**HAND GESTURE CLASSIFICATION BY USING OPEN CV**

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| **SL.NO** | **TITLE** | **PAGE.NO** |
| **01.** | **ABSTRACT** |  |
| **02.** | **EXISTING SYSTEM**  2.1 DRAWBACKS |  |
| **03.** | **INTRODUCTION**  3.1 DATA SCIENCE  3.2 ARTIFICIAL INTELLIGENCE |  |
| **04.** | **MACHINE LEARNING** |  |
| **05.** | **DEEP LEARNING** |  |
| **06.** | **PREPARING THE DATASET** |  |
| **07.** | **PROPOSED SYSTEM**  7.1 ADVANTAGES |  |
| **08.** | **LITERATURE SURVEY** |  |
| **09.** | **SYSTEM STUDY**  9.1 PROJECT GOAL  9.2 OBJECTIVES  9.3 SCOPE OF THE PROJECT |  |
| **10.** | **OUTLINE OF THE PROJECT**  10.1 OVERVIEW OF THE SYSTEM |  |
| **11.** | **PROJECT REQUIREMENT**  11.1 FUNCTIONAL REQUIREMENT  11.2 NON-FUNCTIONAL REQUIREMENT |  |
| **12.** | **FEASIBILITY STUDY**  12.1 DATAFLOW DIAGRAM |  |
| **13.** | **SYSTEM ARCHITECTURE** |  |
| **14.** | **WORKFLOW DIAGRAM** |  |
| **15.** | **USECASE DIAGRAM** |  |
| **16.** | **CLASS DIAGRAM** |  |
| **17.** | **ACTIVITY DIAGRAM** |  |
| **18.** | **SEQUENCE DIAGRAM** |  |
| **19.** | **E.R – DIAGRAM** |  |
| **20.** | **COLLABORATION DIAGRAM** |  |
| **21.** | **ENVIRONMENT REQUIREMENT** |  |
| **22.** | **SOFTWARE DESCRIPTION**  22.1 ANACONDA NAVIGATOR  22.2 JUPYTER NOTEBOOK  22.3 PYCHARM |  |
| **23.** | **PYTHON** |  |
| **24.** | **METHODOLOGY** |  |
| **25.** | **ARTIFICIAL NEURAL NETWORK (ANN)** |  |
| **26.** | **ARCHITECTURE OF CNN** |  |
| **27.** | **TYPES OF CNN**  27.1 ALEX NET  27.2 LENET |  |
| **28.** | **LIST OF MODULES** |  |
| **29.** | **MODULE DESCRIPTION**  29.1 IMPORT THE GIVEN IMAGE FROM DATASET  29.2 TO TRAIN THE MODULE BY GIVEN IMAGE DATASET  29.3 WORKING PROCESS OF LAYERS IN CNN MODEL  29.4 CLASSIFICATION IDENTIFICATION |  |
| **31.** | **CODING** |  |
| **32.** | **CONCLUSION** |  |
| **33.** | **FUTURE WORK** |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **SL.NO** | **TITLE** | **PAGE.NO** |
| **1.** | **DATAFLOW DIAGRAM** |  |
| **2.** | **SYSTEM ARCHITECTURE** |  |
| **3.** | **WORKFLOW DIAGRAM** |  |
| **4.** | **USECASE DIAGRAM** |  |
| **5.** | **CLASS DIAGRAM** |  |
| **6.** | **ACTIVITY DIAGRAM** |  |
| **7.** | **SEQUENCE DIAGRAM** |  |
| **8.** | **E.R – DIAGRAM** |  |
| **9.** | **COLLABORATION DIAGRAM** |  |

**LIST OF SYSMBOLS**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NOTATION**  **NAME** | **NOTATION** | **DESCRIPTION** |
| 1. | Class | *+ public*  *-private*  *# protected*  *Class Name*  *-attribute*  *-attribute*  *+operation*  *+operation*  *+operation* | Represents a collection of similar entities grouped together. |
| 2. | Association | nAME  Class B  Class A    Class A  Class B | Associations represents static relationships between classes. Roles represents the way the two classes see each other. |
| 3. | Actor | Class A  Class A  Class B  Class B | It aggregates several classes into a single classes. |
| 4. | Aggregation | Interaction between the system and external environment |

|  |  |  |  |
| --- | --- | --- | --- |
| 5. | Relation  (uses) | uses | Used for additional process communication. |
| 6. | Relation  (extends) | EXTENDS | Extends relationship is used when one use case is similar to another use case but does a bit more. |
| 7. | Communication |  | Communication between various use cases. |
| 8. | State | State | State of the process. |
| 9. | Initial State |  | Initial state of the object |
| 10. | Final state |  | Final state of the object |
| 11. | Control flow |  | Represents various control flow between the states. |
| 12. | Decision box |  | Represents decision making process from a constraint |
| 13. | Usecase |  | Interact ion between the system and external environment. |

|  |  |  |  |
| --- | --- | --- | --- |
| 14. | Component |  | Represents physical modules which is a collection of components. |
| 15. | Node |  | Represents physical modules which are a collection of components. |
| 16. | Data Process/State |  | A circle in DFD represents a state or process which has been triggered due to some event or acion. |
| 17. | External entity |  | Represents external entities such as keyboard,sensors,etc. |
| 18. | Transition |  | Represents communication that occurs between processes. |
| 19. | Object Lifeline |  | Represents the vertical dimensions that the object communications. |
| 20. | Message | Message | Represents the message exchanged. |

**ABSTRACT:**

Hand gesture recognition systems received great attention in the recent few years because of their manifoldness applications and the ability to interact with machines efficiently through human-computer interaction. Due to the effect of lighting and complex background, most visual hand gesture recognition systems work only under restricted environment. With the rapid development of computer vision, the demand for interaction between human and machine is becoming more and more extensive. Since hand gestures are able to express enriched information, the hand gesture recognition is widely used in robot control, intelligent furniture and other aspects. One of the technical possibilities to implement hand gesture detection systems is to use the vision-based approach. The dataset uses all the required gestures. With all the features, an OpenCV and keras a hand gesture prediction model is built. The validation results indicate the precision and accuracy of the proposed model.

**EXISTING SYSTEM:**

Hand gesture recognition with surface electromyography (sEMG) is indispensable for Muscle-GestureComputer Interface. The usual focus of it is upon performance evaluation involving the accuracy and robustness of hand gesture recognition. However, addressing the reliability of such classifiers has been absent, to our best knowledge. This may be due to the lack of consensus on the definition of model reliability in this field. This paper has raised a concern about model reliability in sEMG-based hand gesture recognition. By defining the model reliability R as the quality of its uncertainty measures and providing an offline framework to investigate it, we have demonstrated that ECNN has great potential for classifying finger movements.

**Drawbacks:**

* Accuracy is low.
* Electromyography Signals are used to prepare the dataset.
* It is a complex process.

**INTRODUCTION:**

Hand gestures are an important part of nonverbal communication and form an integral part of our interactions with the environment. Notably, sign language is a set of hand gestures that is valuable to millions of disabled people. However, deaf/dumb users experience difficulty in communicating with the outside world as most neither understand nor can use sign language. Gesture recognition and classification platforms can aid in translating the gestures to those who do not understand sign language. There are two major approaches in the classification of hand gestures. The first approach is the vision-based approach. This involves the use of cameras to acquire the pose and movement of the hand and algorithms to process the recorded images. Although this approach is popular, it is very computationally intensive, as images or videos have to undergo significant preprocessing to segment features such as the image’s color, pixel values, and shape of hand.

**Domain overview:**

**4.1 Data Science:**

Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from structured and unstructured data, and apply knowledge and actionable insights from data across a broad range of application domains.

The term "data science" has been traced back to 1974, when Peter Naur proposed it as an alternative name for computer science. In 1996, the International Federation of Classification Societies became the first conference to specifically feature data science as a topic. However, the definition was still in flux.

The term “data science” was first coined in 2008 by D.J. Patil, and Jeff Hammer bacher, the pioneer leads of data and analytics efforts at LinkedIn and Facebook. In less than a decade, it has become one of the hottest and most trending professions in the market.

Data science is the field of study that combines domain expertise, programming skills, and knowledge of mathematics and statistics to extract meaningful insights from data.

Data science can be defined as a blend of mathematics, business acumen, tools, algorithms and machine learning techniques, all of which help us in finding out the hidden insights or patterns from raw data which can be of major use in the formation of big business decisions.

**Data Scientist:**

Data scientists examine which questions need answering and where to find the related data. They have business acumen and analytical skills as well as the ability to mine, clean, and present data. Businesses use data scientists to source, manage, and analyze large amounts of unstructured data.

**Required Skills for a Data Scientist:**

* **Programming**: Python, SQL, Scala, Java, R, MATLAB.
* **Machine Learning**: Natural Language Processing, Classification, Clustering.
* **Data Visualization**: Tableau, SAS, D3.js, Python, Java, R libraries.
* **Big data platforms**: MongoDB, Oracle, Microsoft Azure, Cloudera.
  1. **ARTIFICIAL INTELLIGENCE**:

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. The term may also be applied to any machine that exhibits traits associated with a human mind such as learning and problem-solving.

Artificial intelligence (AI) is [intelligence](https://en.wikipedia.org/wiki/Intelligence) demonstrated by [machines](https://en.wikipedia.org/wiki/Machine), as opposed to the natural intelligence [displayed by humans](https://en.wikipedia.org/wiki/Human_intelligence) or [animals](https://en.wikipedia.org/wiki/Animal_cognition). Leading AI textbooks define the field as the study of “[intelligent agents](https://en.wikipedia.org/wiki/Intelligent_agent)” any system that perceives its environment and takes actions that maximize its chance of achieving its goals.

Some popular accounts use the term “artificial intelligence” to describe machines that mimic “cognitive” functions that humans associate with the [human mind](https://en.wikipedia.org/wiki/Human_mind), such as “learning” and “problem solving”, however this definition is rejected by major AI researchers.

Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems. Specific applications of AI include expert systems, natural language processing, speech recognition and machine vision.

AI applications include advanced web search engines, recommendation systems (used by Youtube, Amazon and Netflix), Understanding human speech (such as Siri or Alexa), self-driving cars (e.g. Tesla), and competing at the highest level in strategic game systems (such as chess and Go), As machines become increasingly capable, tasks considered to require “intelligence” are often removed from the definition of AI, a phenomenon known as the AI effect. For instance, optical character recognition is frequently excluded from things considered to be AI, having become a routine technology.

Artificial intelligence was founded as an academic discipline in 1956, and in the years since has experienced several waves of optimism, followed by disappointment and the loss of funding (known as an “AI winter”), followed by new approaches, success and renewed funding.

AI research has tried and discarded many different approaches during its lifetime, including simulating the Paddy Leaves, modeling human problem solving, formal logic, large databases of knowledge and imitating animal behavior. In the first decades of the 21st century, highly mathematical statistical machine learning has dominated the field, and this technique has proved highly successful, helping to solve many challenging problems throughout industry and academia.

The various sub-fields of AI research are centered around particular goals and the use of particular tools. The traditional goals of AI research include reasoning, knowledge representation, planning, learning, natural language processing, perception and the ability to move and manipulate objects. General intelligence (the ability to solve an arbitrary problem) is among the field’s long-term goals.

To solve these problems, AI researchers use versions of search and mathematical optimization, formal logic, artificial neural networks, and methods based on statistics, probability and economics. AI also draws upon computer science, psychology, linguistics, philosophy, and many other fields.

The field was founded on the assumption that human intelligence “can be so precisely described that a machine can be made to simulate it”. This raises philosophical arguments about the mind and the ethics of creating artificial beings endowed with human-like intelligence.

These issues have been explored by myth, fiction and philosophy since antiquity. Science fiction and futurology have also suggested that, with its enormous potential and power, AI may become an existential risk to humanity.

As the hype around AI has accelerated, vendors have been scrambling to promote how their products and services use AI. Often what they refer to as AI is simply one component of AI, such as machine learning.

AI requires a foundation of specialized hardware and software for writing and training machine learning algorithms. No one programming language is synonymous with AI, but a few, including Python, R and Java, are popular.

In general, AI systems work by ingesting large amounts of labeled training data, analyzing the data for correlations and patterns, and using these patterns to make predictions about future states.

In this way, a chatbot that is fed examples of text chats can learn to produce life like exchanges with people, or an image recognition tool can learn to identify and describe objects in images by reviewing millions of examples.

AI programming focuses on three cognitive skills: learning, reasoning and self-correction.

**Learning processes.** This aspect of AI programming focuses on acquiring data and creating rules for how to turn the data into actionable information. The rules, which are called algorithms, provide computing devices with step-by-step instructions for how to complete a specific task.

**Reasoning processes.** This aspect of AI programming focuses on choosing the right algorithm to reach a desired outcome.

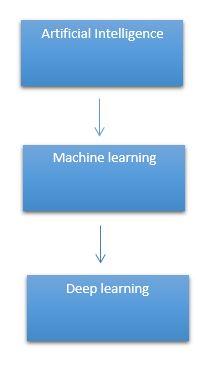
**Self-correction processes.** This aspect of AI programming is designed to continually fine-tune algorithms and ensure they provide the most accurate results possible.

AI is important because it can give enterprises insights into their operations that they may not have been aware of previously and because, in some cases, AI can perform tasks better than humans. Particularly when it comes to repetitive, detail-oriented tasks like analyzing large numbers of legal documents to ensure relevant fields are filled in properly, AI tools often complete jobs quickly and with relatively few errors.

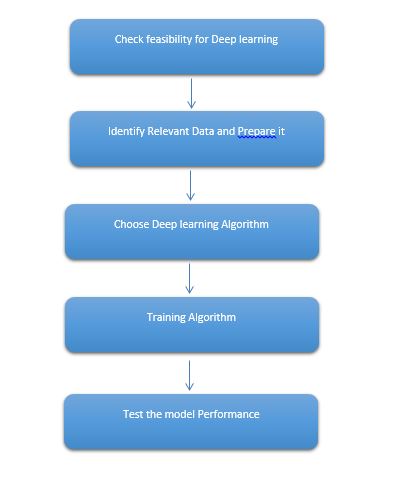
Artificial neural networks and deep learning artificial intelligence technologies are quickly evolving, primarily because AI processes large amounts of data much faster and makes predictions more accurately than humanly possible.

* 1. **DEEP LEARNING**

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human disease so deep learning is also a kind of mimic of human disease. It’s on hype nowadays because earlier we did not have that much processing power and a lot of data. A formal definition of deep learning is- neurons Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. In disease approximately 100 billion neurons all together this is a picture of an individual neuron and each neuron is connected through thousands of their neighbors. The question here is how it recreates these neurons in a computer. So, it creates an artificial structure called an artificial neural net where we have nodes or neurons. It has some neurons for input value and some for output value and in between, there may be lots of neurons interconnected in the hidden layer.



# It need to identify the actual problem in order to get the right solution and it should be understood, the feasibility of the Deep Learning should also be checked (whether it should fit Deep Learning or not). It needs to identify the relevant data which should correspond to the actual problem and should be prepared accordingly. Choose the Deep Learning Algorithm appropriately. Algorithm should be used while training the dataset. Final testing should be done on the dataset



Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

Deep-learning architectures such as deep neural networks, deep belief networks, deep reinforcement learning, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance.

Artificial neural networks (ANNs) were inspired by information processing and distributed communication nodes in biological systems. ANNs have various differences from biological disease. Specifically, neural networks tend to be static and symbolic, while the biological disease of most living organisms is dynamic (plastic) and analogue.

The adjective "deep" in deep learning refers to the use of multiple layers in the network. Early work showed that a linear perceptron cannot be a universal classifier, but that a network with a non-polynomial activation function with one hidden layer of unbounded width can. Deep learning is a modern variation which is concerned with an unbounded number of layers of bounded size, which permits practical application and optimized implementation, while retaining theoretical universality under mild conditions. In deep learning the layers are also permitted to be heterogeneous and to deviate widely from biologically informed connectionist models, for the sake of efficiency, trainability and understandability, whence the "structured" part.

Deep learning is a class of [machine learning](https://en.wikipedia.org/wiki/Machine_learning) [algorithms](https://en.wikipedia.org/wiki/Algorithm) that uses multiple layers to progressively extract higher-level features from the raw input. For example, in [image processing](https://en.wikipedia.org/wiki/Image_processing), lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

**Interpretations:**

Deep neural networks are generally interpreted in terms of the universal approximation theorem or probabilistic inference.

The classic universal approximation theorem concerns the capacity of feed-forward neural networks with a single hidden layer of finite size to approximate continuous functions. In 1989, the first proof was published by George Cybenko for sigmoid activation functions and was generalised to feed-forward multi-layer architectures in 1991 by Kurt Hornik. Recent work also showed that universal approximation also holds for non-bounded activation functions such as the rectified linear unit.

The universal approximation theorem for deep neural networks concerns the capacity of networks with bounded width but the depth is allowed to grow proved that if the width of a deep neural network with ReLU activation is strictly larger than the input dimension, then the network can approximate any Lebesgue integrable function; If the width is smaller or equal to the input dimension, then deep neural network is not a universal approximator.

The probabilistic interpretation derives from the field of machine learning. It features inference, as well as the optimization concepts of training and testing, related to fitting and generalization, respectively. More specifically, the probabilistic interpretation considers the activation nonlinearity as a cumulative distribution function. The probabilistic interpretation led to the introduction of dropout as regularizer in neural networks. The probabilistic interpretation was introduced by researchers including Hopfield, Widrow and Narendra and popularized in surveys such as the one by Bishop.

### Deep learning revolution:

### In 2012, a team led by George E. Dahl won the "Merck Molecular Activity Challenge" using multi-task deep neural networks to predict the biomolecular target of one drug. In 2014, Hochreiter's group used deep learning to detect off-target and toxic effects of environmental chemicals in nutrients, household products and drugs and won the "Tox21 Data Challenge" of NIH, FDA and NCATS.

Significant additional impacts in image or object recognition were felt from 2011 to 2012. Although CNNs trained by back-propagation had been around for decades, and GPU implementations of NNs for years, including CNNs, fast implementations of CNNs on GPUs were needed to progress on computer vision. In 2011, this approach achieved for the first time superhuman performance in a visual pattern recognition contest. Also in 2011, it won the ICDAR Chinese handwriting contest, and in May 2012, it won the ISBI image segmentation contest. Until 2011, CNNs did not play a major role at computer vision conferences, but in June 2012, a paper by Ciresan et al. at the leading conference CVPR showed how max-pooling CNNs on GPU can dramatically improve many vision benchmark records.

In October 2012, a similar system by Krizhevsky et al. won the large-scale ImageNet competition by a significant margin over shallow machine learning methods. In November 2012, Ciresan et al.'s system also won the ICPR contest on analysis of large medical images for cancer detection, and in the following year also the MICCAI Grand Challenge on the same topic. In 2013 and 2014, the error rate on the ImageNet task using deep learning was further reduced, following a similar trend in large-scale speech recognition.

Image classification was then extended to the more challenging task of generating descriptions (captions) for images, often as a combination of CNNs and LSTMs.

Some researchers state that the October 2012 ImageNet victory anchored the start of a "deep learning revolution" that has transformed the AI industry.

In March 2019, Yoshua Bengio, Geoffrey Hinton and Yann LeCun were awarded the Turing Award for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.

**5. PROPOSED SYSTEM:**

We proposed a new and robust deep learning model based on a convolutional neural network (CNN) to automatically detect hand gesture movements. We used whole images, so it was not necessary to perform any pre-processing or the waste types, samples of more number of images are collected that comprised of different class data. Different hand movement’s images are collected for each classes that was classified into input images. The DL method used in the study is the Convolutional Neural Network (CNN). It is predicted that the success of the obtained results will increase if the CNN method is supported by adding extra feature extraction methods and detection of hand gesture movements successfully.

**5.1 Advantages:**

* Accuracy may be improvised.
* No signals are required to prepare the dataset.

**6. PREPARING DATASET:**

This dataset contains approximately train and test image records of features extracted, which were then classified into number of classes:

* ThumbsUp
* ThumbsDown
* Callme
* Looser
* etc

**7. LITERATURE SURVEY**

**General**

A literature review is a body of text that aims to review the critical points of current knowledge on and/or methodological approaches to a particular topic. It is secondary sources and discuss published information in a particular subject area and sometimes information in a particular subject area within a certain time period.

Its ultimate goal is to bring the reader up to date with current literature on a topic and forms the basis for another goal, such as future research that may be needed in the area and precedes a research proposal and may be just a simple summary of sources. Usually, it has an organizational pattern and combines both summary and synthesis.

A summary is a recap of important information about the source, but a synthesis is a re-organization, reshuffling of information. It might give a new interpretation of old material or combine new with old interpretations or it might trace the intellectual progression of the field, including major debates. Depending on the situation, the literature review may evaluate the sources and advise the reader on the most pertinent or relevant of them. Loan default trends have been long studied from a socio-economic stand point.

Most economics surveys believe in empirical modeling of these complex systems in order to be able to predict the loan default rate for a particular individual. The use of machine learning for such tasks is a trend which it is observing now. Some of the survey’s to understand the past and present perspective of loan approval or not.

**Review of Literature Survey**

**Title** : Hand Gesture Recognition Based on Computer Vision: A Review of Techniques

**Author** : Munir Oudah 1 , Ali Al-Naji 1,2,\* and Javaan Chahl 2

**Year** : 2020

Hand gestures are a form of nonverbal communication that can be used in several fields such as communication between deaf-mute people, robot control, human–computer interaction (HCI), home automation and medical applications. Research papers based on hand gestures have adopted many different techniques, including those based on instrumented sensor technology and computer vision. In other words, the hand sign can be classified under many headings, such as posture and gesture, as well as dynamic and static, or a hybrid of the two. This paper focuses on a review of the literature on hand gesture techniques and introduces their merits and limitations under different circumstances. In addition, it tabulates the performance of these methods, focusing on computer vision techniques that deal with the similarity and difference points, technique of hand segmentation used, classification algorithms and drawbacks, number and types of gestures, dataset used, detection range (distance) and type of camera used. This paper is a thorough general overview of hand gesture methods with a brief discussion of some possible applications.

**Title** : Hand Gesture Recognition with Skin Detection and Deep Learning Method

**Author** : Hanwen Huang1 , Yanwen Chong2\*

**Year** : 2019

Gesture recognition, although has been exploring for many years, is still a challenging problem. Complex background, camera angles and illumination conditions make the problem more difficult. Thus, this paper presents a fast and robust method for hand gesture recognition based on RGB video. First we detect the skin based on their color. Then we extract the contour and segment the hand region. Finally we recognize the gesture. The results of experiment demonstrate that the proposed method are efficient to recognize gesture with a higher accuracy than the state of the art.

**Title** : Hand Gesture Recognition System Using Camera

**Author** : Viraj Shinde, Tushar Bacchav, Jitendra Pawar

**Year** : 2014

In this paper, we focus on using pointing behavior for a natural interface, Hand gesture recognition based human-machine interface is being developed vigorously in recent years. Due to the effect of lighting and complex background, most visual hand gesture recognition systems work only under restricted environment. To classify the dynamic hand gestures, we developed a simple and fast motion history image based method. In recent years, the gesture control technique has become a new developmental trend for many human-based electronics products. This technique let people can control these products more naturally, intuitively and conveniently. In this paper, a fast gesture recognition scheme is proposed to be an interface for the human-machine interaction (HMI) of systems. This paper presents some low-complexity algorithms and gestures to reduce the gesture recognition complexity and be more suitable for controlling real-time computer systems.

**Title** : Static Hand Gesture Recognition using Convolutional Neural Network with Data Augmentation

**Author** : Md. Zahirul Islam, Mohammad Shahadat Hossain

Computer is a part and parcel in our day to day life and used in various fields. The interaction of human and computer is accomplished by traditional input devices like mouse, keyboard etc. Hand gestures can be a useful medium of human-computer interaction and can make the interaction easier. Gestures vary in orientation and shape from person to person. So, non-linearity exists in this problem. Recent research has proved the supremacy of Convolutional Neural Network (CNN) for image representation and classification. Since, CNN can learn complex and non-linear relationships among images, in this paper, a static hand gesture recognition method using CNN was proposed. Data augmentation like re-scaling, zooming, shearing, rotation, width and height shifting was applied to the dataset. The model was trained on 8000 images and tested on 1600 images which were divided into 10 classes. The model with augmented data achieved accuracy 97.12% which is nearly 4% higher than the model without augmentation (92.87%).

**Title** : Real-Time Hand Gesture Recognition Using Finger Segmentation

**Author** : Zhi-hua Chen,1 Jung-Tae Kim,1 Jianning Liang

**Year** : 2014

Hand gesture recognition is very significant for human-computer interaction. In this work, we present a novel real-time method for hand gesture recognition. In our framework, the hand region is extracted from the background with the background subtraction method. Then, the palm and fingers are segmented so as to detect and recognize the fingers. Finally, a rule classifier is applied to predict the labels of hand gestures. The experiments on the data set of 1300 images show that our method performs well and is highly efficient. Moreover, our method shows better performance than a state-of-art method on another data set of hand gestures.

**8. SYSTEM STUDY**

**Aim:**

Hand Gesture is one of the major factors in our communication society. There are lot of Sign languages that are actively present in the world. So we can’t able to classify the action easily. So this project can easily classify the hand gesture.

**Objectives:**

The goal is to develop a deep learning model for Hand gesture classification by convolutional neural network algorithm for potentially classifying the results in the form of best accuracy by comparing the CNN architectures.

**Scope:**

Hand Gesture images are collected. We have to train the machine to classify the types of gesture. This project contains different types of gesture like ThumbsUp, Thumbsdown, Callme, Looser, etc. We train to teach the machine to ac`hieve the accuracy and get the possible outcome.

**OUTLINE OF THE PROJECT**

**Overview of the system:**

* Define a problem
* Gathering image data set
* Evaluating algorithms
* Detecting results

The steps involved in Building the data model is depicted below.

**Data collection** (Splitting Training set & Test) set)

**Pre Processing** (Sequential)

**Building classification Model**

**Prediction (**Hand Gesture Detection**)**

Fig: data flow diagram for CNN model

1. **PROJECT REQUIREMENTS**

**General:**

Requirements are the basic constrains that are required to develop a system. Requirements are collected while designing the system. The following are the requirements that are to be discussed.

1. Functional requirements

2. Non-Functional requirements

3. Environment requirements

A. Hardware requirements

B. software requirements

**9.1 Functional requirements:**

The software requirements specification is a technical specification of requirements for the software product. It is the first step in the requirements analysis process. It lists requirements of a particular software system. The following details to follow the special libraries like tensorflow, keras, matplotlib.

**9.2 Non-Functional Requirements:**

Process of functional steps,

1. Problem define
2. Preparing data
3. Evaluating algorithm
4. Improving results
5. Prediction the result

**ENVIRONMENTAL REQUIREMENTS:**

**Software Requirements:**

* Operating System : Windows 10 or later
* Simulation Tool : Anaconda with Jupyter Notebook
* Language : Python

**Hardware requirements:**

* Processor : intel i3 or later
* Hard disk : minimum 10 GB
* RAM : minimum 4 GB

**FEASIBILITY STUDY**

**Splitting the dataset:**

The data use is usually split into training data and test data. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. It has the test dataset (or subset) in order to test our models and it will do this using Tensorflow library in Python using the Keras method.

**Construction of a Detecting Model:**

## Deep learning needs data gathering have lot of past image data’s. Training and testing this model working and predicting correctly.

Data Gathering

CNN Algorithm

Train model

Test model

Prediction

Steps of dataflow diagram

**DESIGN ARCHITECTURE**

**General**

Design is meaningful engineering representation of something that is to be built. Software design is a process design is the perfect way to accurately translate requirements in to a finished software product. Design creates a representation or model, provides detail about software data structure, architecture, interfaces and components that are necessary to implement a system.

**Data Flow Diagram:**

Image Details

Test dataset

Pre-processing

Sign Gesture Detection

CNN Algorithm

Training dataset

Process of dataflow diagram

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design). A DFD shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will be stored. It does not show information about process timing or whether processes will operate in sequence or in parallel, unlike a traditional structured flowchart which focuses on control flow, or a UML activity workflow diagram, which presents both control and data flows as a unified model. Data flow diagrams are also known as bubble charts. DFD is a designing tool used in the top down approach to Systems Design. Symbols and Notations Used in DFDs Using any convention’s DFD rules or guidelines, the symbols depict the four components of data flow diagrams.

External entity: an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system or a business system. They are also known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram.

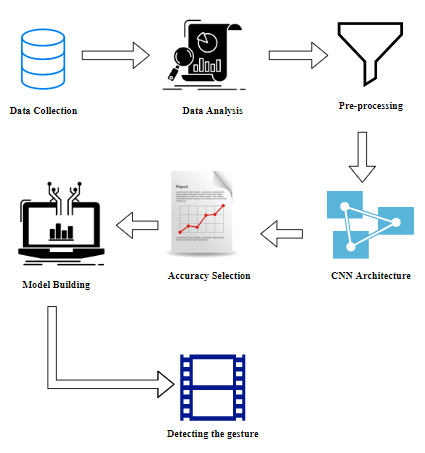
Process: any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules.

Data store: files or repositories that hold information for later use, such as a database table or a membership form.

Data flow: the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labeled with a short data name, like “Billing details.”

DFD levels and layers A data flow diagram can dive into progressively more detail by using levels and layers, zeroing in on a particular piece. DFD levels are numbered 0, 1 or 2, and occasionally go to even Level 3 or beyond. The necessary level of detail depends on the scope of what you are trying to accomplish. DFD Level 0 is also called a Context Diagram. It’s a basic overview of the whole system or process being analyzed or modeled. It’s designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.

**System Architecture:**



**Work flow diagram:**

Data Collection

Pre-processing

Training Dataset

Testing Dataset

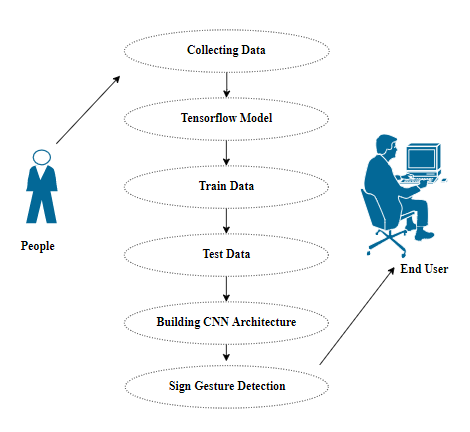
CNN algorithm

Model

Gesture Detection

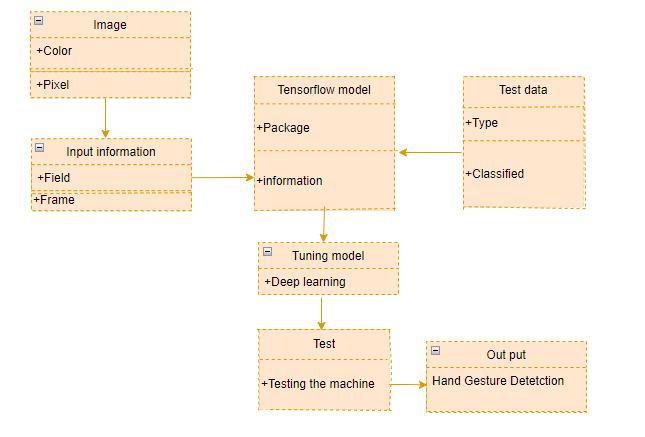
Workflow Diagram

**Use Case Diagram:**



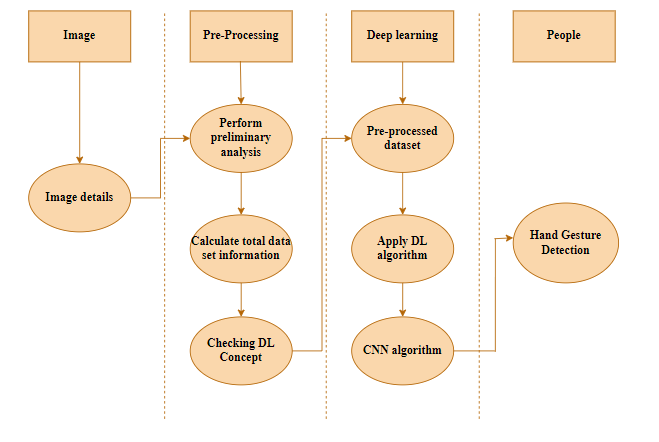
Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases. So, it can say that uses cases are nothing but the system functionalities written in an organized manner.

**CLASS DIAGRAM:**

****

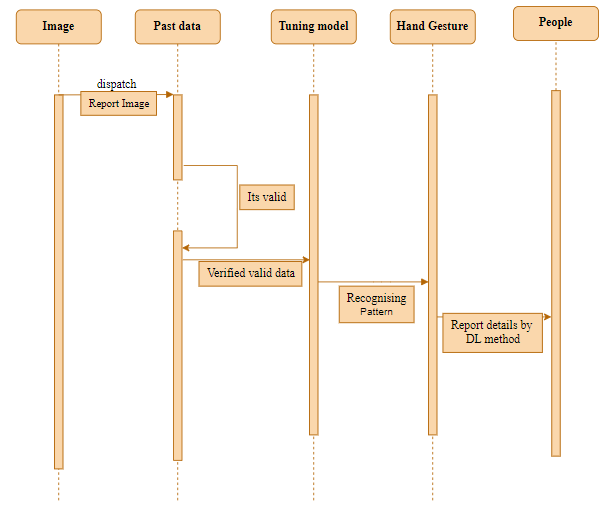
Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represent the whole system. The name of the class diagram should be meaningful to describe the aspect of the system. Each element and their relationships should be identified in advance Responsibility (attributes and methods) of each class should be clearly identified for each class minimum number of properties should be specified and because, unnecessary properties will make the diagram complicated. Use notes whenever required to describe some aspect of the diagram and at the end of the drawing it should be understandable to the developer/coder. Finally, before making the final version, the diagram should be drawn on plain paper and rework as many times as possible to make it correct.

**ACTIVITY DIAGRAM:**

****

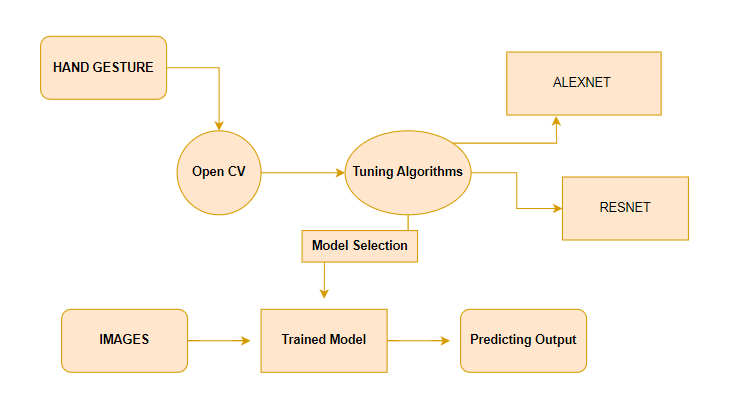
Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part. It does not show any message flow from one activity to another. Activity diagram is some time considered as the flow chart. Although the diagrams looks like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

**SEQUENCE DIAGRAM:**

****

Sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifact for dynamic modelling, which focuses on identifying the behaviour within your system. Other dynamic modelling techniques include [activity diagramming](http://agilemodeling.com/artifacts/activityDiagram.htm), [communication diagramming](http://agilemodeling.com/artifacts/communicationDiagram.htm), [timing diagramming](http://agilemodeling.com/artifacts/timingDiagram.htm), and [interaction overview diagramming](http://agilemodeling.com/artifacts/interactionOverviewDiagram.htm). Sequence diagrams, along with [class diagrams](http://agilemodeling.com/artifacts/classDiagram.htm) and [physical data models](http://agiledata.org/essays/dataModeling101.html) are in my opinion the most important design-level models for modern business application development.

**ER DIAGRAM:**

****

An entity relationship diagram (ERD), also known as an entity relationship model, is a graphical representation of an information system that depicts the relationships among people, objects, places, concepts or events within that system. An ERD is a data modeling technique that can help define business processes and be used as the foundation for a relational database. Entity relationship diagrams provide a visual starting point for database design that can also be used to help determine information system requirements throughout an organization. After a relational database is rolled out, an ERD can still serve as a referral point, should any debugging or business process re-engineering be needed later.

1. **SOFTWARE DESCRIPTION**

Anaconda is a [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source) distribution of the [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) and [R](https://en.wikipedia.org/wiki/R_(programming_language)) programming languages for [scientific computing](https://en.wikipedia.org/wiki/Scientific_computing) ([data science](https://en.wikipedia.org/wiki/Data_science), [machine learning](https://en.wikipedia.org/wiki/Machine_learning) applications, large-scale data processing, [predictive analytics](https://en.wikipedia.org/wiki/Predictive_analytics), etc.), that aims to simplify [package management](https://en.wikipedia.org/wiki/Package_management) and deployment. Package versions are managed by the [package management system](https://en.wikipedia.org/wiki/Package_manager) “Conda”. The Anaconda distribution is used by over 12 million users and includes more than 1400 popular data-science packages suitable for Windows, Linux, and MacOS. So, Anaconda distribution comes with more than 1,400 packages as well as the [Conda](https://en.wikipedia.org/wiki/Conda_(package_manager)) package and virtual environment manager called Anaconda Navigator and it eliminates the need to learn to install each library independently. The open source packages can be individually installed from the Anaconda repository with the conda install command or using the pip install command that is installed with Anaconda. [Pip packages](https://en.wikipedia.org/wiki/Pip_(package_manager)) provide many of the features of conda packages and in most cases they can work together. Custom packages can be made using the conda build command, and can be shared with others by uploading them to Anaconda Cloud, [PyPI](https://en.wikipedia.org/wiki/Python_Package_Index) or other repositories. The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, you can create new environments that include any version of Python packaged with conda.

**12.1 ANACONDA NAVIGATOR:**

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda.org or in a local Anaconda Repository.

Anaconda. Now, if you are primarily doing data science work, Anaconda is also a great option. Anaconda is created by Continuum Analytics, and it is a Python distribution that comes preinstalled with lots of useful python libraries for data science.

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

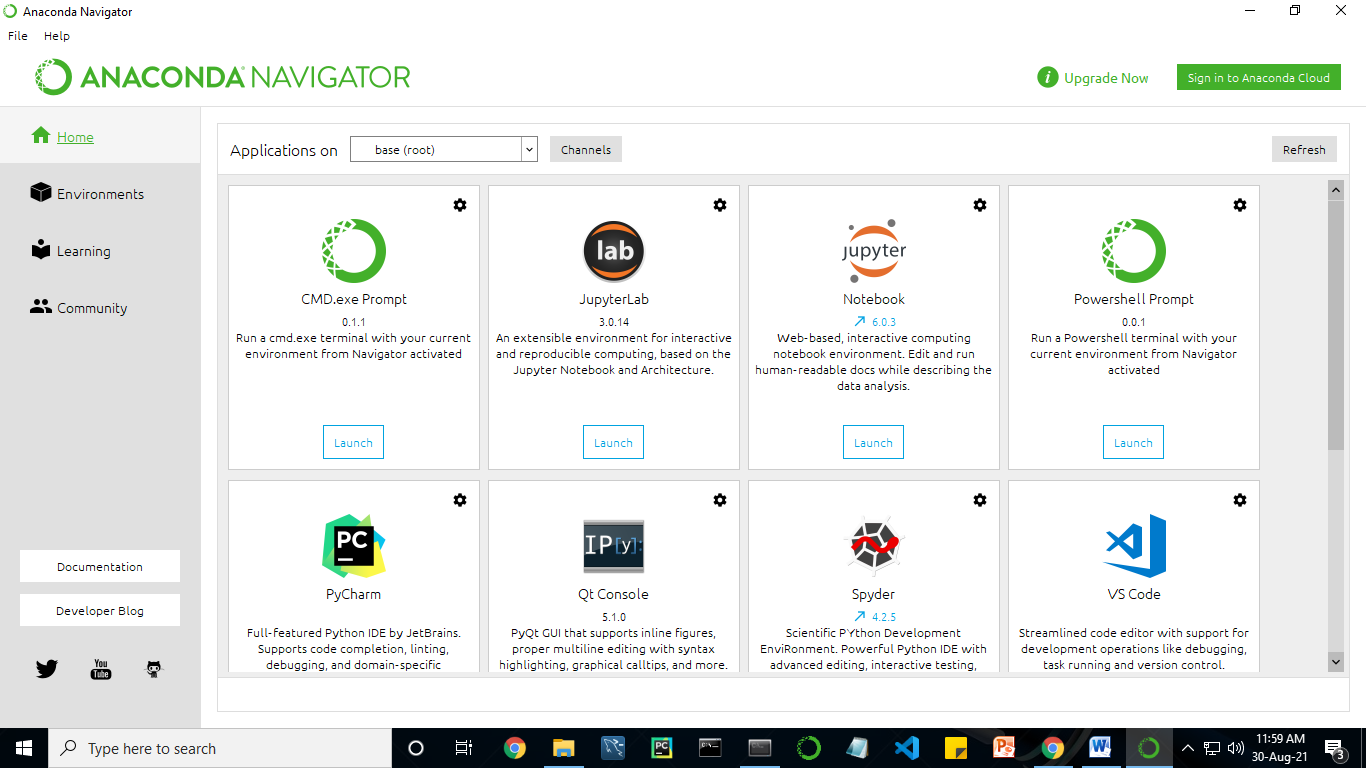
In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages and use multiple environments to separate these different versions.

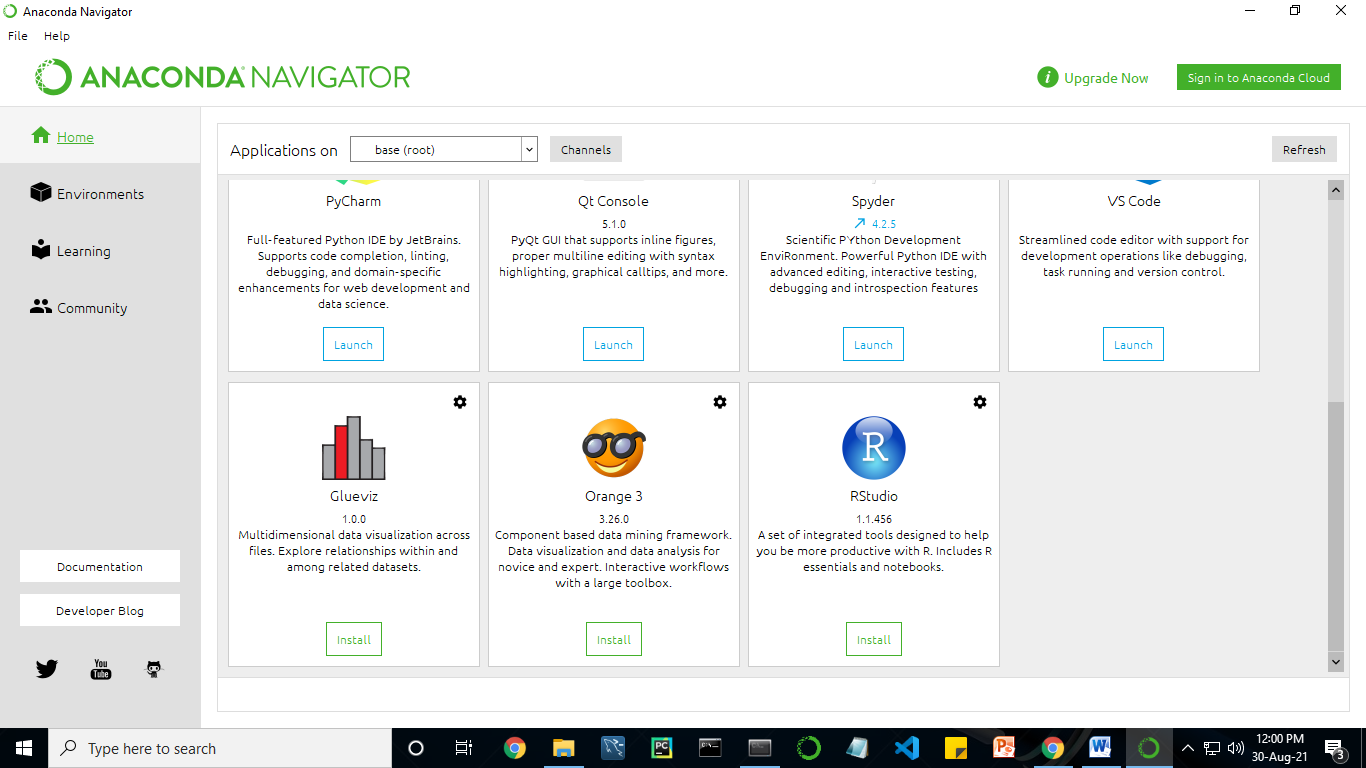
The command-line program conda is both a package manager and an environment manager. This helps data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages, and update them – all inside Navigator.

The following applications are available by default in Navigator:

* [JupyterLab](https://jupyterlab.readthedocs.io/en/stable/)
* [Jupyter Notebook](https://jupyter.readthedocs.io/en/latest/)
* [Spyder](https://www.spyder-ide.org/)
* [PyCharm](https://www.jetbrains.com/pycharm/documentation/)
* [VSCode](https://code.visualstudio.com/docs)
* [Glueviz](http://glueviz.org/en/stable/)
* [Orange 3 App](http://orange.biolab.si/docs/)
* [RStudio](http://docs.rstudio.com/)
* Anaconda Prompt (Windows only)
* Anaconda PowerShell (Windows only)





Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution.

Navigator allows you to launch common Python programs and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository.

Anaconda comes with many built-in packages that you can easily find with conda list on your anaconda prompt. As it has lots of packages (many of which are rarely used), it requires lots of space and time as well. If you have enough space, time and do not want to burden yourself to install small utilities like JSON, YAML, you better go for Anaconda.

**JUPYTER NOTEBOOK:**

This website acts as “meta” documentation for the Jupyter ecosystem. It has a collection of resources to navigate the tools and communities in this ecosystem, and to help you get started.

Project Jupyter is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". It was spun off from IPython in 2014 by Fernando Perez.

Notebook documents are documents produced by the [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-app), which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc…). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc.) as well as executable documents which can be run to perform data analysis.

## Installation: The easiest way to install the Jupyter Notebook App is installing a scientific python distribution which also includes scientific python packages. The most common distribution is called **Anaconda**

# Running the Jupyter Notebook

## Launching Jupyter Notebook App: The [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-app) can be launched by clicking on the Jupyter Notebook icon installed by Anaconda in the start menu (Windows) or by typing in a terminal (cmd on Windows): “jupyter notebook”

## This will launch a new browser window (or a new tab) showing the [Notebook Dashboard](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#dashboard), a sort of control panel that allows (among other things) to select which notebook to open.

## When started, the [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-app) can access only files within its start-up folder (including any sub-folder). No configuration is necessary if you place your notebooks in your home folder or subfolders. Otherwise, you need to choose a [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-app) start-up folder which will contain all the notebooks.

## Save notebooks: Modifications to the notebooks are automatically saved every few minutes. To avoid modifying the original notebook, make a copy of the notebook document (menu file -> make a copy…) and save the modifications on the copy.

## Executing a notebook: Download the notebook you want to execute and put it in your notebook folder (or a sub-folder of it).

* Launch the jupyter notebook app
* In the [Notebook Dashboard](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#dashboard) navigate to find the notebook: clicking on its name will open it in a new browser tab.
* Click on the menu Help -> User Interface Tour for an overview of the [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-app) user interface.
* You can run the notebook document step-by-step (one cell a time) by pressing shift + enter.
* You can run the whole notebook in a single step by clicking on the menu Cell -> Run All.
* To restart the [kernel](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#kernel) (i.e. the computational engine), click on the menu Kernel -> Restart. This can be useful to start over a computation from scratch (e.g. variables are deleted, open files are closed, etc…).

[**Purpose**](https://www.google.com/search?q=project+jupyter+purpose&sa=X&ved=2ahUKEwin49vtmdjyAhXx4zgGHXSOCuwQ6BMoADAkegQINxAC&cshid=1630307847256010)**:** To support [interactive](https://www.google.com/search?q=interactive&stick=H4sIAAAAAAAAAONgVuLUz9U3MM0uyYpfxMqdmVeSWpSYXJJZlgoApkTFPhsAAAA&sa=X&ved=2ahUKEwin49vtmdjyAhXx4zgGHXSOCuwQmxMoATAkegQINxAD&cshid=1630307847256010) data science and scientific computing across all programming languages.

**File Extension:** An IPYNB file is a notebook document created by Jupyter Notebook, an interactive computational environment that helps scientists manipulate and analyze data using Python.

**JUPYTER Notebook App:** The Jupyter Notebook Appis a server-client application that allows editing and running [notebook documents](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-document) via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet.

In addition to displaying/editing/running notebook documents, the Jupyter Notebook App has a “Dashboard” ([Notebook Dashboard](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#dashboard)), a “control panel” showing local files and allowing to open notebook documents or shutting down their [kernels](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#kernel).

## [**kernel**](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#id7)**:** A notebook kernel is a “computational engine” that executes the code contained in a [Notebook document](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-document). The ipython kernel*,* referenced in this guide, executes python code. Kernels for many other languages exist ([official kernels](http://jupyter.readthedocs.org/en/latest/#kernels)).

When you open a [Notebook document](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-document), the associated kernel is automatically launched. When the notebook is executed (either cell-by-cell or with menu Cell -> Run All), the kernel performs the computation and produces the results. Depending on the type of computations, the kernel may consume significant CPU and RAM. Note that the RAM is not released until the kernel is shut-down

## [Notebook Dashboard](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#id8): The Notebook Dashboard is the component which is shown first when you launch [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-app). The Notebook Dashboard is mainly used to open [notebook documents](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-document), and to manage the running [kernels](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#kernel) (visualize and shutdown).

The Notebook Dashboard has other features similar to a file manager, namely navigating folders and renaming/deleting files

**Working Process:**

* Download and install anaconda and get the most useful package for machine learning in Python.
* Load a dataset and understand its structure using statistical summaries and data visualization.
* Machine learning models, pick the best and build confidence that the accuracy is reliable.

Python is a popular and powerful interpreted language. Unlike R, Python is a complete language and platform that you can use for both research and development and developing production systems. There are also a lot of modules and libraries to choose from, providing multiple ways to do each task. It can feel overwhelming.

The best way to get started using Python for machine learning is to complete a project.

* It will force you to install and start the Python interpreter (at the very least).
* It will give you a bird’s eye view of how to step through a small project.
* It will give you confidence, maybe to go on to your own small projects.

When you are applying machine learning to your own datasets, you are working on a project. A machine learning project may not be linear, but it has a number of well-known steps:

* Define Problem.
* Prepare Data.
* Evaluate Algorithms.
* Improve Results.
* Present Results.

The best way to really come to terms with a new platform or tool is to work through a machine learning project end-to-end and cover the key steps. Namely, from loading data, summarizing data, evaluating algorithms and making some predictions.

Here is an overview of what we are going to cover:

1. Installing the Python anaconda platform.
2. Loading the dataset.
3. Summarizing the dataset.
4. Visualizing the dataset.
5. Evaluating some algorithms.
6. Making some predictions.

**PYCHARM:**

PyCharm is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for Python developers, tightly integrated to create a convenient environment for productive [Python](https://www.jetbrains.com/help/pycharm/python.html), [web](https://www.jetbrains.com/help/pycharm/web-frameworks.html), and [data science](https://www.jetbrains.com/help/pycharm/scientific-tools.html) development. Code faster and with more easily in a smart and configurable editor with code completion, snippets, code folding and split windows support.

**PyCharm Features**

* **Intelligent Coding Assistance** – PyCharm provides smart code completion, code inspections, on-the-fly error highlighting and quick-fixes, along with automated code refactorings and rich navigation capabilities.
* **Intelligent Code Editor** – PyCharm’s smart code editor provides first-class support for Python, JavaScript, CoffeeScript, TypeScript, CSS, popular template languages and more. Take advantage of language-aware code completion, error detection, and on-the-fly code fixes!
* **Smart Code Navigation** – Use smart search to jump to any class, file or symbol, or even any IDE action or tool window. It only takes one click to switch to the declaration, super method, test, usages, implementation, and more.
* **Fast and Safe Refactorings** – Refactor your code the intelligent way, with safe Rename and Delete, Extract Method, Introduce Variable, Inline Variable or Method, and other refactorings. Language and framework-specific refactorings help you perform project-wide changes.
* **Built-in Developer Tools** – PyCharm’s huge collection of tools out of the box includes an integrated debugger and test runner; Python profiler; a built-in terminal; integration with major VCS and built-in database tools; remote development capabilities with remote interpreters; an integrated ssh terminal; and integration with Docker and Vagrant.
* **Debugging, Testing and Profiling** – Use the powerful debugger with a graphical UI for Python and JavaScript. Create and run your tests with coding assistance and a GUI-based test runner. Take full control of your code with Python Profiler integration.
* **VCS, Deployment and Remote Development** – Save time with a unified UI for working with Git, SVN, Mercurial or other version control systems. Run and debug your application on remote machines. Easily configure automatic deployment to a remote host or VM and manage your infrastructure with Vagrant and Docker.
* **Database tools** – Access Oracle, SQL Server, PostgreSQL, MySQL and other databases right from the IDE. Rely on PyCharm’s help when editing SQL code, running queries, browsing data, and altering schemas.
* **Web Development** – In addition to Python, PyCharm provides first-class support for various Python web development frameworks, specific template languages, JavaScript, CoffeeScript, TypeScript, HTML/CSS, AngularJS, Node.js, and more.
* **Python Web frameworks** – PyCharm offers great framework-specific support for modern web development frameworks such as Django, Flask, Google App Engine, Pyramid, and web2py, including Django templates debugger, manage.py and appcfg.py tools, special auto completion and navigation, just to name a few.
* **JavaScript & HTML –**PyCharm provides first-class support for JavaScript, Coffee Script, Type Script, HTML and CSS, as well as their modern successors. The JavaScript debugger is included in PyCharm and is integrated with the Django server run configuration.
* **Live Edit** – Live Editing Preview lets you open a page in the editor and the browser and see the changes being made in code instantly in the browser. PyCharm auto-saves your changes, and the browser smartly updates the page on the fly, showing your edits.
* **Scientific Tools** – PyCharm integrates with Ipython Notebook, has an interactive Python console, and supports Anaconda as well as multiple scientific packages including Matplotlib and NumPy.
* **Interactive Python console** – You can run a REPL Python console in PyCharm which offers many advantages over the standard one: on-the-fly syntax check with inspections, braces and quotes matching, and of course code completion.
* **Scientific Stack Support** – PyCharm has built-in support for scientific libraries. It supports Pandas, Numpy, Matplotlib, and other scientific libraries, offering you best-in-class code intelligence, graphs, array viewers and much more.
* **Conda Integration** – Keep your dependencies isolated by having separate Conda environments per project, PyCharm makes it easy for you to create and select the right environment.
* **Customizable and Cross-platform IDE** – Use PyCharm on Windows, Mac OS and Linux with a single license key. Enjoy a fine-tuned workspace with customizable color schemes and key-bindings, with VIM emulation available.
* **Customizable UI** – Enjoy a fine-tuned workspace with customizable color schemes and key-bindings.
* **Plugins** – More than 10 years of IntelliJ platform development gives PyCharm 50+ IDE plugins of different nature, including support for additional VCS, integrations with different tools and frameworks, and editor enhancements such as Vim emulation.
* **Cross-platform IDE** – PyCharm works on Windows, Mac OS or Linux. You can install and run PyCharm on as many machines as you have, and use the same environment and functionality across all your machines.

**PYTHON**

**Introduction:**

**Python** is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language) [high-level](https://en.wikipedia.org/wiki/High-level_programming_language) [general-purpose programming language](https://en.wikipedia.org/wiki/General-purpose_programming_language). Its design philosophy emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability) with its use of [significant indentation](https://en.wikipedia.org/wiki/Off-side_rule). Its [language constructs](https://en.wikipedia.org/wiki/Language_construct) as well as its [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) approach aim to help [programmers](https://en.wikipedia.org/wiki/Programmers) write clear, logical code for small and large-scale projects.

Python is [dynamically-typed](https://en.wikipedia.org/wiki/Type_system#DYNAMIC) and [garbage-collected](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)). It supports multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigms), including [structured](https://en.wikipedia.org/wiki/Structured_programming) (particularly, [procedural](https://en.wikipedia.org/wiki/Procedural_programming)), object-oriented and [functional programming](https://en.wikipedia.org/wiki/Functional_programming). It is often described as a "batteries included" language due to its comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library).

[Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) began working on Python in the late 1980s, as a successor to the [ABC programming language](https://en.wikipedia.org/wiki/ABC_(programming_language)), and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features, such as [list comprehensions](https://en.wikipedia.org/wiki/List_comprehension) and a garbage collection system using [reference counting](https://en.wikipedia.org/wiki/Reference_counting). Python 3.0 was released in 2008 and was a major revision of the language that is not completely [backward-compatible](https://en.wikipedia.org/wiki/Backward_compatibility). Python 2 was discontinued with version 2.7.18 in 2020.

Python consistently ranks as one of the most popular programming languages

**History:**

Python was conceived in the late 1980s by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) at [Centrum Wiskunde & Informatica](https://en.wikipedia.org/wiki/Centrum_Wiskunde_%26_Informatica) (CWI) in the [Netherlands](https://en.wikipedia.org/wiki/Netherlands) as a successor to [ABC programming language](https://en.wikipedia.org/wiki/ABC_(programming_language)), which was inspired by [SETL](https://en.wikipedia.org/wiki/SETL),  capable of [exception handling](https://en.wikipedia.org/wiki/Exception_handling) and interfacing with the [Amoeba](https://en.wikipedia.org/wiki/Amoeba_(operating_system)) operating system. Its implementation began in December 1989.  Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's [Benevolent Dictator For Life](https://en.wikipedia.org/wiki/Benevolent_Dictator_For_Life), a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January 2019, active Python core developers elected a 5-member "Steering Council" to lead the project.  As of 2021, the current members of this council are Barry Warsaw, Brett Cannon, Carol Willing, Thomas Wouters, and Pablo Galindo Salgado.

Python 2.0 was released on 16 October 2000, with many major new features, including a [cycle-detecting](https://en.wikipedia.org/wiki/Cycle_detection) [garbage collector](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)) and support for [Unicode](https://en.wikipedia.org/wiki/Unicode).

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely [backward-compatible](https://en.wikipedia.org/wiki/Backward_compatibility). Many of its major features were [backported](https://en.wikipedia.org/wiki/Backporting) to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2 to 3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's [end-of-life](https://en.wikipedia.org/wiki/End-of-life_(product)) date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. No more security patches or other improvements will be released for it. With Python 2's [end-of-life](https://en.wikipedia.org/wiki/End-of-life_(product)), only Python 3.6.x  and later are supported.

Python 3.9.2 and 3.8.8 were expeditedas all versions of Python (including 2.7) had security issues, leading to possible [remote code execution](https://en.wikipedia.org/wiki/Remote_code_execution) and [web cache poisoning](https://en.wikipedia.org/wiki/Cache_poisoning).

**Design Philosophy & Feature**

Python is a [multi-paradigm programming language](https://en.wikipedia.org/wiki/Multi-paradigm_programming_language). [Object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) and [structured programming](https://en.wikipedia.org/wiki/Structured_programming) are fully supported, and many of its features support functional programming and [aspect-oriented programming](https://en.wikipedia.org/wiki/Aspect-oriented_programming) (including by [meta-programming](https://en.wikipedia.org/wiki/Metaprogramming) and [meta-objects](https://en.wikipedia.org/wiki/Metaobject) (magic methods)). Many other paradigms are supported via extensions, including [design by contract](https://en.wikipedia.org/wiki/Design_by_contract) and [logic programming](https://en.wikipedia.org/wiki/Logic_programming).

Python uses [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_typing) and a combination of [reference counting](https://en.wikipedia.org/wiki/Reference_counting) and a cycle-detecting garbage collector for [memory management](https://en.wikipedia.org/wiki/Memory_management). It also features dynamic [name resolution](https://en.wikipedia.org/wiki/Name_resolution_(programming_languages)) ([late binding](https://en.wikipedia.org/wiki/Late_binding)), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the [Lisp](https://en.wikipedia.org/wiki/Lisp_(programming_language)) tradition. It has filter, map and reduce functions;  [list comprehensions](https://en.wikipedia.org/wiki/List_comprehension), [dictionaries](https://en.wikipedia.org/wiki/Associative_array), sets, and [generator](https://en.wikipedia.org/wiki/Generator_(computer_programming)) expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language)) and [Standard ML](https://en.wikipedia.org/wiki/Standard_ML).

The language's core philosophy is summarized in the document The [Zen of Python](https://en.wikipedia.org/wiki/Zen_of_Python) (PEP 20), which includes [aphorisms](https://en.wikipedia.org/wiki/Aphorism) such as:

* Beautiful is better than ugly.
* Explicit is better than implicit.
* Simple is better than complex.
* Complex is better than complicated.
* Readability counts.

Rather than having all of its functionality built into its core, Python was designed to be highly [extensible](https://en.wikipedia.org/wiki/Extensibility) (with modules). This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with [ABC](https://en.wikipedia.org/wiki/ABC_(programming_language)), which espoused the opposite approach.

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to [Perl](https://en.wikipedia.org/wiki/Perl)'s "[there is more than one way to do it](https://en.wikipedia.org/wiki/There_is_more_than_one_way_to_do_it)" motto, Python embraces a "there should be one— and preferably only one —obvious way to do it" design philosophy. [Alex Martelli](https://en.wikipedia.org/wiki/Alex_Martelli), a [Fellow](https://en.wikipedia.org/wiki/Fellow) at the [Python Software Foundation](https://en.wikipedia.org/wiki/Python_Software_Foundation) and Python book author, writes that "To describe something as 'clever' is not considered a compliment in the Python culture."

Python's developers strive to avoid [premature optimization](https://en.wikipedia.org/wiki/Premature_optimization), and reject patches to non-critical parts of the [C-Python](https://en.wikipedia.org/wiki/CPython) reference implementation that would offer marginal increases in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use [PyPy](https://en.wikipedia.org/wiki/PyPy), a [just-in-time compiler](https://en.wikipedia.org/wiki/Just-in-time_compilation). [Cython](https://en.wikipedia.org/wiki/Cython) is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

Python's developers aim to keep the language fun to use. This is reflected in its name a tribute to the British comedy group [Monty Python](https://en.wikipedia.org/wiki/Monty_Python) and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (a reference to a [Monty Python sketch](https://en.wikipedia.org/wiki/Spam_(Monty_Python))) instead of the standard [foo and bar](https://en.wikipedia.org/wiki/Foobar).

A common [neologism](https://en.wikipedia.org/wiki/Neologism) in the Python community is pythonic, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called unpythonic.

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonistas

**Syntax and Semantics :**

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use [curly brackets](https://en.wikipedia.org/wiki/Curly_bracket_programming_language) to delimit blocks, and semicolons after statements are allowed but are rarely, if ever, used. It has fewer syntactic exceptions and special cases than [C](https://en.wikipedia.org/wiki/C_(programming_language)) or [Pascal](https://en.wikipedia.org/wiki/Pascal_(programming_language)).

**Indentation :**

Main article: [Python syntax and semantics & Indentation](https://en.wikipedia.org/wiki/Python_syntax_and_semantics#Indentation)

Python uses [whitespace](https://en.wikipedia.org/wiki/Whitespace_character) indentation, rather than [curly brackets](https://en.wikipedia.org/wiki/Curly_bracket_programming_language) or keywords, to delimit [blocks](https://en.wikipedia.org/wiki/Block_(programming)). An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is sometimes termed the [off-side rule](https://en.wikipedia.org/wiki/Off-side_rule), which some other languages share, but in most languages indentation does not have any semantic meaning. The recommended indent size is four spaces.

**Statements and control flow :**

Python's [statements](https://en.wikipedia.org/wiki/Statement_(computer_science)) include:

* The [assignment](https://en.wikipedia.org/wiki/Assignment_(computer_science)) statement, using a single equals sign =.
* The if statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if).
* The for statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.
* The while statement, which executes a block of code as long as its condition is true.
* The Try statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in a finally block will always be run regardless of how the block exits.
* The raise statement, used to raise a specified exception or re-raise a caught exception.
* The class statement, which executes a block of code and attaches its local namespace to a [class](https://en.wikipedia.org/wiki/Class_(computer_science)), for use in object-oriented programming.
* The def statement, which defines a [function](https://en.wikipedia.org/wiki/Function_(computing)) or [method](https://en.wikipedia.org/wiki/Method_(computing)).
* The with statement, which encloses a code block within a context manager (for example, acquiring a [lock](https://en.wikipedia.org/wiki/Lock_(computer_science)) before the block of code is run and releasing the lock afterwards, or opening a [file](https://en.wikipedia.org/wiki/Computer_file) and then closing it), allowing [resource-acquisition-is-initialization](https://en.wikipedia.org/wiki/Resource_acquisition_is_initialization) (RAII) - like behavior and replaces a common try/finally idiom.
* The break statement, exits from a loop.
* The continue statement, skips this iteration and continues with the next item.
* The del statement, removes a variable, which means the reference from the name to the value is deleted and trying to use that variable will cause an error. A deleted variable can be reassigned.
* The pass statement, which serves as a [NOP](https://en.wikipedia.org/wiki/NOP_(code)). It is syntactically needed to create an empty code block.
* The assert statement, used during debugging to check for conditions that should apply.
* The yield statement, which returns a value from a [generator](https://en.wikipedia.org/wiki/Generator_(computer_programming)#Python) function and yield is also an operator. This form is used to implement [co-routines](https://en.wikipedia.org/wiki/Coroutine).
* The return statement, used to return a value from a function.
* The import statement, which is used to import modules whose functions or variables can be used in the current program.

The assignment statement (=) operates by binding a name as a [reference](https://en.wikipedia.org/wiki/Pointer_(computer_programming)) to a separate, dynamically-allocated [object](https://en.wikipedia.org/wiki/Object_(computer_science)). Variables may be subsequently rebound at any time to any object. In Python, a variable name is a generic reference holder and does not have a fixed [data type](https://en.wikipedia.org/wiki/Type_system) associated with it. However, at a given time, a variable will refer to some object, which will have a type. This is referred to as [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_type) and is contrasted with [statically-typed](https://en.wikipedia.org/wiki/Statically-typed) programming languages, where each variable may only contain values of a certain type.

Python does not support [tail call](https://en.wikipedia.org/wiki/Tail_call) optimization or [first-class continuations](https://en.wikipedia.org/wiki/First-class_continuations), and, according to Guido van Rossum, it never will.[[80]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-55-80)[[81]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-56-81) However, better support for [co-routine](https://en.wikipedia.org/wiki/Coroutine)-like functionality is provided, by extending Python's [generators](https://en.wikipedia.org/wiki/Generator_(computer_programming)). Before 2.5, generators were [lazy](https://en.wikipedia.org/wiki/Lazy_evaluation) [iterators](https://en.wikipedia.org/wiki/Iterator); information was passed uni-directionally out of the generator. From Python 2.5, it is possible to pass information back into a generator function, and from Python 3.3, the information can be passed through multiple stack levels.

**Expressions** :

Some Python [expressions](https://en.wikipedia.org/wiki/Expression_(computer_science)) are similar to those found in languages such as C and [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), while some are not:

* Addition, subtraction, and multiplication are the same, but the behavior of division differs. There are two types of divisions in Python. They are floor division (or integer division) // and floating-point/division. Python also uses the \*\* operator for exponentiation.
* From Python 3.5, the new @ infix operator was introduced. It is intended to be used by libraries such as [NumPy](https://en.wikipedia.org/wiki/NumPy) for [matrix multiplication](https://en.wikipedia.org/wiki/Matrix_multiplication).
* From Python 3.8, the syntax :=, called the 'walrus operator' was introduced. It assigns values to variables as part of a larger expression.
* In Python, == compares by value, versus Java, which compares numerics by value and objects by reference. (Value comparisons in Java on objects can be performed with the equals() method.) Python's is operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example A<=B<=C.
* Python uses the words and, or, not for or its boolean operators rather than the symbolic &&, ||, ! used in Java and C.
* Python has a type of expression termed a [list comprehension](https://en.wikipedia.org/wiki/List_comprehension#Python) as well as a more general expression termed a [generator](https://en.wikipedia.org/wiki/Generator_(computer_programming)) expression.
* [Anonymous functions](https://en.wikipedia.org/wiki/Anonymous_function) are implemented using [lambda expressions](https://en.wikipedia.org/wiki/Lambda_(programming)); however, these are limited in that the body can only be one expression.
* Conditional expressions in Python are written as x if c else y (different in order of operands from the c ? x : y operator common to many other languages).
* Python makes a distinction between [lists](https://en.wikipedia.org/wiki/List_(computer_science)) and [tuples](https://en.wikipedia.org/wiki/Tuple). Lists are written as [1, 2, 3], are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be [immutable](https://en.wikipedia.org/wiki/Immutable) in Python). Tuples are written as (1, 2, 3), are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The + operator can be used to concatenate two tuples, which does not directly modify their contents, but rather produces a new tuple containing the elements of both provided tuples. Thus, given the variable t initially equal to (1, 2, 3), executing t = t + (4, 5) first evaluates t + (4, 5), which yields (1, 2, 3, 4, 5), which is then assigned back to t, thereby effectively "modifying the contents" of t, while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.
* Python features sequence unpacking wherein multiple expressions, each evaluating to anything that can be assigned to (a variable, a writable property, etc.), are associated in an identical manner to that forming tuple literals and, as a whole, are put on the left-hand side of the equal sign in an assignment statement. The statement expects an iterable object on the right-hand side of the equal sign that produces the same number of values as the provided writable expressions when iterated through and will iterate through it, assigning each of the produced values to the corresponding expression on the left.
* Python has a "string format" operator %. This functions analogously ton printf format strings in C, e.g. “spam=%s eggs=%d” % (“blah”,2) evaluates to “spam=blah eggs=2”. In Python 3 and 2.6+, this was supplemented by the format() method of the str class, e.g. “spam={0} eggs={1}”.format(“blah”,2). Python 3.6 added "f-strings": blah = “blah”; eggs = 2; f‘spam={blah} eggs={eggs}’
* Strings in Python can be [concatenated](https://en.wikipedia.org/wiki/Concatenation), by "adding" them (same operator as for adding integers and floats). E.g. “spam” + “eggs” returns “spameggs”. Even if your strings contain numbers, they are still added as strings rather than integers. E.g. “2” + “2” returns “2”.
* Python has various kinds of [string literals](https://en.wikipedia.org/wiki/String_literal):
  + Strings delimited by single or double quote marks. Unlike in [Unix shells](https://en.wikipedia.org/wiki/Unix_shell), [Perl](https://en.wikipedia.org/wiki/Perl) and Perl-influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (\) as an [escape character](https://en.wikipedia.org/wiki/Escape_character). [String interpolation](https://en.wikipedia.org/wiki/String_interpolation) became available in Python 3.6 as "formatted string literals".
  + Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like [here documents](https://en.wikipedia.org/wiki/Here_document) in shells, Perl and [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)).
  + [Raw string](https://en.wikipedia.org/wiki/Raw_string) varieties, denoted by prefixing the string literal with an r. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as [regular expressions](https://en.wikipedia.org/wiki/Regular_expression) and [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows)-style paths. Compare "@-quoting" in [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)).
* Python has [array index](https://en.wikipedia.org/wiki/Array_index) and [array slicing](https://en.wikipedia.org/wiki/Array_slicing) expressions on lists, denoted as a[Key], a[start:stop] or a[start:stop:step]. Indexes are [zero-based](https://en.wikipedia.org/wiki/Zero-based_numbering), and negative indexes are relative to the end. Slices take elements from the start index up to, but not including, the stop index. The third slice parameter, called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted, for example a[:] returns a copy of the entire list. Each element of a slice is a [shallow copy](https://en.wikipedia.org/wiki/Shallow_copy).

In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as [Common Lisp](https://en.wikipedia.org/wiki/Common_Lisp), [Scheme](https://en.wikipedia.org/wiki/Scheme_(programming_language)), or [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)). This leads to duplicating some functionality. For example:

* [List comprehensions](https://en.wikipedia.org/wiki/List_comprehensions) vs. for-loops
* [Conditional](https://en.wikipedia.org/wiki/Conditional_(programming)) expressions vs. if blocks
* The eval() vs. exec() built-in functions (in Python 2, exec is a statement); the former is for expressions, the latter is for statements.

Statements cannot be a part of an expression, so list and other comprehensions or [lambda expressions](https://en.wikipedia.org/wiki/Lambda_(programming)), all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as a=1 cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator = for an equality operator == in conditions: if (c==1) {…} is syntactically valid (but probably unintended) C code but if c=1: … causes a syntax error in Python.

**Methods** :

[Methods](https://en.wikipedia.org/wiki/Method_(programming)) on objects are [functions](https://en.wikipedia.org/wiki/Function_(programming)) attached to the object's class; the syntax instance.method(argument) is, for normal methods and functions, [syntactic sugar](https://en.wikipedia.org/wiki/Syntactic_sugar) for Class.method(instance, argument). Python methods have an explicit self parameter access [instance data](https://en.wikipedia.org/wiki/Instance_data), in contrast to the implicit self (or this) in some other object-oriented programming languages (e.g., [C++](https://en.wikipedia.org/wiki/C%2B%2B), Java, [Objective-C](https://en.wikipedia.org/wiki/Objective-C), or [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language))). Apart from this Python also provides methods, sometimes called d-under methods due to their names beginning and ending with double-underscores, to extend the functionality of custom class to support native functions such as print, length, comparison, support for arithmetic operations, type conversion, and many more.

### Typing :

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically-typed, Python is strongly-typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Python allows programmers to define their own types using [classes](https://en.wikipedia.org/wiki/Class_(computer_science)), which are most often used for object-oriented programming. New instances of classes are constructed by calling the class (for example, SpamClass() or EggsClass()), and the classes are instances of the metaclass type (itself an instance of itself), allowing meta-programming and reflection.

Before version 3.0, Python had two kinds of classes: old-style and new-style.The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from object and are instances of type). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

The long-term plan is to support gradual typing and from Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named mypy supports compile-time type checking.

**LIST OF MODULES**

1. Data Preparation

2. Implementing CNN architecture

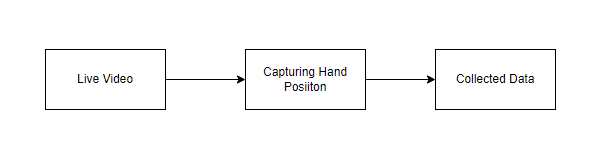
3. Prediction of Sign Gesture.

**MODULE DESCRIPTION**

**1. Data Preparation:**

The dataset collection is the very important part of our project. The collection of the dataset is based on the hand posture. Using open CV we can access the live videos. With the help of open CV we record the videos frame by frame. First we record the videos one by one. And save the hand position data as a image in a folder. We repeat this process for the gestures which are needed for us.

**Module Diagram:**



**Input :** Video capture

**Output :** Data Collection

**2. Implementing CNN architecture:**

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. Their network consists of four layers with 1,024 input units, 256 units in the first hidden layer, eight units in the second hidden layer, and two output units.

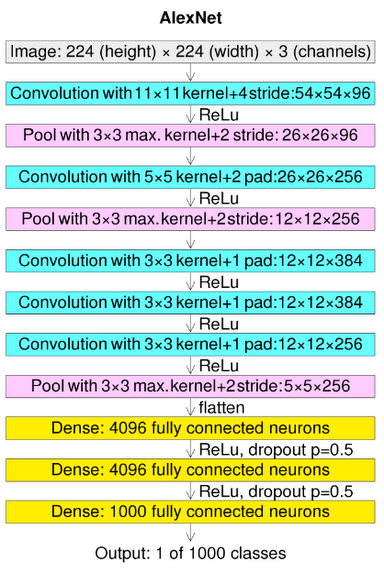
Input Layer:

Input layer in CNN contain image data. Image data is represented by three dimensional matrixes. It needs to reshape it into a single column. Suppose you have image of dimension 28 x 28 =784, it need to convert it into 784 x 1 before feeding into input.

AlexNet is the name of a convolutional neural network which has had a large impact on the field of machine learning, specifically in the application of deep learning to machine vision. AlexNet was the first [convolutional network](https://www.mygreatlearning.com/blog/cnn-model-architectures-and-applications/) which used GPU to boost performance.

AlexNet architecture consists of 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and 1 softmax layer. Each convolutional layer consists of convolutional filters and a nonlinear activation function ReLU. The pooling layers are used to perform max pooling.

**Architecture of AlexNet:**



Architecture of AlexNet

**Convolutional layers:**

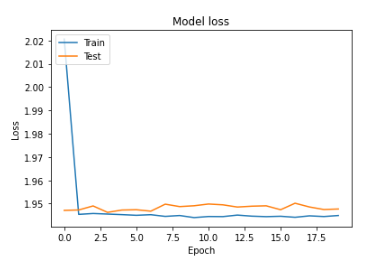
Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

**Pooling layers:**

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

**Dense or Fully connected layers:**

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

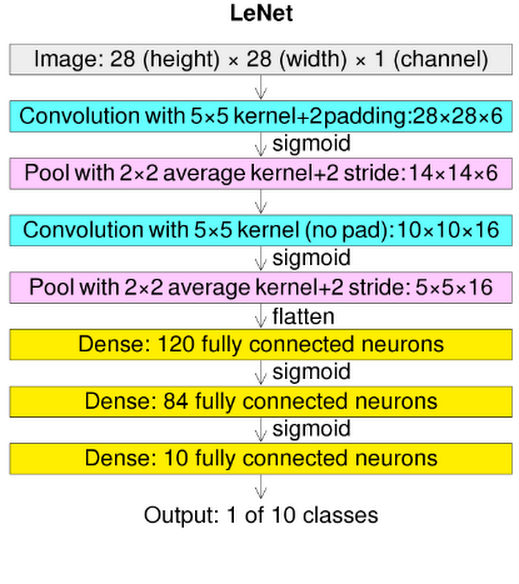


**LENET:**

LeNet was one among the earliest [convolutional neural networks](https://www.analyticssteps.com/blogs/how-transfer-learning-done-neural-networks-and-convolutional-neural-networks) which promoted the event of deep learning. After innumerous years of analysis and plenty of compelling iterations, the end result was named LeNet.

## **Architecture of LeNet-5:**

LeNet-5 CNN architecture is made up of 7 layers. The layer composition consists of 3 convolutional layers, 2 subsampling layers and 2 fully connected layers.



Architecture of LeNet

**Convolutional layers:**

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

**Pooling layers:**

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

**Dense or Fully connected layers:**

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

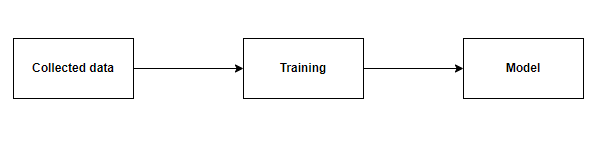
## **Softmax / Logistic Layer:**

Softmax or Logistic layer is the last layer of CNN. It resides at the end of FC layer. Logistic is used for binary classification and softmax is for multi-classification.

## **Output Layer:**

Output layer contains the label which is in the form of one-hot encoded. Now you have a good understanding of CNN.

**Module Diagram:**



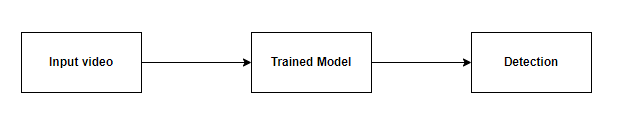
**Input :** Data

**Output :** Trained Model

**3. Prediction of hand Gesture:**

After training the model we have to use the trained model to use for deployment. In this deployment model we use opencv to access the live video. It capture the videos and it will help to recognize the pattern and make it sequential data and those recognized data will flow through our trained model. Finally the model will predict the gesture and it will display the text based on the sign gesture.

**Module Diagram:**



**Input :** Input video

**Output :** Detection

**CODING:**

**MODULE – 1:**

# Manual Network

# Import the necessary libraries.

import numpy as np

import matplotlib.pyplot as plt

import os

# Import the necessary libraries.

import tensorflow as tf

import glob

from PIL import Image

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, Dense, MaxPooling2D, Flatten, Convolution2D, Dropout, BatchNormalization

# Ignoring the warings

import warnings

warnings.filterwarnings("ignore")

dir\_name\_train\_with\_helmet = 'datasets/train/with\_helmet'

dir\_name\_train\_no\_helmet = 'datasets/train/no\_helmet'

def plot\_images(item\_dir, n=5):

all\_item\_dir = os.listdir(item\_dir)

item\_files = [os.path.join(item\_dir, file) for file in all\_item\_dir][:n]

plt.figure(figsize=(35, 10))

for idx, img\_path in enumerate(item\_files):

plt.subplot(2, n, idx+1)

img = plt.imread(img\_path)

plt.imshow(img, cmap='gray')

plt.axis('off')

plt.tight\_layout()

def Images\_details\_Print\_data(data, path):

print(" ====== Images in: ", path)

for k, v in data.items():

print("%s:\t%s" % (k, v))

def Images\_details(path):

files = [f for f in glob.glob(path + "\*\*/\*.\*", recursive=True)]

data = {}

data['images\_count'] = len(files)

data['min\_width'] = 10\*\*100 # No image will be bigger than that

data['max\_width'] = 0

data['min\_height'] = 10\*\*100 # No image will be bigger than that

data['max\_height'] = 0

for f in files:

im = Image.open(f)

width, height = im.size

data['min\_width'] = min(width, data['min\_width'])

data['max\_width'] = max(width, data['max\_height'])

data['min\_height'] = min(height, data['min\_height'])

data['max\_height'] = max(height, data['max\_height'])

Images\_details\_Print\_data(data, path)

print("")

print("Trainned data for :")

print("")

Images\_details(dir\_name\_train\_with\_helmet)

print("")

plot\_images(dir\_name\_train\_with\_helmet)

print("")

print("Trainned data for :")

print("")

Images\_details(dir\_name\_train\_no\_helmet)

print("")

plot\_images(dir\_name\_train\_no\_helmet)

### Data augmentation

train\_datagen=ImageDataGenerator(rescale=1./255, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True)

test\_datagen=ImageDataGenerator(rescale=1./255)

# having train and test images

training\_set=train\_datagen.flow\_from\_directory('datasets/train', batch\_size=32, class\_mode='categorical', target\_size=(255,255))

testing\_set=test\_datagen.flow\_from\_directory('datasets/test', batch\_size=32, class\_mode='categorical', target\_size=(255,255))

## Creating cnn layer

# initialized the model

model=Sequential()

# Adding first convolution layer

model.add(Convolution2D(32,(3,3),input\_shape=(255,255,3), padding='same', activation='relu'))

# Adding first Maxpool layer

model.add(MaxPooling2D(pool\_size=(2,2)))

# Adding Flatten layer

model.add(Flatten())

# Adding input layer

model.add(Dense(units=64, activation='relu', ))

# Adding dropout layer

model.add(Dropout(0.4))

# Adding output layer

model.add(Dense(units=2, activation='softmax'))

model.summary()

# Compilin the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Fitting the model

history= model.fit(training\_set,

steps\_per\_epoch= training\_set.samples//32,

validation\_data=testing\_set,

validation\_steps= testing\_set.samples//32,

epochs=10)

def plot():

# Plot training & validation accuracy values

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

# Plot training & validation loss values

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

plot()

**MODULE – 2:**

# Alexnet Architecture

# Import the neccessary Libraries.

import numpy as np

import matplotlib.pyplot as plt

# Import the neccesary Libraries.

import warnings

warnings.filterwarnings("ignore")

# Import the neccesary packages.

import glob

import tensorflow as tf

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dropout

from tensorflow.keras.layers import Activation

from tensorflow.keras.layers import Convolution2D

from tensorflow.keras.layers import MaxPool2D

from tensorflow.keras.layers import Conv2D

from tensorflow.keras.layers import BatchNormalization

from tensorflow.keras.layers import MaxPooling2D

from tensorflow.keras.models import Sequential

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.callbacks import ModelCheckpoint

# Data Augumentation.

train\_data\_gen = ImageDataGenerator(rescale=1./255, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True)

test\_data\_gen = ImageDataGenerator(rescale= 1./255)

# Splitting train and test

training\_set=train\_data\_gen.flow\_from\_directory('datasets/train',target\_size=(224,224),batch\_size=32,class\_mode='categorical')

test\_set=test\_data\_gen.flow\_from\_directory('datasets/test',target\_size=(224,224),batch\_size=32,class\_mode='categorical')

# Create a sequential model

model = Sequential()

# 1st Convolutional Layer

model.add(Conv2D(filters=96, input\_shape=(224,224,3), kernel\_size=(11,11),\

strides=(4,4), padding='valid'))

model.add(Activation('relu'))

# Pooling

model.add(MaxPooling2D(pool\_size=(2,2), strides=(2,2), padding='valid'))

# Batch Normalisation before passing it to the next layer

model.add(BatchNormalization())

# 2nd Convolutional Layer

model.add(Conv2D(filters=256, kernel\_size=(11,11), strides=(1,1), padding='valid'))

model.add(Activation('relu'))

# Pooling

model.add(MaxPooling2D(pool\_size=(2,2), strides=(2,2), padding='valid'))

# Batch Normalisation

model.add(BatchNormalization())

# 3rd Convolutional Layer

model.add(Conv2D(filters=384, kernel\_size=(3,3), strides=(1,1), padding='valid'))

model.add(Activation('relu'))

# Batch Normalisation

model.add(BatchNormalization())

# 4th Convolutional Layer

model.add(Conv2D(filters=384, kernel\_size=(3,3), strides=(1,1), padding='valid'))

model.add(Activation('relu'))

# Batch Normalisation

model.add(BatchNormalization())

# 5th Convolutional Layer

model.add(Conv2D(filters=256, kernel\_size=(3,3), strides=(1,1), padding='valid'))

model.add(Activation('relu'))

# Pooling

model.add(MaxPooling2D(pool\_size=(2,2), strides=(2,2), padding='valid'))

# Batch Normalisation

model.add(BatchNormalization())

# Passing it to a dense layer

model.add(Flatten())

# 1st Dense Layer

model.add(Dense(4096, input\_shape=(224\*224\*3,)))

model.add(Activation('relu'))

# Add Dropout to prevent overfitting

model.add(Dropout(0.4))

# Batch Normalisation

model.add(BatchNormalization())

# 2nd Dense Layer

model.add(Dense(4096))

model.add(Activation('relu'))

# Add Dropout

model.add(Dropout(0.4))

# Batch Normalisation

model.add(BatchNormalization())

# 3rd Dense Layer

model.add(Dense(1000))

model.add(Activation('relu'))

# Add Dropout

model.add(Dropout(0.4))

# Batch Normalisation

model.add(BatchNormalization())

# Output Layer

model.add(Dense(2))

model.add(Activation('softmax'))

# Compile the model

model.compile(loss = 'categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()

epochs = 40

batch\_size = 32

mc = ModelCheckpoint('ALEXNET.h5', monitor = 'accuracy', verbose=1, save\_best\_only = True)

#### Fitting the model

History = model.fit(training\_set, steps\_per\_epoch=training\_set.samples // batch\_size, epochs=epochs,

validation\_data=test\_set,validation\_steps=test\_set.samples // batch\_size, callbacks=[mc])

def plot():

# Plot training & validation accuracy values

plt.plot(History.history['accuracy'])

plt.plot(History.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

# Plot training & validation loss values

plt.plot(History.history['loss'])

plt.plot(History.history['val\_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

plot()

**MODULE – 3:**

# LENET

# LeNet Architecture

# Import the warnings

import warnings

warnings.filterwarnings("ignore")

# Import the neccesary Packages.

import os

import glob

import numpy as np

import pandas as pd

from PIL import Image

import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.layers import Activation

from tensorflow.keras.callbacks import ModelCheckpoint

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D

from tensorflow.keras.layers import Convolution2D

from tensorflow.keras.layers import Dropout

from tensorflow.keras.layers import MaxPool2D

from tensorflow.keras.layers import MaxPooling2D

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dense

train\_datagen=ImageDataGenerator(rescale=1./255,shear\_range=0.2,zoom\_range=0.2,horizontal\_flip=True)

training\_set=train\_datagen.flow\_from\_directory('datasets/train',target\_size=(224,224),batch\_size=32,class\_mode='categorical')

test\_datagen=ImageDataGenerator(rescale=1./255)

test\_set=test\_datagen.flow\_from\_directory('datasets/test',target\_size=(224,224),batch\_size=32,class\_mode='categorical')

Classifier=Sequential()

Classifier.add(Convolution2D(32,3,3,input\_shape=(224,224,3),activation='relu'))

Classifier.add(MaxPooling2D(pool\_size=(2,2)))

Classifier.add(Convolution2D(128,3,3,activation='relu'))

Classifier.add(MaxPooling2D(pool\_size=(2,2)))

Classifier.add(Flatten())

Classifier.add(Dense(256, activation='relu'))

Classifier.add(Dense(2, activation='softmax'))

Classifier.compile(optimizer='rmsprop',loss='categorical\_crossentropy',metrics=['accuracy'])

Classifier.summary()

# Set the file path

model\_path = "LeNet1.h5"

callbacks = [

ModelCheckpoint(model\_path, monitor='accuracy', verbose=1, save\_best\_only=True)

]

epochs = 100

batch\_size = 32

#### Fitting the model

history = Classifier.fit(

training\_set, steps\_per\_epoch=training\_set.samples // batch\_size,

epochs=epochs,

validation\_data=test\_set,validation\_steps=test\_set.samples // batch\_size,

callbacks=callbacks)

plt.figure(figsize=(20, 8))

plt.plot(history.history['accuracy'])

for i in range(epochs):

if i%5 == 0:

plt.annotate(np.round(history.history['accuracy'][i]\*100,2),xy=(i,history.history['accuracy'][i]))

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.show()

plt.figure(figsize=(20, 8))

plt.plot(history.history['loss'])

for i in range(epochs):

if i%5 == 0:

plt.annotate(np.round(history.history['loss'][i]\*100,2),xy=(i,history.history['loss'][i]))

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.show()

**MODULE – 4:**

import cv2

import numpy as np

import os

import imutils

from tensorflow.keras.models import load\_model

os.environ['TF\_FORCE\_GPU\_ALLOW\_GROWTH'] = 'true'

net = cv2.dnn.readNet("yolov3-custom\_7000.weights", "yolov3-custom.cfg")

net.setPreferableBackend(cv2.dnn.DNN\_BACKEND\_CUDA)

net.setPreferableTarget(cv2.dnn.DNN\_TARGET\_CUDA)

model = load\_model('helmet-nonhelmet\_cnn.h5')

print('model loaded!!!')

cap = cv2.VideoCapture('video.mp4')

COLORS = [(0,255,0),(0,0,255)]

layer\_names = net.getLayerNames()

output\_layers = [layer\_names[i - 1] for i in net.getUnconnectedOutLayers()]

fourcc = cv2.VideoWriter\_fourcc(\*"XVID")

writer = cv2.VideoWriter('output.avi', fourcc, 5,(888,500))

def helmet\_or\_nohelmet(helmet\_roi):

    try:

        helmet\_roi = cv2.resize(helmet\_roi, (224, 224))

        helmet\_roi = np.array(helmet\_roi,dtype='float32')

        helmet\_roi = helmet\_roi.reshape(1, 224, 224, 3)

        helmet\_roi = helmet\_roi/255.0

        return int(model.predict(helmet\_roi)[0][0])

    except:

            pass

ret = True

while ret:

    ret, img = cap.read()

    img = imutils.resize(img,height=500)

    # img = cv2.imread('test.png')

    height, width = img.shape[:2]

    blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

    net.setInput(blob)

    outs = net.forward(output\_layers)

    confidences = []

    boxes = []

    classIds = []

    for out in outs:

        for detection in out:

            scores = detection[5:]

            class\_id = np.argmax(scores)

            confidence = scores[class\_id]

            if confidence > 0.3:

                center\_x = int(detection[0] \* width)

                center\_y = int(detection[1] \* height)

                w = int(detection[2] \* width)

                h = int(detection[3] \* height)

                x = int(center\_x - w / 2)

                y = int(center\_y - h / 2)

                boxes.append([x, y, w, h])

                confidences.append(float(confidence))

                classIds.append(class\_id)

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

    for i in range(len(boxes)):

        if i in indexes:

            x,y,w,h = boxes[i]

            color = [int(c) for c in COLORS[classIds[i]]]

            # green --> bike

            # red --> number plate

            if classIds[i]==0: #bike

                helmet\_roi = img[max(0,y):max(0,y)+max(0,h)//4,max(0,x):max(0,x)+max(0,w)]

            else: #number plate

                x\_h = x-60

                y\_h = y-350

                w\_h = w+100

                h\_h = h+100

                cv2.rectangle(img, (x, y), (x + w, y + h), color, 7)

                # h\_r = img[max(0,(y-330)):max(0,(y-330 + h+100)) , max(0,(x-80)):max(0,(x-80 + w+130))]

                if y\_h>0 and x\_h>0:

                    h\_r = img[y\_h:y\_h+h\_h , x\_h:x\_h +w\_h]

                    c = helmet\_or\_nohelmet(h\_r)

                    cv2.putText(img,['helmet','no-helmet'][c],(x,y-100),cv2.FONT\_HERSHEY\_SIMPLEX,2,(0,255,0),2)

                    cv2.rectangle(img, (x\_h, y\_h), (x\_h + w\_h, y\_h + h\_h),(255,0,0), 10)

    writer.write(img)

    cv2.imshow("Image", img)

    if cv2.waitKey(1) == 27:

        break

writer.release()

cap.release()

cv2.waitKey(0)

cv2.destroyAllWindows()

**Conclusion:**

It focused how image from given dataset (trained dataset) in field and past data set used predict the pattern of different gesture using NN model. This brings some of the following different live sign prediction. We applied NN where the accuracy and loss of the neural network makes better classification and the .h5 file is taken from there and that is deployed in real time.

**Future Work:**

* To make connect two way communication.
* Deploying it in an application.