], loc='upper left ')
plt.show()
```

Dashboard.py

```
from cv2.cv2 import VideoCapture
from predictor_function import *
from myFunction import *
# ------ PyQt is a GUI widgets toolkit---------
from PyQt5 import QtWidgets , uic
from PyQt5.QtWidgets import QMessageBox
from PyQt5.QtCore import QUrl
from PyQt5.QtGui import QImage
from PyQt5.QtGui import QPixmap
from PyQt5 import QtCore
from PyQt5.QtCore import QTimer , Qt
from PyQt5 import QtGui

# -------for gesture viewer--------####
# from scipy.ndimage import imread
# from matplotlib.pyplot import imread
from matplotlib import pyplot as plt
# from matplotlib.widgets import Button

# -----------------------------------------
from tkinter import filedialog , Tk  # for file export module
# from tkinter import *
# import tkinter as tk
import sys                              # for pyqt
import os                              # for files operations
import cv2                             # for the camera operations
import numpy as np                     # processing on images
# import qimage2ndarray                # converts images into matrix
# import win32api
import winGuiAuto
import win32gui
import win32con                    # for removing title cv2 window and always on top
import keyboard                    # for pressing keys
import pyttsx3                     # for tts assistance
import shutil

engine = pyttsx.init()             #engine initialization for audio tts assistance
def nothing(x):
```

```
plt.plot(model.history['loss'])
plt.plot(model.history['val_loss'])
plt.title('model loss ')
plt.ylabel('loss ')
plt.xlabel('epoch ')
plt.legend(['train', 'test'], loc='upper left ')
plt.show()
```

Dashboard.py

```
from cv2.cv2 import VideoCapture
from predictor_function import *
from myFunction import *
# ------ PyQt is a GUI widgets toolkit---------
from PyQt5 import QtWidgets , uic
from PyQt5.QtWidgets import QMessageBox
from PyQt5.QtCore import QUrl
from PyQt5.QtGui import QImage
from PyQt5.QtGui import QPixmap
from PyQt5 import QtCore
from PyQt5.QtCore import QTimer , Qt
from PyQt5 import QtGui

# -------for gesture viewer--------####
# from scipy.ndimage import imread
# from matplotlib.pyplot import imread
from matplotlib import pyplot as plt
# from matplotlib.widgets import Button

# -----------------------------------------
from tkinter import filedialog , Tk  # for file export module
# from tkinter import *
# import tkinter as tk
import sys                                # for pyqt
import os                                 # for files operations
import cv2                                # for the camera operations
import numpy as np                        # processing on images
# import qimage2ndarray                    # converts images into matrix
# import win32api
import winGuiAuto
import win32gui
import win32con                   # for removing title cv2 window and always on top
import keyboard                   # for pressing keys
import pyttsx3                    # for tts assistance
import shutil

engine = pyttsx.init()            #engine initialization for audio tts assistance
def nothing(x):
  pass
class Dashboard ( QtWidgets.QMainWindow ):
  def __init__(self):
     super ( Dashboard , self ).__init__ ( )
     self.cam = cv2.VideoCapture ( 0 )
     self.setWindowFlags (
        QtCore.Qt.WindowCloseButtonHint | QtCore.Qt.WindowMinimizeButtonHint |
QtCore.Qt.FramelessWindowHint )
     self.setWindowIcon ( QtGui.QIcon ( 'icons/click.png' ) )
     self.title = 'Sign language Recognition'
     uic.loadUi ( 'UI_Files/dash.ui' , self )
     self.setWindowTitle ( self.title )
     self.timer = QTimer ( )
     self.create.clicked.connect ( self.create_Gest )
     self.scan_sen.clicked.connect ( self.scanSent )
     self.exp.clicked.connect(self.exportFile)
     if self.scan_single.clicked.connect ( self.scan_Single ):
        self.timer.timeout.connect ( self.scan_Single )
     self.create.setCursor ( QtGui.QCursor ( QtCore.Qt.PointingHandCursor ) )
     self.scan_sen.setCursor ( QtGui.QCursor ( QtCore.Qt.PointingHandCursor ) )
     self.scan_single.setCursor ( QtGui.QCursor ( QtCore.Qt.PointingHandCursor ) )
     self.exp.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
     self.exit_button.clicked.connect (  self.quitApplication )
     self._layout = self.layout ( )
     self.label_3 = QtWidgets.QLabel ( )
     movie = QtGui.QMovie ( "icons/1.gif" )
     self.label_3.setMovie ( movie )
     self.label_3.setGeometry ( 95 , 75 , 900 , 700 )    # (x , y, width, hight)
     movie.start ( )
     self._layout.addWidget ( self.label_3 )
     self.setObjectName ( 'Message_Window' )
     self.label_4 = QtWidgets.QLabel ( )
     movie2 = QtGui.QMovie ( "icons/r.gif" )
     self.label_4.setMovie ( movie2 )
     self.label_4.setGeometry ( 470, 125 , 110 , 70 )
     movie2.start ( )
     self._layout.addWidget ( self.label_4 )

def quitApplication(self): ...
def create_Gest(self): ...
def exportFile(self): ...
def on_click(self): ...
def scanSent(self): ...
def scan_Single(self): ...

app = QtWidgets.QApplication ( [] )  # all user interface objects will inherit the QtWidgets class
# Built-in features: minimize, maximize, resize, move, close…

win = Dashboard ( )
```

```
win.show ( )
# start the application
sys.exit ( app.exec ( ) )
```

My_function.py

```
import cv2   # for the camera operations
import os    # for the files operations
import shutil
from PyQt5 import QtWidgets
from tkinter import filedialog, Tk

image_x, image_y = 64, 64  # image resolution
def capture_images(self, cam, save_img, mask):
  """Saves the images for custom gestures if button 'create gesture' is pressed in new gesture
generation through
  gui """
  cam.release()
  cv2.destroyAllWindows()
  if not os.path.exists('./SampleGestures'):
     os.mkdir('./SampleGestures')

  ges_name = save_img[-1]
  if len(ges_name) >= 1:
     img_name = "./SampleGestures/"+"{}.png".format(str(ges_name))
     save_img = cv2.resize(mask, (image_x, image_y))
     cv2.imwrite(img_name, save_img)
##################################################################
def checkFile():
  """retrieve the content of temp.txt for export module """
  check_file = os.path.isfile('temp.txt')
  if check_file:
     fr = open("temp.txt", "r")
     content = fr.read()
     fr.close()
  else:
     content = "No Content Available"
  return content

##################################################################
def controlTimer(self):
  # if timer is stopped
  self.timer.isActive()
  # create video capture
 cam = cv2.VideoCapture(0)
  # start timer
  self.timer.start(20)
##################################################################
def clearfunc(cam):
  """shut downs the opened camera and calls removeFile() Func"""
  cam.release()
  cv2.destroyAllWindows()
  removeFile()
def clearfunc2(cam):
  """shut downs the opened camera"""
  cam.release()
  cv2.destroyAllWindows()
#####################################################################
def removeFile():
  """Removes the temp.txt and temp_gest directory if any stop button is pressed or application is
closed"""
  try:
     os.remove("temp.txt")
  except:
     pass
  try:
     shutil.rmtree("TempGest")
  except:
     pass
#####################################################################
def saveBuff(self,cam,finalBuffer):
  """Save the file as temp.txt if save button is pressed in sentence formation through gui"""
  cam.release()
  cv2.destroyAllWindows()
  if len(finalBuffer) >= 1:
     f = open("temp.txt", "w")
     for i in finalBuffer:
        f.write(i)
     f.close()
#####################################################################
def openimg():
  """displays predefined gesture images at right most window"""
  cv2.namedWindow("Image", cv2.WINDOW_NORMAL)
  image = cv2.imread('all_signs_grey.png')
  cv2.imshow("Image", image)
  cv2.setWindowProperty("Image", cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)
   cv2.resizeWindow("Image", 298, 430)
  cv2.moveWindow("Image", 1052, 214)
#################################################################
def toggle_imagesfwd(event):
  """displays next images act as a gesture viewer"""
  img=load_images_from_folder('TempGest/')
  global index
  index += 1
  try:
     if index < len(img):
        plt.axes()
        plt.imshow(img[index])
        plt.draw()
  except:
    pass
################################################################
def toggle_imagesrev(event):
  """displays previous images act as a gesture viewer"""
  img=load_images_from_folder('TempGest/')
  global index
  index -= 1
  try:
    if index < len(img) and index>=0:
      plt.axes()
      plt.imshow(img[index])
      plt.draw()
  except:
    Pass
```