IMPLEMENTATION OF CNN BASED WORD RECOGNITION SYSTEM

A
MAJOR PROJECT REPORT

Submitted in the partial Fulfilment of the requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted By

1. SAI KIRAN BAGLI	19R21A0462
2. SATYA PRAKASH DASH	19R21A04B0
3. BHAIDISHETTY SHASHANTH	19R21A0463
4. PRADEEP OALATI	19R21A0497

UNDER THE GUIDANCE OF

Dr. Bittu Kumar
Assistant Professor

MARRI LAXMAN REDDY GROUP OF INSTITUTIONS

MLR Institute of Technology
(Autonomous)
(Affiliated to JNTUH, Hyderabad)
Dundigal, Hyderabad-500043

2019-2023

MLR Institute of Technology

(Autonomous)

(Affiliated to JNTUH, Hyderabad)

Dundigal, Hyderabad-500043

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the project *entitled "**Implementation of CNN based Word Recognition System**" is the bonafied work done by Pradeep Oalati-19R21A0497* in partial fulfilment of the requirement for the award of the degree of B.Tech in Electronics and Communication Engineering, during the academic year 2022 - 23.

Internal Guide **Head of the Department**

External Examiner

ACKNOWLEDGEMENT

We express our profound thanks to the management of **MLR Institute of Technology**, Dundigal, Hyderabad, for supporting us to complete this project.

We take immense pleasure in expressing our sincere thanks to **Dr.K.Srinivasa Rao**, Principal, MLR Institute of Technology, for his kind support and encouragement.

We are very much grateful to **Dr S.V.S Prasad**, Professor & Head of the Department, MLR Institute of Technology, for encouraging us with his valuable suggestions.

We are very much grateful to **Dr. Bittu Kumar,Assistant Professor** for his unflinching cooperation throughout the project.

We would like to express our sincere thanks to the teaching and non teaching faculty members of ECE Dept., MLR Institute of Technology, who extended their help to us in making our project work successful.

Project associates:

Sai Kiran Bagli	19R21A0462
Satya Prakash Dash	19R21A04B0
Shashanth Bhaidishetty	19R21A0463
Pradeep Oalati	19R21A0497

ABSTRACT

Sign language is used by deaf and hard hearing people to exchange information between their own community and other people. Computer recognition of Sign language deals from Sign gesture acquisition and continues till text/speech generation. Sign gestures can be classified as static and dynamic. However static gesture recognition is simpler than dynamic gesture recognition but both recognition systems are important to the human community. However, most people do not know the sign language. In this project, we aim to automatically recognize sign language Alphabet / Numbers / Words in image-based methodology using PYTHON TENSORFLOW.

This system expects to achieve recognizing gestures of ISL (Indian Sign Language) by special people and converting into speech/text. The statistic of the result of the implementation, it is therefore concluded that the method is used for cross-correlation and colour segmentation work with hand gesture recognition. The results obtained are applicable and can be implemented in a mobile device smartphone having a frontal camera. The vision of an efficient system to translate sign language to text is quite achievable, but the challenges lie in optimization.

Visual-based gesture recognition can help to overcome this communication limitation. Recently, several techniques and methods have been proposed in this area of research and showed some improvements. Despite all of the proposed methods, most of hand gesture recognition approaches that have been applied still lack of compatibility and have lots of limitations. For instance, hand segmentation meets the complication of distinguishing the hand and the face region by using skin detection. Motivated by those facts, this paper presents a review and explains progress of feature extraction in sign language recognition mostly in the last ten years. In this contribution, we focus on studying feature extraction methods. Smartphone having a frontal camera. The vision of an efficient system to translate sign language to text is quite achievable, but the challenges lie in optimization.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER – 1

INTRODUCTION

1.1 INTRODUCTION

Image processing allows one to enhance image features of interest while attenuating detail irrelevant to a given application, and then extract useful information about the scene from the enhanced image. An Introduction to Digital Image Processing Bill Silver Chief Technology Officer Cognex Corporation, Modular Vision Systems Division Digital image processing allows one to enhance image features of interest while attenuating detail irrelevant to a given application, and then extract useful information about the scene from the enhanced image. This introduction is a practical guide to the challenges, and the hardware and algorithms used to meet them. Images are produced by a variety of physical devices, including still and video cameras, x-ray devices, electron microscopes, radar, and ultrasound, and used for a variety of purposes, including entertainment, medical, business (e.g., documents), industrial, military, civil (e.g., traffic), security, and scientific.

The goal in each case is for an observer, human or machine, to extract useful information about the scene being imaged. Often the raw image is not directly suitable for this purpose, and must be processed in some way. Such processing is called image enhancement; processing by an observer to extract information is called image analysis. Enhancement and analysis are distinguished by their output, images vs. scene information, and by the challenges faced and methods employed. Image enhancement has been done by chemical, optical, and electronic means, while analysis has been done mostly by humans and electronically. Digital image processing is a subset of the electronic domain wherein the image is converted to an array of small integers, called pixels, representing a physical quantity such as scene radiance, stored in a digital memory, and processed by computer or other digital hardware. Digital image processing, either as enhancement for human observers or performing autonomous analysis, offers advantages in cost, speed, and flexibility, and with the rapidly falling price and rising performance of personal computers it has become the dominant method in use.

1.2 IMAGE

Image can be represented using a two-dimensional function f(x, y) where x and y are the spatial coordinates of each point in the image and the amplitude off at any coordinate is known as the intensity or grey value of that point. Images may be two or three-dimensional, such as

a photograph or screen display, or three-dimensional, such as a statue or hologram. They may be captured by optical devices such as cameras, mirrors, lenses, telescopes, microscopes, and natural objects and phenomena, such as the human eye or water. The word 'image' is also used in the broader sense of any two-dimensional figure such as a map, graph, pie chart, painting or banner. In this wider sense, images can also be rendered manually, such as by drawing, the art of painting, carving, rendered automatically by printing or computer graphics technology, or developed by a combination of methods. An image does not have to use the entire visual system to be a visual representation. A popular example of this is of a greyscale image, which uses the visual system's sensitivity to brightness across all wavelengths, without taking into account different colours. A black and white visual representation of something is still an image, even though it does not make full use of the visual system's capabilities. Images are typically still, but in some cases can be moving or animated.

1.3 ANALOG IMAGE

Analog image processing is done on analog signals. It includes processing on two dimensional analog signals. In this type of processing, the images are manipulated by electrical means by varying the electrical signal. The common example include is the television image. Digital image processing has dominated over analog image processing with the passage of time due its wider range of applications.

1.4 DIGITAL IMAGE

We can describe a digital image as a finite set of digital values, called pixels. Pixels are the smallest individual element of an image, holding values that represent the brightness of a given color at any specific point. So, we can think of an image as a matrix (or a two-dimensional array) of pixels which contains a fixed number of rows and columns.

Fig. 1.1 Digital Image

1.5 DIGITAL IMAGE PROCESSING

Digital image processing, the manipulation of images by computer, is relatively recent development in terms of man's ancient fascination with visual stimuli. In its short history, it has been applied to practically every type of images with varying degree of success.

1.5.1 REPRESENTATION OF IMAGE

Image representations are based on integer samples and thus evaluation by established indices such as the peak signal to noise ratio (PSNR) is, at least in principle, possible. In a 24-bit color image representation, each pixel is recoded with red, green, and blue (RGB) components. With each component value encoded in 8 bits, resulting in 24 bits in total, we achieve a full colour RGB image.

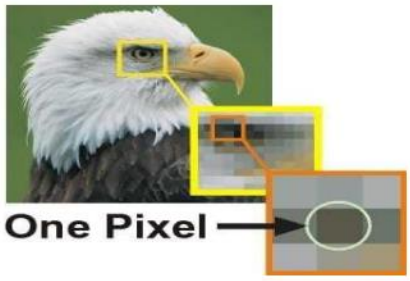


Fig. 1.2 Image Representation in Pixel Form

1.5.2 RELATION BETWEEN DIGITAL IMAGE AND A SIGNAL

If the image is a two-dimensional array, then what does it have to do with a signal? In order to understand that, we need to first understand what is a signal?

1.5.2.1 SIGNAL

In physical world, any quantity measurable through time over space or any higher dimension can be taken as a signal. A signal is a mathematical function, and it conveys some information. A signal can be one dimensional or two dimensional or higher dimensional signal. One dimensional signal is a signal that is measured over time. The common example is a voice signal. The two-dimensional signals are those that are measured over some other physical quantities. The example of two-dimensional signal is a digital image. We will look in more detail in the next tutorial of how a one dimensional or two dimensional single and higher signals are formed and interpreted.

1.5.2.2 RELATION

Since anything that conveys information or broadcast a message in physical world between two observers is a signal. That includes speech or (human voice) or an image as a signal. Since when we speak, our voice is converted to a sound wave/signal and transformed with respect to the time to person we are speaking to. Not only this , but the way a digital camera works, as while acquiring an image from a digital camera involves transfer of a signal from one part of the system to the other.

1.5.3 FORMATION OF DIGITAL IMAGE

Since capturing an image from a camera is a physical process. The sunlight is used as a source of energy. A sensor array is used for the acquisition of the image. So, when the sunlight falls upon the object, then the amount of light reflected by that object is sensed by the sensors, and a continuous voltage signal is generated by the amount of sensed data. In order to create a digital image, we need to convert this data into a digital form. This involves sampling and quantization. (They are discussed later on). The result of sampling and quantization results in a two-dimensional array or matrix of numbers which are nothing but a digital image.

1.5.4 COMPONENTS OF DIGITAL IMAGE PROCESSING

As recently as the mid-1980s, numerous models of image processing systems being sold throughout the world were rather substantial peripheral devices that attached to equally substantial host computers. Late in the 1980s and early in the 1990s, the market shifted to image processing hardware in the form of single boards designed to be compatible with industry standard buses and to fit into engineering workstation cabinets and personal computers. In addition to lowering costs, this market shift also served as a catalyst for a significant number of new companies whose specialty is the development of software written specifically for image processing. Although large-scale image processing systems still are being sold for massive imaging applications, such as processing of satellite images, the trend continues toward miniaturizing and blending of general-purpose small computers with specialized image processing hardware. cabinets and personal computers. In addition to lowering costs, this market shift also served as a catalyst for a significant number of new companies whose specialty is the development of software written specifically for image processing. Although

large-scale image processing systems still are being sold for massive imaging applications, such as processing of satellite images, the trend continues toward miniaturizing and blending of general-purpose small computers with specialized image processing hardware.

The basic components comprising a typical general-purpose system used for digital image processing . The function of each component is discussed in the following paragraphs, starting with image sensing. With reference to sensing, two elements are required to acquire digital images. The first is a physical device that is sensitive to the energy radiated by the object we wish to image. The second, called a digitizer, is a device for converting the output of the physical sensing device into digital form. For instance, in a digital video camera, the sensors produce an electrical output proportional to light intensity.

The digitizer converts these outputs to digital data. Specialized image processing hardware usually consists of the digitizer just mentioned, plus hardware that performs other primitive operations, such as an arithmetic logic unit (ALU), which performs arithmetic and logical operations in parallel on entire images. One example of how an ALU is used is in averaging images as quickly as they are digitized, for the purpose of noise reduction. This type of hardware sometimes is called a front-end subsystem, and its most distinguishing characteristic is speed. In other words, this unit performs functions that require fast data throughputs (e.g., digitizing and averaging video images at 30 frames) that the typical main computer cannot handle. The computer in an image processing system is a general-purpose computer and can range from a PC to a supercomputer.

In dedicated applications, sometimes specially designed computers are used to achieve a required level of performance, but our interest here is on general-purpose image processing systems. In these systems, almost any well-equipped PC-type machine is suitable for offline image processing tasks. Software for image processing consists of specialized modules that perform specific tasks. A well-designed package also includes the capability for the user to write code that, as a minimum, utilizes the specialized modules.

In dedicated applications, sometimes specially designed computers are used to achieve a required level of performance, but our interest here is on general-purpose image processing systems. In these systems, almost any well-equipped PC-type machine is suitable for offline image processing tasks. Software for image processing consists of specialized modules that perform specific tasks. A well-designed package also includes the capability for the user to write code that, as a minimum, utilizes the specialized modules.

Digital storage for image processing applications falls into three principal categories:

(1) Short-term storage for use during processing,

(2) On-line storage for relatively fast recall, and

(3) Archival storage, characterized by infrequent access.

Storage is measured in bytes (eight bits), Kbytes (one thousand bytes), Mbytes (one million bytes), G- bytes (meaning Giga, or one billion, bytes), and T-bytes (i.e., tear, or one trillion, bytes). One method of providing short-term storage is computer memory. Another is by specialized boards, called frame buffers, that store one or more images and can be accessed rapidly, usually at video rates (e.g. at 30 complete images per second). The latter method allows virtually instantaneous image zoom, as well as scroll (vertical shifts) and pan (horizontal shifts). Frame buffers usually are housed in the specialized image processing hardware unit.

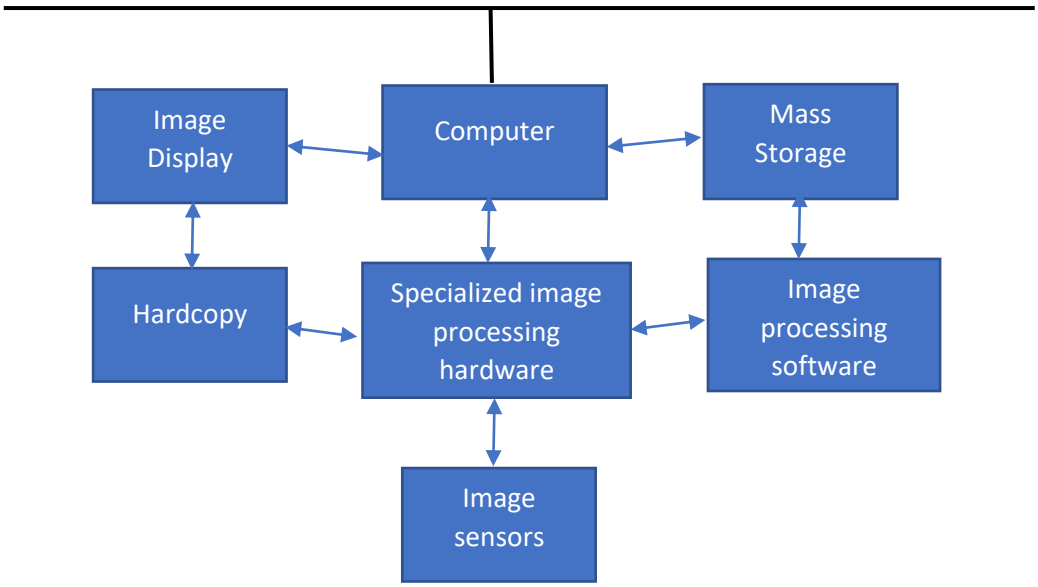


Fig. 1.3 Components of Digital Image Processing

Online storage generally takes the form of magnetic disks or optical-media storage. The key factor characterizing on-line storage is frequent access to the stored data. Finally, archival storage is characterized by massive storage requirements but infrequent need for access. Magnetic tapes and optical disks housed in “jukeboxes” are the usual media for archival applications. Image displays in use today are mainly colour (preferably flat screen) TV monitors. Monitors are driven by the outputs of image and graphics display cards that are an integral part of the computer system. Seldom there are requirements for image display

applications that cannot be met by display cards available commercially as part of the computer system.

1.5.5 IMAGE OPERATION ON PIXEL BASIS

Numerous references are made in the following chapters to operations between images, such as dividing one image by another. In Eq. (2.4-2), images were represented in the form of matrices. As we know, matrix division is not defined. However, when we refer to an operation like "dividing one image by another," we mean specifically that the division is carried out between corresponding pixels in the two images. Thus, for example, if f and g are images, the first element of the image formed by "dividing" f by g is simply the first pixel in f divided by the first pixel in g; of course, the assumption is that none of the pixels in g have value 0. Other arithmetic and logic operations are similarly defined between corresponding pixels in the images involved.

1.5.6 LINEAR AND NON-LINEAR OPERATIONS

The result of applying a linear operator to the sum of two images (that have been multiplied by the constants shown) is identical to applying the operator to the images individually, multiplying the results by the appropriate constants, and then adding those results. For example, an operator whose function is to compute the sum of K images is a linear operator. An operator that computes the absolute value of the difference of two images is not. An operator that fails the test of linear operation is by definition nonlinear. Linear operations are exceptionally important in image processing because they are based on a significant body of well-understood theoretical and practical results. Although nonlinear operations sometimes offer better performance, they are not always predictable, and for the most part are not well understood theoretically.

1.6 FUNDAMENTALS IN DIGITAL IMAGE PROCESSING

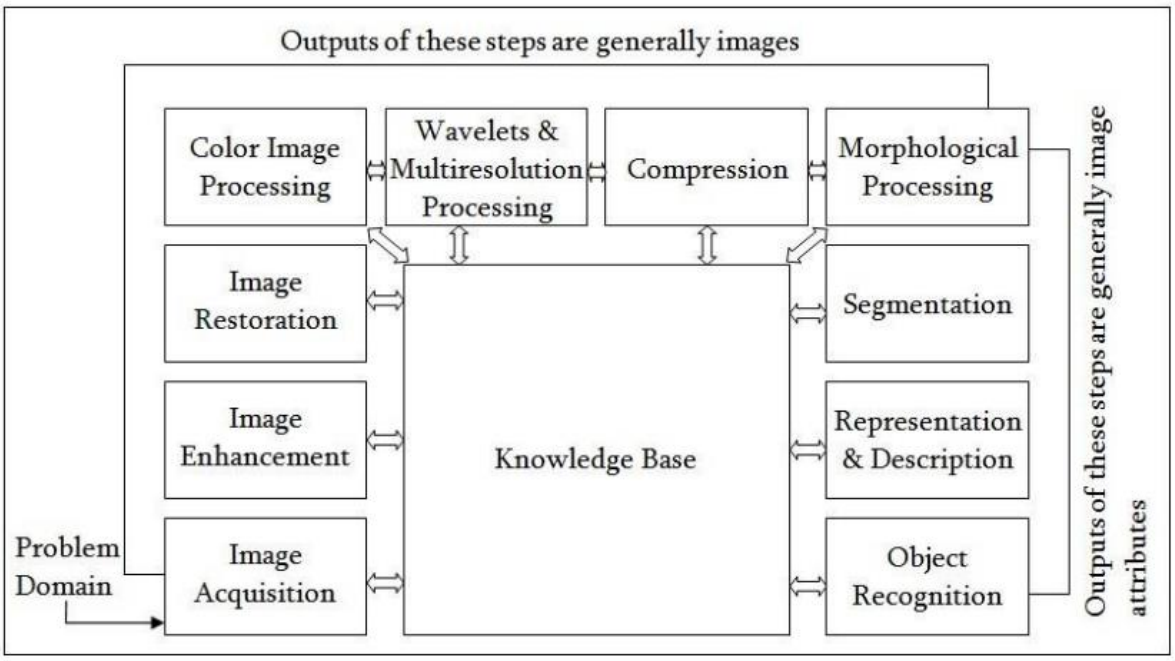


Fig. 1.4 Fundamentals of Digital Image Processing

1.6.1 IMAGE ACQUISITION

Image acquisition is to acquire a digital image. To do so requires an image sensor and the capability to digitize the signal produced by the sensor. The sensor could be monochrome or colour TV camera. The image sensor could be line scan camera that produces a single image line at a time.

Fig. 1.5 Image Acquisition

1.6.2 IMAGE ENHANCEMENT

Basically, the idea behind enhancement techniques is to bring out detail that is obscured, or simply to highlight certain features of interesting an image. A familiar example of enhancement is when we increase the contrast of an image because “it looks better” .

Fig. 1.6 Image Enhancement

1.6.3 IMAGE RESTORATION

Image restoration is the process of recovering an image from a degraded version—usually a blurred and noisy image. Image restoration is a fundamental problem in image processing, and it also provides a testbed for more general inverse problems.

Fig. 1.7 Image Restoration

1.6.4 COLOUR IMAGE PROCESSING

The use of colour in image processing is motivated by two principal factors. First, color is a powerful descriptor that often simplifies object identification and extraction from a scene. Second, humans can discern thousands of colour shades and intensities, compared to about only two dozen shades of gray.

Fig. 1.8 Colour Image Processing

1.6.5 WAVELETS AND MULTI RESOLUTION PROCESSING

Wavelets are the formation for representing images in various degrees of resolution. Unlike the Fourier transform, whose basis functions are sinusoids, wavelet transforms are based on small values, called Wavelets, of varying frequency and limited duration.

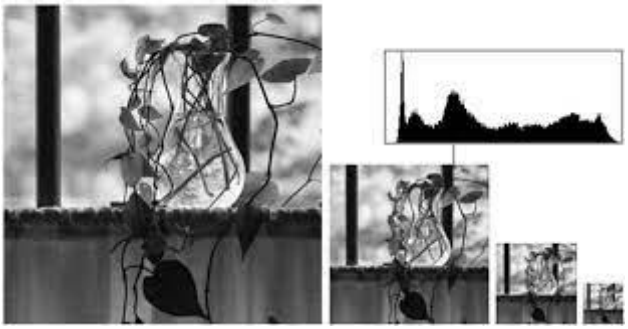


Fig. 1.9 Wavelets and Multi Resolution Processing

1.6.6 COMPRESSION

Compression, as the name implies, deals with techniques for reducing the storage required saving an image, or the bandwidth required for transmitting it. Image compression is familiar to most users of computers in the form of image file extensions, such as jpg file extension used in the JPEG (Joint Photographic Experts Group) image compression standard.

Fig. 1.10 Compression

1.6.7 MORPHOLOGICAL PROCESSING

Morphological processing deals with tools for extracting image components that are useful in the representation and description of shape. Sets in mathematical morphology represent objects in an image. For example, the set of all black pixels in a binary image is a complete morphological description of the image.

Fig. 1.11 Blur to Deblur Image

1.6.8 SEGMENTATION

Image segmentation involves converting an image into a collection of regions of pixels that are represented by a mask or a labelled image. By dividing an image into segments, you can process only the important segments of the image instead of processing the entire image.

Fig. 1.12 Segmentation

1.6.9 REPRESENTATION AND DESCRIPTION

Description, also called feature selection, deals with extracting attributes that result in some quantitative information of interest or are basic for differentiating one class of objects from another.

1.6.10 OBJECT RECOGNITION

The last stage involves recognition and interpretation. Recognition is the process that assigns a label to an object based on the information provided by its descriptors. Interpretation involves assigning meaning to an ensemble of recognized objects.

Fig. 1.13 Object Recognition

CHAPTER - 2

LITERATURE SURVEY

This implementation is based on Python Model, a multi linguistic model. This features a computer tool box functions to determine the results, besides USB webcam support package and image processing tool box also have a key role in implementation.

A Novel Framework for Dynamic Hand Gesture Recognition Using Convolutional Neural Networks and Hidden Markov Models. Dynamic Hand Gesture Based Sign Word Recognition Using Convolutional Neural Network with Feature Fusion. This paper proposes a hand gesture recognition system using convolutional neural networks (CNNs).

The system uses a combination of hand shape and motion features extracted from the input hand gesture frames to train a CNN. The proposed system achieves an accuracy of 95% on the hand gesture recognition task.

This study proposed a framework for dynamic hand gesture recognition that uses CNNs and hidden Markov models (HMMs). The CNN is used to extract features from the hand gesture frames, and the HMM is used to model the temporal dynamics of the gestures. The system achieved an accuracy of 96.5% on the dynamic hand gesture recognition task.

Hand gesture recognition for human computer interaction is an area of active research in computer vision and machine learning. One of its primary goals is to create systems, which can identify specific gestures and use them to convey information. Though, gestures need to be modelled in the spatial and temporal domains, where a hand posture is the static structure of the hand and a gesture is the dynamic movement of the hand.

This work main focus is on creating a vision-based system able to do real-time sign language recognition. The reason for choosing a system based on vision relates to the fact that it provides a simpler and more intuitive way of communication between a human and a computer.

[1] M. A. Rahim, J. Shin and M. R. Islam, "Dynamic Hand Gesture Based Sign Word Recognition Using Convolutional Neural Network with Feature Fusion," *2019 IEEE 2nd International Conference on Knowledge Innovation and Invention (ICKII)*, Seoul, Korea (South), 2019, pp. 221-224, doi: 10.1109/ICKII46306.2019.9042600.

In this paper, we developed a dynamic hand gesture-based sign word recognition system using a convolutional neural network that translates sign gestures into text. The different steps such as pre-processing, feature extraction and classification are implemented to detect hand gestures and sign word recognition. The hand gesture images are pre-processed according to the YCbCr conversion, grayscale image selection, binarization, erosion and fills the gaps. A deep learning technique i.e., CNN with feature fusion are used to extract the features and provide to the classifier for classification. A SoftMax classifier is used to classify these gestures. Also, the gesture-based sign-word recognition system is implemented by a webcam in real-time. As a result, the average acceptance of gesture-based sign word recognition is 96.96% which leads to better results than state-of-art systems. The future work will be aimed at developing the hand gesture-based sign words and novel sentences interpretation algorithm for sign language recognition system.

[2] S. Marcel, O. Bernier, J. . -E. Viallet and D. Collobert, "Hand gesture recognition using input-output hidden Markov models," *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*, Grenoble, France, 2000, pp. 456-461, doi: 10.1109/AFGR.2000.840674.

We have developed a method to recognize the unknown input gestures by using HMMs. Since the variation of the hand gestures is usually large, the transition between states is necessary in each gesture for an effective hand tracking. We apply this system to recognize the single gesture. In the experiments, we assume stationary background so that our system will have smaller search region for tracking. With a larger training set and context modelling, lower error rates are expected and generalization to user independent gesture recognition system should be developable. Once we add a new gesture into the system, we only need to re-train another HMM for the new gesture, since the relationships between new model and the original models are independent.

[3] J. Yashas and G. Shivakumar, "Hand Gesture Recognition: A Survey," *2019 International Conference on Applied Machine Learning (ICAML)*, Bhubaneswar, India, 2019, pp. 3-8, doi: 10.1109/ICAML48257.2019.00009.

In this paper an effective gesture recognition pipeline for the Leap Motion, for depth sensors and for their combined usage has been proposed. The different nature of data provided by the Leap Motion (i.e., a higher level but more limited data description) with respect to the depth cameras, poses challenging issues for which effective solutions have been presented. An ad-hoc calibration scheme allowed to jointly calibrate the Leap Motion with depth sensors. The limited number of points computed by the first device makes this task Multiped Tools Apply quite challenging but the proposed scheme allows to obtain a good accuracy sufficient for the joint exploitation of the data from the two devices. Several different feature sets have been presented for both sensors. Four different types of features have been extracted from the Leap Motion while different types of descriptors have been computed from the depth data based on different clues likes the distances from the hand centroid, the curvature of the hand contour and the convex hull of the hand shape. It has also been shown how to exploit Leap Motion data to improve the computation time and accuracy of the depth features. Experimental results have shown how the data provided by Leap Motion, even if not completely reliable, allows to obtain a reasonable overall accuracy with the proposed set of features and classification algorithms. A very good accuracy can be obtained from depth data that is a more complete description of the hand shape, in particular distance and curvature descriptors allow to obtain almost optimal performances. Performances remain very good even when the classification algorithm is changed or feature selection approaches are used to reduce the dimensionality of the feature vectors. Future work will address the recognition of dynamic gestures with the proposed setup and improved schemes for the detection and localization of the fingertips jointly exploiting the data from the two sensors.

[4] T. Yamashita and T. Watasue, "Hand posture recognition based on bottom-up structured deep convolutional neural network with curriculum learning," *2014 IEEE International Conference on Image Processing (ICIP)*, Paris, France, 2014, pp. 853-857, doi: 10.1109/ICIP.2014.7025171.

We propose a base up organized profound convolutional brain organization. The concept of curriculum learning, which breaks down difficult tasks into simpler ones with intermediate information, serves as the foundation for this network. In order to produce binary

images against a background of clutter, we make use of a binarization layer to obtain the intermediate information. As a result, the network does a better job of recognizing hand posture than the standard method without a binarization layer. In a future project, we will apply the proposed approach to other segmentation-based object recognition methods.

[5] G. Marin, F. Dominio and P. Zanuttigh, "Hand gesture recognition with leap motion and kinect devices," *2014 IEEE International Conference on Image Processing (ICIP)*, Paris, France, 2014, pp. 1565-1569, doi: 10.1109/ICIP.2014.7025313.

In this paper two different gesture recognition algorithms for the Leap Motion and Kinect devices have been proposed. Different feature sets have been used to deal with the different nature of data provided by the two devices, the Leap Motion provides a higher level but more limited data description while Kinect provides the full depth map. Even if the data provided by the Leap Motion is not completely reliable, since some fingers might not be detected, the proposed set of features and classification algorithm allows to obtain a good overall accuracy. The more complete description provided by the depth map of the Kinect allows to capture other properties missing in the Leap Motion output and by combining the two devices a very good accuracy can be obtained. Experimental results show also that the assignment of each finger to a specific angular region leads to a considerable increase of performance. Future work will address the joint calibration of the two devices in order to compute new features based on the combination of the 3D positions computed by the two devices, and the recognition of dynamic gestures with the two sensors.

[6] F. Zhan, "Hand Gesture Recognition with Convolution Neural Networks," *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*, Los Angeles, CA, USA, 2019, pp. 295-298, doi: 10.1109/IRI.2019.00054.

In this paper, we developed a CNN-based human hand gesture recognition system. The salient feature of the system is that there is no need to build a model for every gesture using hand features such as fingertips and contours. To have robust performance, we applied a GMM to learn the skin model and segment the hand area for recognition. Also, the calibration of the hand pose was used to rotate and shift the hand on the image to a neutral pose. Then, a CNN was trained to learn seven gesture types in this paper. In the experiments, we conducted 4-fold cross-validation on the system where 600 and 200 images from a subject were used to train and

test, respectively and the results showed that the average recognition rates of the seven gesture types were around 99%. To test the proposed method on multiple subjects, we trained and tested the hand images of the seven gesture types from seven subjects. The average recognition rate was 95.96%. The proposed system also had the satisfactory results on the transitive gestures in a continuous motion using the proposed rules. In the future, a high-level semantic analysis will be applied to the current system to enhance the recognition capability for complex human tasks

[7] L. Guo, Z. Lu and L. Yao, "Human-Machine Interaction Sensing Technology Based on Hand Gesture Recognition: A Review," in *IEEE Transactions on Human-Machine Systems*, vol. 51, no. 4, pp. 300-309, Aug. 2021, doi: 10.1109/THMS.2021.3086003

This paper presents an efficient and accurate hand gesture recognition system using skeleton information of Kinect sensor. The proposed method is to detect the hand area and fingertip, and recognize the movements of hands for performing the different gesture instructions to interact with human and machine. In addition, this system recognizes the numeric number using fingertip detection. The purpose of this research is to develop a method of performing hand gestures in an uncontrolled environment or in a situation where a person cannot contiguity with devices. For doing this, an experimental setup was established in a lab environment where 15 people participated. From the experimental results, the system shows high accuracy to recognize the hand gestures as well as the numbers using fingertips detection.

[8] T. Starner, J. Weaver and A. Pentland, "Real-time American sign language recognition using desk and wearable computer-based video," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 12, pp. 1371-1375, Dec. 1998, doi: 10.1109/34.735811.

We present two real-time hidden Markov model-based systems for recognizing sentence-level continuous American Sign Language (ASL) using a single camera to track the user’s unadorned hands. The first system observes the user from a desk mounted camera and achieves 92 percent word accuracy. The second system mounts the camera in a cap worn by the user and achieves 98 percent accuracy (97 percent with an unrestricted grammar). Both experiments use a 40-word lexicon

[9] S. Mitra and T. Acharya, "Gesture Recognition: A Survey," in *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 3, pp. 311-324, May 2007, doi: 10.1109/TSMCC.2007.893280.

The importance of gesture recognition lies in building efficient human–machine interaction. Its applications range from sign language recognition through medical rehabilitation to virtual reality. In this article, we have provided a survey on gesture recognition, with particular emphasis on hand gestures and facial expressions. The major tools surveyed for this purpose include HMMs, particle filtering and condensation algorithm, FSMs, and ANNs. A lot of research has been undertaken on sign language recognition, mainly using the hands (and lips). Facial expression modelling involves the use of eigenfaces, FACS, contour models, wavelets, optical flow, skin colour modelling, as well as a generic, unified feature-extraction-based approach. A hybridization of HMMs and FSMs is a potential study in order to increase the reliability and accuracy of gesture recognition systems. HMMs are computationally expensive and require large amount of training data. Performance of HMM-based systems could be limited by the characteristics of the training dataset. On the other hand, the statistically predictive state transition of the FSM might possibly lead to more reliable recognition. An interesting approach worth exploring is the independent modelling of each state of the FSM as an HMM. This can be useful in recognizing a complex gesture consisting a sequence of smaller gestures

[10] R. Cutler and M. Turk, "View-based interpretation of real-time optical flow for gesture recognition," *Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition*, Nara, Japan, 1998, pp. 416-421, doi: 10.1109/AFGR.1998.670984.

We have used real-time optical flow to segment a user’s dominant motions. Using a rule-based technique and general characteristics of the motion blobs, we can recognize among a set of gestures. The parameters for each gesture are estimated in a context specific manner. The system has been successfully applied to an interactive system for children. Future work for this system includes: adding more gestures (e.g., pointing) and allowing simultaneous gestures (e.g., clapping while marching); using fuzzy logic for the reasoning system to make the gesture recognition more robust; enhancing the motion segmentation to allow intersecting motion blobs; and adding the ability to handle multiple users in the field of view, with associated gestures for interaction (e.g., shaking hands).

CHAPTER - 3
METHODOLOGY

3.1 PROPOSED METHOD

Implementation of Sign Language Recognition(SLR) :-

Training the model by Python, Tensor Flow, Keras, Media Pipe tools. Testing the model by trained model data set. At the end performance is evaluated by practical application.

In this project, we design a real-time American Sign Language (ASL) to English text translation system for sign language. The system recognises hand movements and converts them into text using a deep learning method that blends convolutional and recurrent neural networks. In both hand gesture recognition and phrase reconstruction, the system produced highly accurate results. The method, which may be expanded to other sign languages, has the potential to enhance accessibility and communication for deaf people.

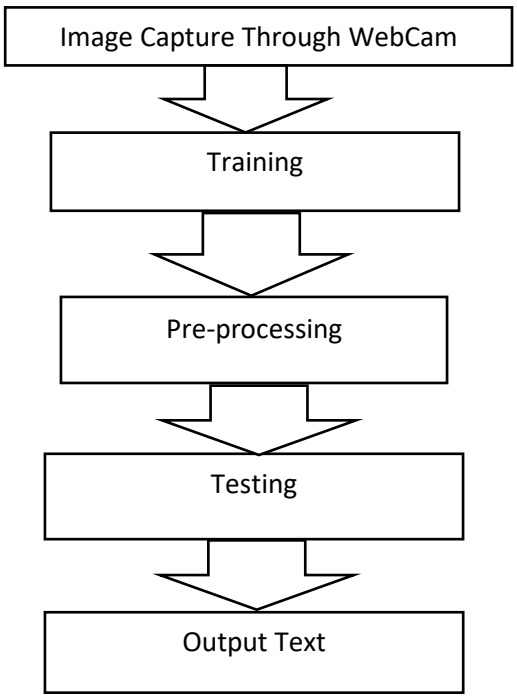


Fig. 3.1. Block Diagram of Proposed architecture

3.2 WORKING PRINCIPLE

The working of Sign Language Recognition (SLR) is simple but the technology is impressive. The input image is captured from the live video using a low-cost device, such as a webcam and pre-processed hand gesture image. The pre-processing is accomplished with the conversion of YCbCr, Binarization, Erosion and finally hole fillings. Two channels of CNN are used to extract the features from pre-processed images. The feature fusion is performed at the fully connected layer and this feature is used for gesture classification by the SoftMax classifier. An experimental setup established in our laboratory environment and the user can recognize the signs of fifteen common words in real-time. The experimental results show high recognition accuracy in gesture-based sign word recognition compared with the state-of-art systems.

In this we have five step process which is processed through software,

1. Collection of data
2. Functioning data
3. Conversion of data to .npy (Matrix)
4. Training Model
5. App data

3.3 INPUT DATA

The data which is given as input are given below:-

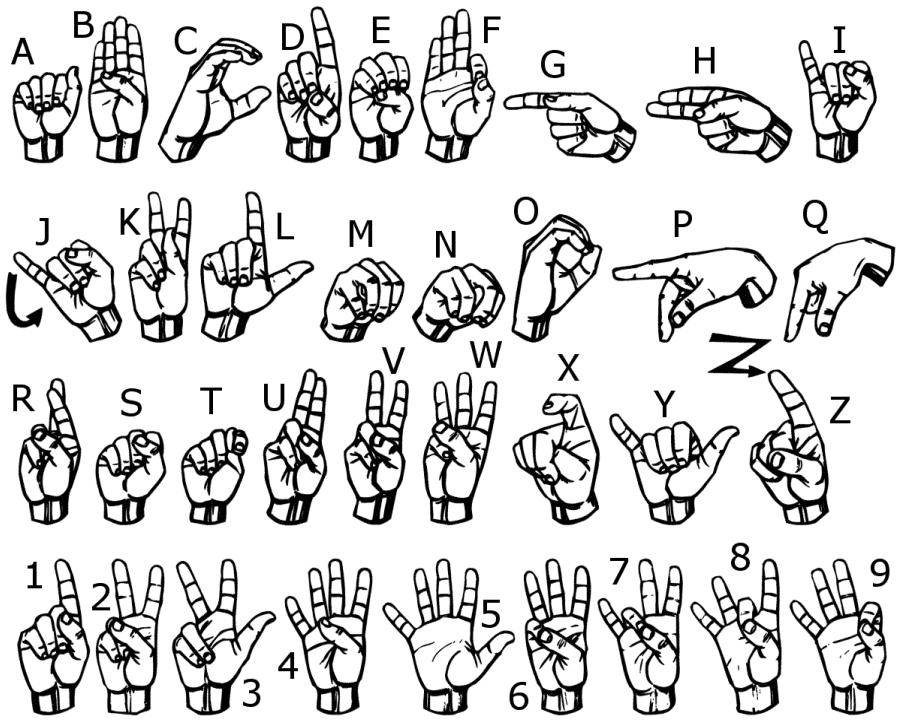


Fig. 3.2 Hand gestures for alphabets and numbers

3.4 SOFTWARE REQUIREMENTS

- Python
- Tensor Flow
- Keras
- OpenCV or CV2
- Media Pipe

3.5 FLOW CHART

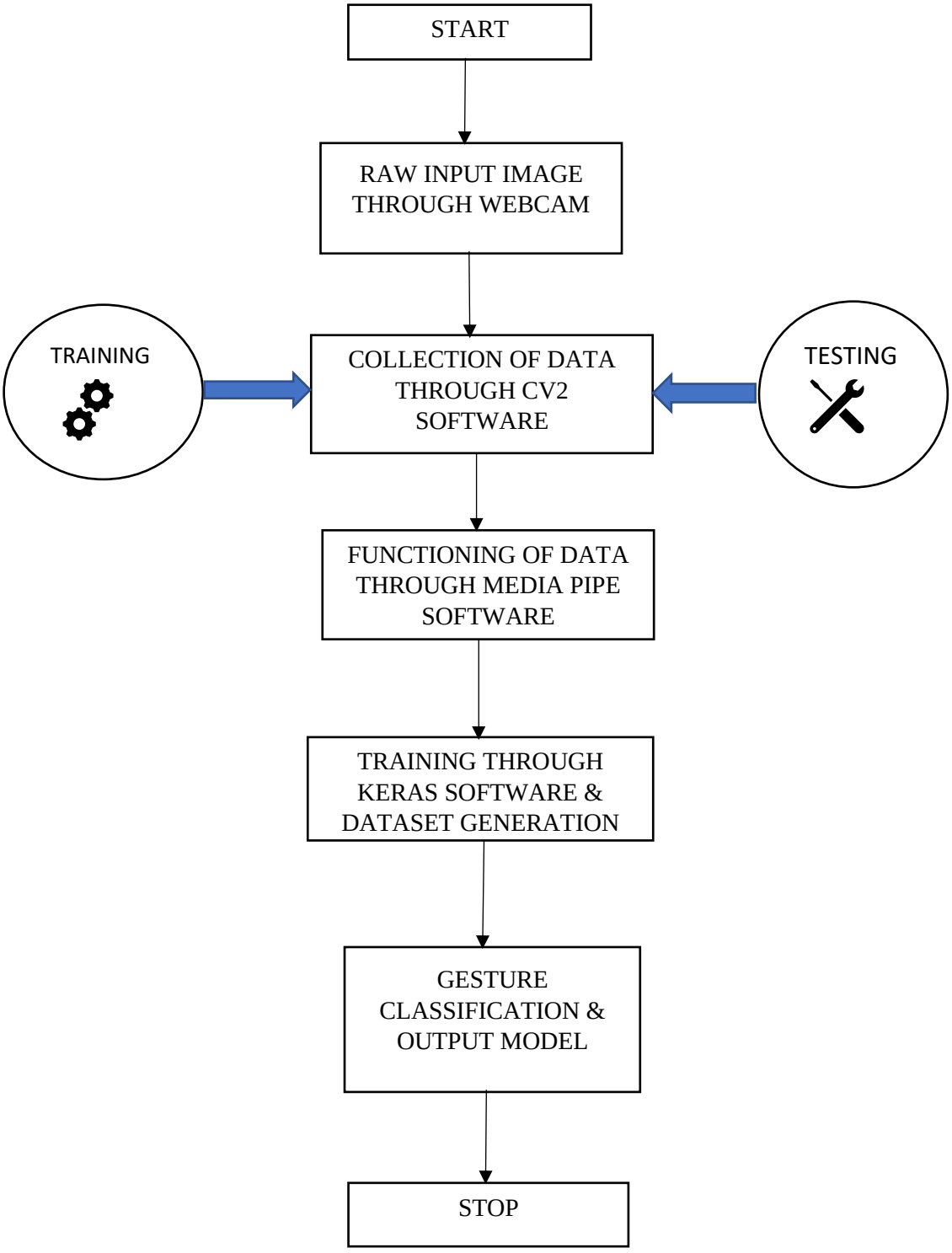


Fig. 3.3 Flow Chart of Working

3.6 IMPLEMENTATION OF SLR MODEL

3.6.1 TRAINING, TESTING & WORKING:

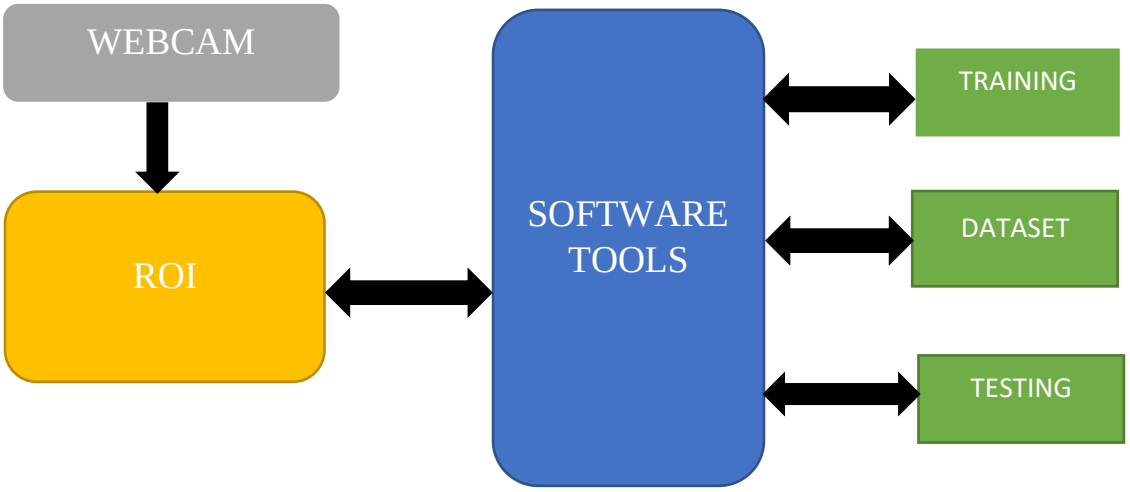


Fig. 3.4 Block Diagram of Training, Testing & Working

We convert our input images(RGB) into grayscale and apply gaussian blur to remove unnecessary noise. We apply adaptive threshold to extract our hand from the background and resize our images to 128 x 128. We feed the input images after pre-processing to our model for training and testing after applying all the operations mentioned above. The prediction layer estimates how likely the image will fall under one of the classes. So the output is normalized between 0 and 1 and such that the sum of each values in each class sums to 1. We have achieved this using soft max function.

At first the output of the prediction layer will be somewhat far from the actual value. To make it better we have trained the networks using labelled data. The cross-entropy is a performance measurement used in the classification. It is a continuous function which is positive at values which is not same as labelled value and is zero exactly when it is equal to the labelled value. Therefore, we optimized the cross-entropy by minimizing it as close to zero. To do this in our network layer we adjust the weights of our neural networks. TensorFlow has an inbuilt function to calculate the cross entropy.

3.7 PYTHON

3.7.1 INTRODUCTION

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation via the off-side rule. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Fig. 3.5 Python

3.7.2 DESIGN PHILOSOPHY AND FEATURES

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of their features support functional programming and aspect-oriented programming (including metaprogramming and metaobjects). Many other paradigms are supported via extensions, including design by contract and logic programming. Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. It uses dynamic name resolution (late binding), which binds method and variable names during program execution.

Its design offers some support for functional programming in the Lisp tradition. It has filter, map and reduce functions; list comprehensions, dictionaries, sets, and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

Its core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms 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 building all of its functionality into its core, Python was designed to be highly extensible via 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, which espoused the opposite approach.

3.7.3 SYNTAX AND SEMANTICS

Python is meant to be an easily readable language. Its formatting is visually uncluttered and often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are allowed but rarely used. It has fewer syntactic exceptions and special cases than C or Pascal.

3.7.4 INDENTATION

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block.[82] Thus, the program's visual structure accurately represents its semantic structure.[83] This feature is sometimes termed the off-side rule. Some other languages use indentation this way; but in most, indentation has no semantic meaning. The recommended indent size is four spaces.

3.7.5 LIBRARIES

Python's large standard library provides tools suited to many tasks and is commonly cited as one of its greatest strengths. For Internet-facing applications, many standard formats and protocols such as MIME and HTTP are supported. It includes modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary-precision decimals, manipulating regular expressions, and unit testing.

Some parts of the standard library are covered by specifications—for example, the Web Server Gateway Interface (WSGI) implementation follows PEP but most are specified by their code, internal documentation, and test suites. However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

As of 14 November 2022, the Python Package Index (PyPI), the official repository for third-party Python software, contains over 415,000[121] packages with a wide range of functionality, including:

- Automation
- Data analytics
- Databases
- Documentation
- Graphical user interfaces
- Image processing
- Machine learning
- Mobile apps
- Multimedia
- Computer networking
- Scientific computing
- System administration
- Test frameworks
- Text processing
- Web frameworks

3.7.6 DEVELOPMENET ENVIRONMENTS

Most Python implementations (including CPython) include a read–eval–print loop (REPL), permitting them to function as a command line interpreter for which users enter statements sequentially and receive results immediately. Python also comes with an Integrated development environment (IDE) called IDLE, which is more beginner-oriented. Other shells, including IDLE and IPython, add further abilities such as improved auto-completion, session state retention, and syntax highlighting.

As well as standard desktop integrated development environments, there are Web browser-based IDEs, including SageMath, for developing science- and math-related programs; PythonAnywhere, a browser-based IDE and hosting environment; and Canopy IDE, a commercial IDE emphasizing scientific computing.

3.8 TENSOR FLOW

3.8.1 INTRODUCTION

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.TensorFlow was developed by the Google Brain team for internal Google use in research and production. The initial version was released under the Apache License 2.0 in 2015.Google released the updated version of TensorFlow, named TensorFlow 2.0, in September 2019.TensorFlow can be used in a wide variety of programming languages, including Python, JavaScript, C++, and Java. This flexibility lends itself to a range of applications in many different sectors.

TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

Fig. 3.6 Tensor Flow

Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors. During the Google I/O Conference in June 2016.

3.8.2 TENSOR PROCESSING UNIT

In May 2016, Google announced its Tensor processing unit (TPU), an application-specific integrated circuit (ASIC, a hardware chip) built specifically for machine learning and tailored for TensorFlow. A TPU is a programmable AI accelerator designed to provide high throughput of low-precision arithmetic (e.g., 8-bit), and oriented toward using or running models rather than training them. Google announced they had been running TPUs inside their data centers for more than a year, and had found them to deliver an order of magnitude better-optimized performance per watt for machine learning.

In May 2017, Google announced the second-generation, as well as the availability of the TPUs in Google Compute Engine. The second-generation TPUs deliver up to 180 teraflops of performance, and when organized into clusters of 64 TPUs, provide up to 11.5 petaflops.

In May 2018, Google announced the third-generation TPUs delivering up to 420 teraflops of performance and 128 GB high bandwidth memory (HBM). Cloud TPU v3 Pods offer 100+ petaflops of performance and 32 TB HBM. In February 2018, Google announced that they were making TPUs available in beta on the Google Cloud Platform.

3.8.3 TENSORFLOW 2.0

As TensorFlow's market share among research papers was declining to the advantage of PyTorch, the TensorFlow Team announced a release of a new major version of the library in September 2019. TensorFlow 2.0 introduced many changes, the most significant being TensorFlow eager, which changed the automatic differentiation scheme from the static computational graph, to the "Define-by-Run" scheme originally made popular by Chainer and later PyTorch. Other major changes included removal of old libraries, cross-compatibility between trained models on different versions of TensorFlow, and significant improvements to the performance on GPU.

3.8.4 FEATURES

- **Auto-Differentiation**

  Auto-Differentiation is the process of automatically calculating the gradient vector of a model with respect to each of its parameters. With this feature, TensorFlow can automatically compute the gradients for the parameters in a model, which is useful to algorithms such as backpropagation which require gradients to optimize performance. To do so, the framework must keep track of the order of operations done to the input Tensors in a model, and then compute the gradients with respect to the appropriate parameters.

- **Eager Execution**

  TensorFlow includes an "eager execution" mode, which means that operations are evaluated immediately as opposed to being added to a computational graph which is executed later. Code executed eagerly can be examined step-by step-through a debugger, since data is augmented at each line of code rather than later in a computational graph. This execution paradigm is considered to be easier to debug because of its step by step transparency.

- **Distribute**

  In both eager and graph executions, TensorFlow provides an API for distributing computation across multiple devices with various distribution strategies.[34] This distributed computing can often speed up the execution of training and evaluating of TensorFlow models and is a common practice in the field of AI.

- **Losses**

  To train and assess models, TensorFlow provides a set of loss functions (also known as cost functions). Some popular examples include mean squared error (MSE) and binary cross entropy (BCE). These loss functions compute the "error" or "difference" between a model's output and the expected output (more broadly, the difference between two tensors). For different datasets and models, different losses are used to prioritize certain aspects of performance.

- **Metrics**

  In order to assess the performance of machine learning models, TensorFlow gives API access to commonly used metrics. Examples include various accuracy metrics (binary, categorical, sparse categorical) along with other metrics such as Precision, Recall, and Intersection-over-Union (IoU).

- **TF.nn**

  TensorFlow.nn is a module for executing primitive neural network operations on models. Some of these operations include variations of convolutions (1/2/3D, Atrous, depth-wise), activation functions (Soft max, RELU, GELU, Sigmoid, etc.) and their variations, and other Tensor operations (max-pooling, bias-add, etc.).

- **Optimizers**

  TensorFlow offers a set of optimizers for training neural networks, including ADAM, ADAGRAD, and Stochastic Gradient Descent (SGD).[39] When training a model, different optimizers offer different modes of parameter tuning, often affecting a model's convergence and performance.

3.8.5 APPLICATIONS

- **Medical**

  GE Healthcare used TensorFlow to increase the speed and accuracy of MRIs in identifying specific body parts. Google used TensorFlow to create DermAssist, a free mobile application that allows users to take pictures of their skin and identify potential health complications. Sinovation Ventures used TensorFlow to identify and classify eye diseases from optical coherence tomography (OCT) scans.

- **Social Media**

  Twitter implemented TensorFlow to rank tweets by importance for a given user, and changed their platform to show tweets in order of this ranking. Previously, tweets were simply shown in reverse chronological order. The photo sharing app VSCO used TensorFlow to help suggest custom filters for photos.

- **Search Engine**

  Google officially released RankBrain on October 26, 2015, backed by TensorFlow.

- **Education**

  InSpace, a virtual learning platform, used TensorFlow to filter out toxic chat messages in classrooms. Liulishuo, an online English learning platform, utilized TensorFlow to create an adaptive curriculum for each student. TensorFlow was used to accurately assess a student's current abilities, and also helped decide the best future content to show based on those capabilities.

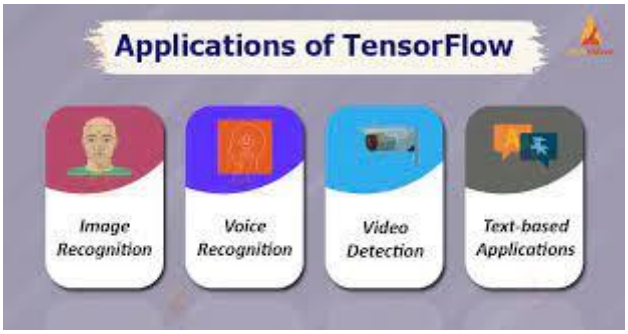


Fig. 3.7 Applications of Tensor Flow

3.9 KERAS

3.9.1 INTRODUCTION

Keras is a high-level neural networks library written in python that works as a wrapper to TensorFlow. It is used in cases where we want to quickly build and test the neural network with minimal lines of code. It contains implementations of commonly used neural network elements like layers, objective, activation functions, optimizers, and tools to make working with images and text data easier.

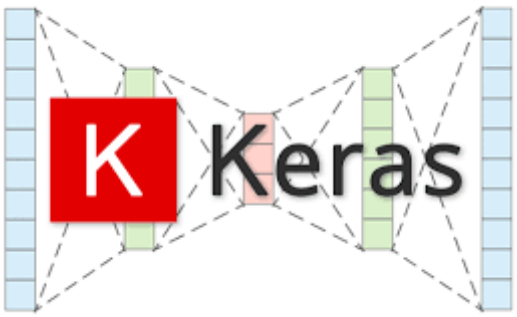


Fig. 3.8 Keras

3.9.2 FEATURES

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel.In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. It also allows use of distributed training of deep-learning models on clusters of Graphics processing units (GPU) and tensor processing units (TPU).

3.10 OPEN CV

3.10.1 INTRODUCTION

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly for real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage, then Itseez (which was later acquired by Intel). OpenCV(Open Source Computer Vision) is an open source library of programming functions used for real-time computer-vision. It is mainly used for image processing, video capture and analysis for features like face and object recognition. It is written in C++ which is its primary interface, however bindings are available for Python, Java, MATLAB/OCTAVE. OpenCV features GPU acceleration for real-time operations.

Fig. 3.9 Open CV

3.10.2 APPLICATIONS

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object detection
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion video tracking
- Augmented reality

3.11 MEDIAPIPE

3.11.1 INTRODUCTION

MediaPipe Solutions allows you to apply machine learning (ML) solutions to your apps, with a framework that lets you configure pre-built processing pipelines to get immediate, engaging, and useful output for your users. You can even customize these solutions using Model Maker to update the default models. MediaPipe is a Framework for building machine learning pipelines for processing time-series data like video, audio, etc. This cross-platform Framework works on Desktop/Server, Android, iOS, and embedded devices like Raspberry Pi and Jetson Nano.

Fig. 3.10 Media Pipe

The MediaPipe Hand Landmarker task lets you detect the landmarks of the hands in an image. You can use this Task to localize key points of the hands and render visual effects over the hands. This task operates on image data with a machine learning (ML) model as static data or a continuous stream and outputs hand landmarks in image coordinates, hand landmarks in world coordinates and handedness(left/right hand) of multiple detected hands.

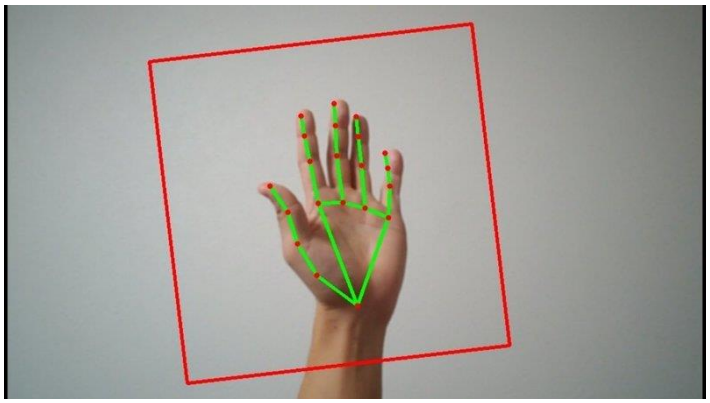


Fig. 3.11 Media pipe Hand Tracking

3.11.2 ADVANTAGES

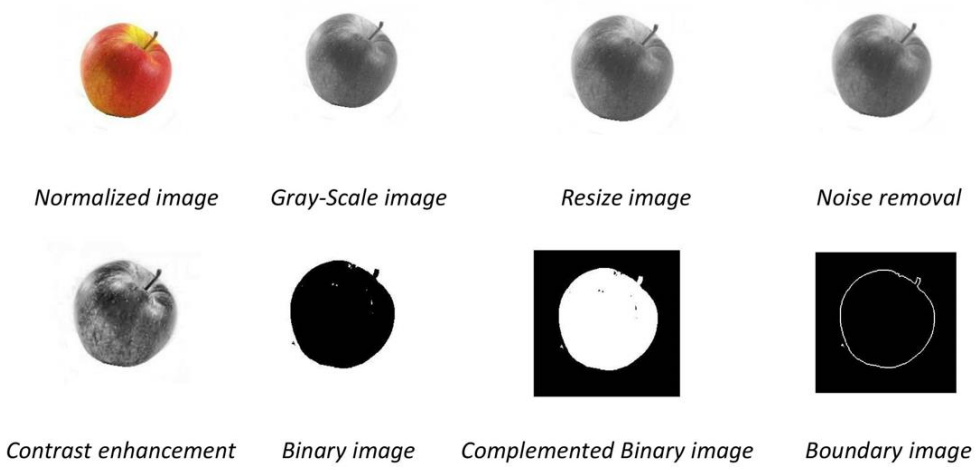
End-to-end acceleration: Use common hardware to build-in fast ML inference and video processing, including GPU, CPU, or TPU.

Build once, deploy anywhere: The unified framework is suitable for Android, iOS, desktop, edge, cloud, web, and IoT platforms.

Ready-to-use solutions: Prebuilt ML solutions demonstrate the full power of the Media Pipe framework.

3.12 STEP BY STEP PROCESS

1. Acquisition of image through web cam.
2. Detecting the part where only image of hand is required.
3. In next we use pre processing technique where RGB image is converted into gray scale image.
4. Segmentation, this part is used to determine the location of temporal boundaries of hand image that has been captured by a camera is segmented to obtain the hand area.
5. Feature Extraction, the hand gestures themselves are rich in variation of shapes, motions, and textures & also binary information of image.
6. Various classification algorithms are used: SVM, k-NN and CNN, so here we used CNN technique.
7. CNN - Convolutional Neural Networks(CNN) are the go-to choice for an image classification problem, is an artificial neural network consisting of convolutional layers, works well with the image data.
8. Trained imaged will be stored in database and classified.
9. The tested image compared with trained dataset models, so image with higher accuracy will be output in form of text.



3.12 Processing of Image

3.13 BLOCK DIAGRAM OF WORKING

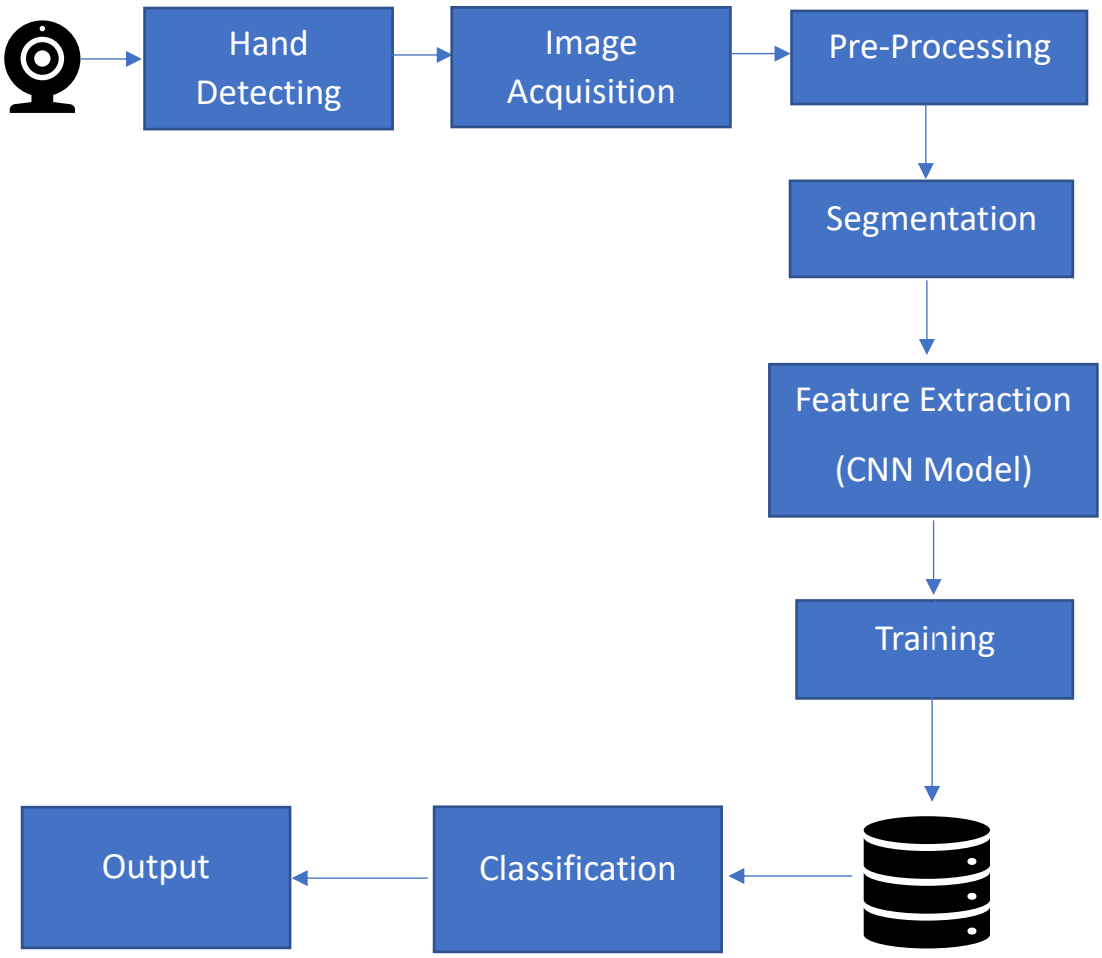


Fig. 3.13 Block Diagram of Working

3.14 INPUT THROUGH CAMERA FOR HAND DETECTION

In vision based methods computer camera is the input device for observing the information of hands or fingers. The Vision Based methods require only a camera, thus realizing a natural interaction between humans and computers without the use of any extra devices. These systems tend to complement biological vision by describing artificial vision systems that are implemented in software and/or hardware. The main challenge of vision-based hand detection is to cope with the large variability of human hand's appearance due to a huge number of hand movements, to different skin-colour possibilities as well as to the variations in view points, scales, and speed of the camera capturing the scene.

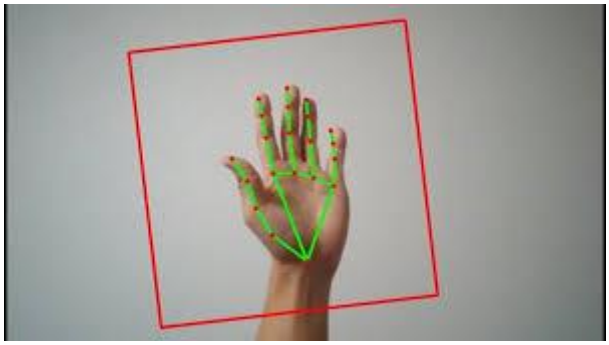


Fig. 3.14 Hand Detection

3.15 IMAGE ACQUISITION

Image acquisition can be defined as the act of procuring an image from sources. This can be done via hardware systems such as cameras, encoders, sensors, etc.The general aim of Image Acquisition is to transform an optical image (Real World Data) into an array of numerical data which could be later manipulated on a computer, before any video or image processing can commence an image must be captured by camera and converted into manageable entity . The Image Acquisition process consists of three steps:-

1. Optical system which focuses the energy

2. Energy reflected from the object of interest

3. A sensor which measure the amount of energy.

Image Acquisition is achieved by suitable camera. We use different cameras for different application. If we need an x-ray image, we use a camera (film) that is sensitive to x-ray. If we wan infrared image, we use camera which are sensitive to infrared radiation. For normal images (family pictures etc.) we use cameras which are sensitive to visual spectrum. Image Acquisition is the first step in any image processing system.

3.16 PRE-PROCESSING

The aim of pre-processing is to improve the quality of the image so that we can analyse it in a better way. By preprocessing we can suppress undesired distortions and enhance some features which are necessary for the particular application we are working for. Those features might vary for different applications.

For example, if we are working on a project which can automate Vehicle Identification, then our main focus lies on the vehicle, its colour, the registration plate, etc., We do not focus on the road or the sky or something which isn't necessary for this particular application. In pre processing RGB image converted into grayscale image.

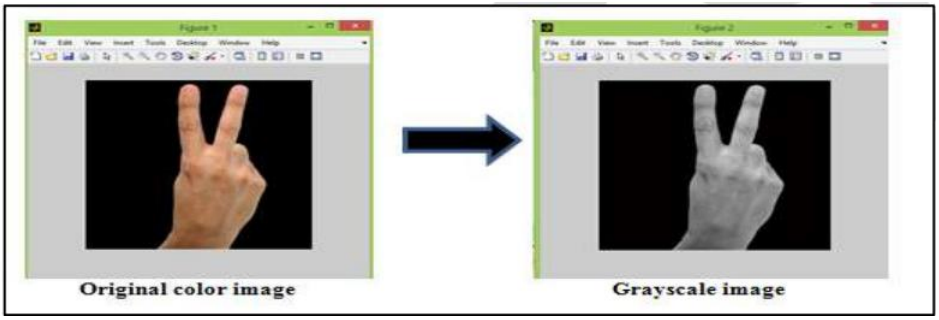


Fig. 3.15 Original to Gray Scale Conversion

Color or RGB image, every pixel is represented using R, G, and B values. This means that you need three numbers to represent an image, a pixel that is red in color will be represented as 255, 0, 0. The R value is 255, G and B are 0's this is a 3-channel image or a multi-channel image. you'll find that neural networks work better when the numeric values are small, so these pixel values in the range 0–255 are often scaled to be in the 0–1 range.

Grayscale images, Single channel ;one value to represent a pixel, and that is the intensity of a pixel. Each pixel represents only intensity information, so you have one value in the range 0-1A value of 1 means a pixel with the highest intensity, 0 represents a pixel with no intensity at all.

3.17 SEGMENTATION

Image segmentation is a commonly used technique in digital image processing and analysis to partition an image into multiple parts or regions, often based on the characteristics of the pixels in the image. Image segmentation could involve separating foreground from background, or clustering regions of pixels based on similarities in color or shape. For example, a common application of image segmentation in medical imaging is to detect and label pixels in an image or voxels of a 3D volume that represent a tumor in a patient's brain or other organs. Image segmentation is a method of dividing a digital image into subgroups called image segments by

boundaries reducing the complexity of the image and enabling further processing or analysis of each image segment.

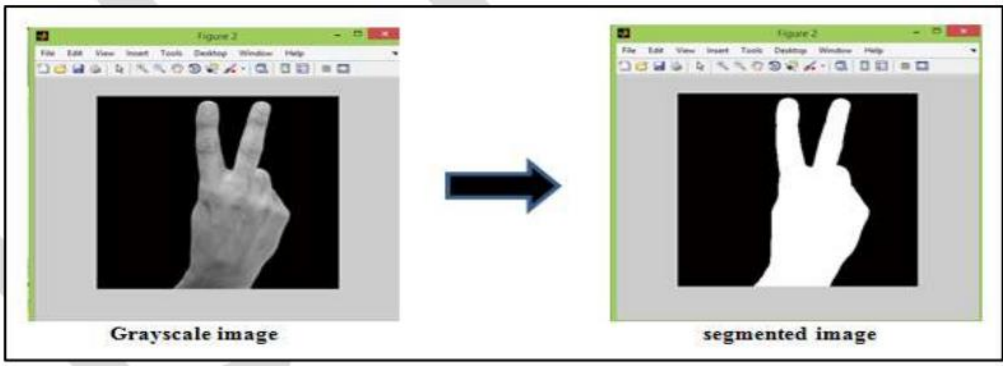


Fig. 3.16 Gray Scale to Segmentation

3.18 SEGMENTATION TECHNIQUES

3.18.1 TRESHOLDING

Using Otsu's method, im-binarize performs thresholding on a 2D or 3D grayscale image to create a binary image. To produce a binary image from an RGB color image, use rgb2gray to first convert it to a grayscale image.

Fig. 3.17 Thresholding

3.18.2 CLUSTERING

This technique lets you create a segmented labeled image using a specific clustering algorithm. Using K-means clustering–based segmentation, im-segkmeans segments an image into K number of clusters.

Fig. 3.18 Clustering

3.18.3 GRAPH-BASED SEGMENTATION

Graph-based segmentation techniques like lazy-snapping enable you to segment an image into foreground and background regions. MATLAB lets you perform this segmentation on your image either programmatically (lazysnapping) or interactively using the Image Segmenter app.

Fig. 3.19 Graph-Based Segmentation

Region growing is a simple region-based (also classified as a pixel-based) image segmentation method. A popularly used algorithm is active contour, which examines neighboring pixels of initial seed points and determines iteratively whether the pixel neighbors should be added to the region. You can also perform this segmentation on images using the Image Segmenter app.

3.19 FEATURE EXTRACTION

Feature extraction refers to the process of transforming raw data into numerical features that can be processed while preserving the information in the original data set. Feature extraction has functions like Detect, Extract, Matching & Storing features. Feature extraction refers to the process of transforming raw data into numerical features that can be processed while preserving the information in the original data set. Feature extraction has functions like Detect, Extract, Matching & Storing features.

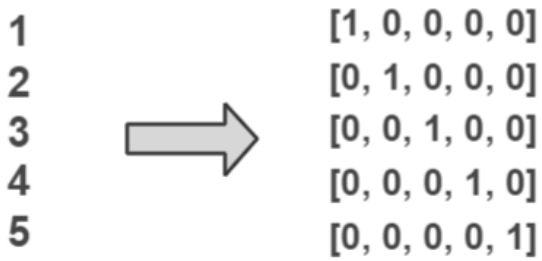


Fig. 3.20 Feature Extraction

3.20 CNN – CONVOLUTION NEURAL NETWORK

The convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer. With each layer, the CNN increases in its complexity, identifying greater portions of the image. Earlier layers focus on simple features, such as colors and edges. As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object. Convolutional Neural Networks (CNN), are deep neural networks used to process data that have a grid-like topology, e.g images that can be represented as a 2-D array of pixels.

A CNN model consists of four main operations: Convolution, Non-Linearity (Relu), Pooling and Classification (Fully-connected layer). CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons are usually fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The “full connectivity” of these networks make them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.).

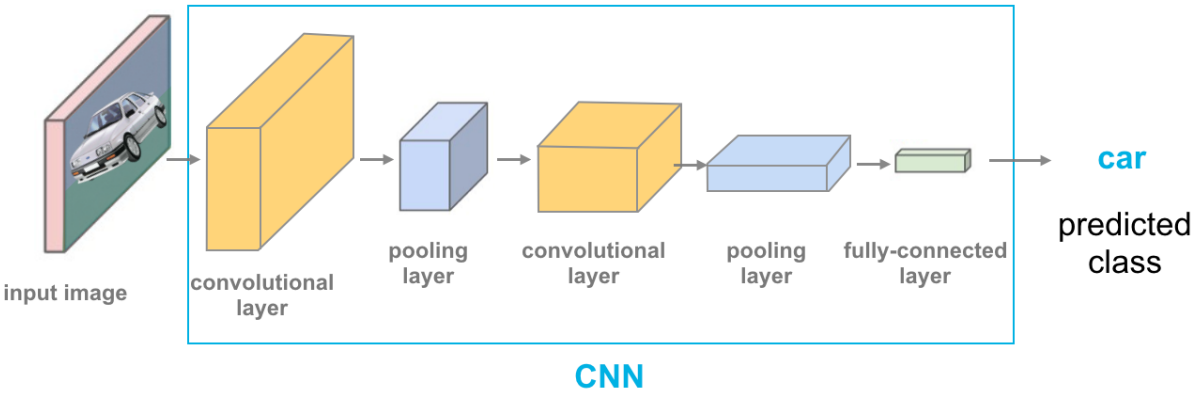


Fig. 3.21 Convolution

3.20.1 CONVOLUTION

A convolutional layer is the main building block of a CNN. It contains a set of filters (or kernels), parameters of which are to be learned throughout the training. The size of the filters is usually smaller than the actual image. The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image.

A convolutional layer is the main building block of a CNN. It contains a set of filters (or kernels), parameters of which are to be learned throughout the training.

The size of the filters is usually smaller than the actual image. The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image.

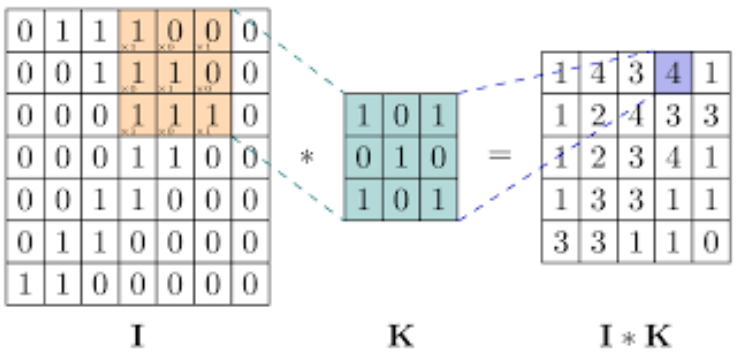


Fig. 3.22 CNN – Convolution Neural Network

3.20.2 RELU

A rectified linear unit (ReLU) is an activation function that introduces the property of non-linearity to a deep learning model and solves the vanishing gradients issue. “It interprets the positive part of its argument. It is one of the most popular activation functions in deep learning. A rectified linear unit (ReLU) is an activation function that introduces the property of non-linearity to a deep learning model and solves the vanishing gradients issue. “It interprets the positive part of its argument. It is one of the most popular activation functions in deep learning.

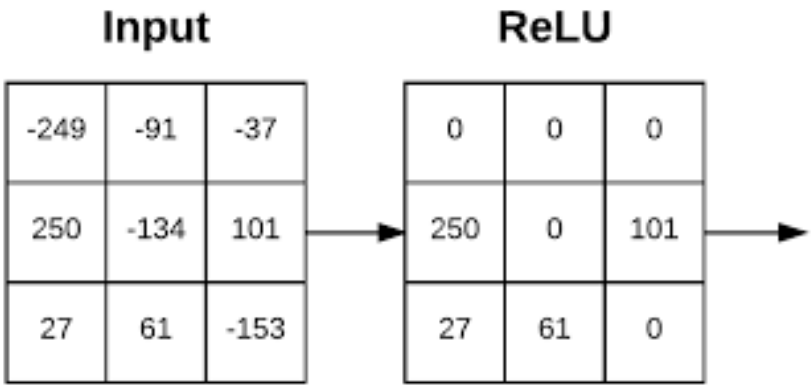


Fig. 3.23 ReLU – Rectified Linear Unit

3.20.3 POOLING

Pooling (also called down-sampling) reduces the dimensionality of each feature. It reduces the amount of information in each feature obtained in the convolutional layer while maintaining the most important information. We use pooling layer to decrease the size of activation matrix and ultimately reduce the learnable parameters. There are two type of pooling :

a) Max Pooling : In max pooling we take a window size [for example window of size 2*2], and only take the maximum of 4 values. Well lid this window and continue this process, so well finally get a activation matrix half of its original Size.

b) Average Pooling : In average pooling we take average of all values in a window.

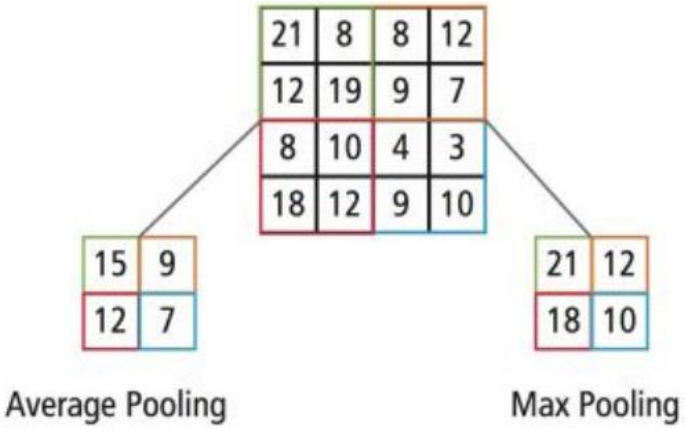


Fig. 3.24 Types of Pooling

3.20.4 FULLY CONNECTED LAYER

It is a multi layer perceptron that uses soft max function in the output layer. It takes the output of the previous layers, "flattens" them and turns them into a single vector that can be an input for the next stage. A fully connected neural network consists of a series of fully connected layers that connect every neuron in one layer to every neuron in the other layer. The major advantage of fully connected networks is that they are "structure agnostic" i.e. there are no special assumptions needed to be made about the input.

While being structure agnostic makes fully connected networks very broadly applicable, such networks do tend to have weaker performance than special-purpose networks tuned to the structure of a problem space.

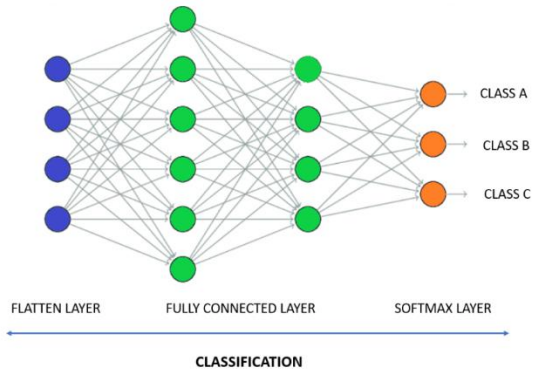


Fig. 3.25 Fully Connected Layer

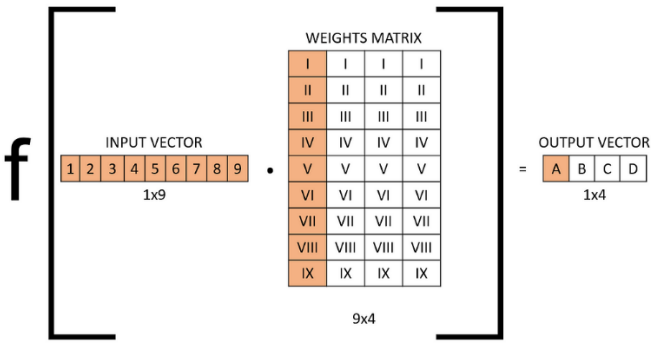


Fig. 3.26 Function of Fully Connected Layer

3.20.5 TRAINING AND TESTING

Training is the process of training data & storing them in Data base and classification is the process of categorizing and labeling groups of pixels. We convert our input images(RGB) into grayscale and apply gaussian blur to remove unnecessary noise.We apply adaptive threshold to extract our hand from the background and resize our images to 128 x 128. We feed the input images after pre-processing to our model for training and testing after applying all the operations mentioned above. The prediction layer estimates how likely the image will fall under one of the classes. So the output is normalized between 0 and 1 and such that the sum of each values in each class sums to 1.

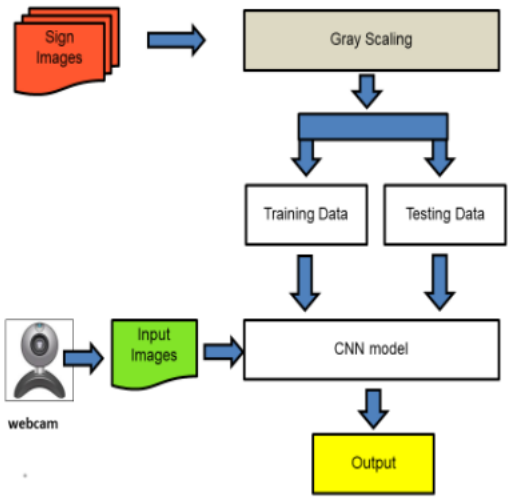


Fig. 3.27 Training & Testing

We have achieved this using softmax function. At first the output of the prediction layer will be somewhat far from the actual value. To make it better we have trained the networks using labeled data. The cross-entropy is a performance measurement used in the classification.

It is a continuous function which is positive at values which is not same as labeled value and is zero exactly when it is equal to the labeled value. Therefore we optimized the cross-entropy by minimizing it as close to zero. To do this in our network layer we adjust the weights of our neural networks. TensorFlow has an inbuilt function to calculate the cross entropy. As we have found out the cross entropy function, we have optimized it using Gradient Descent in fact with the best gradient descent optimizer is called Adam Optimizer.

3.21 IDE

Visual Studio Code, also commonly referred to as VS Code, is a source-code editor made by Microsoft with the Electron Framework, for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add functionality.

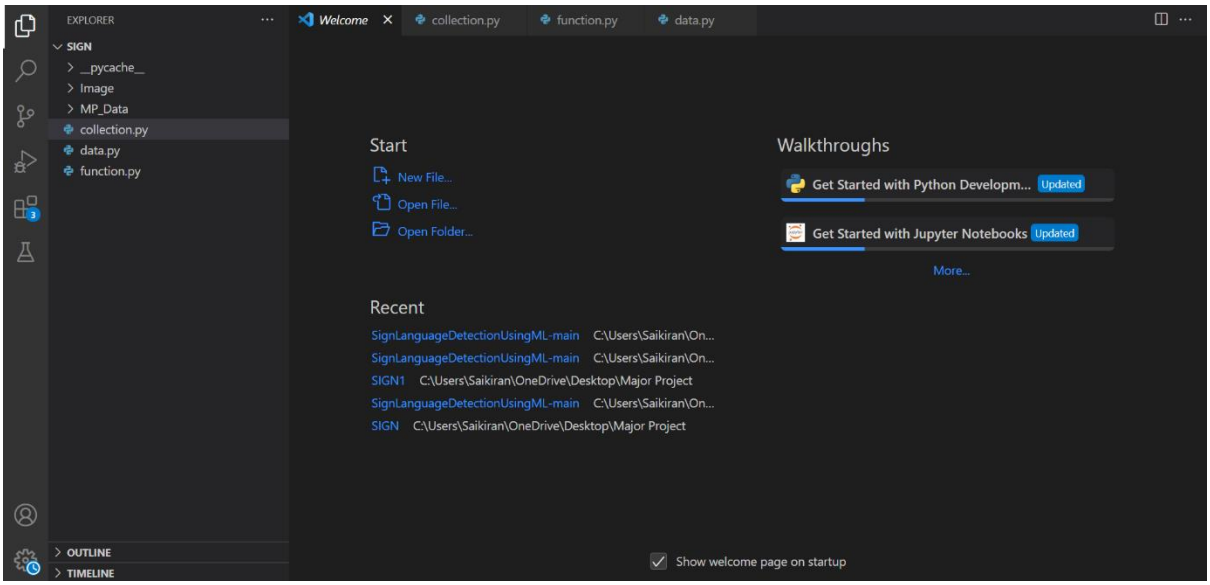


Fig. 3.28 VS Code Ide

Visual Studio Code is a source-code editor that can be used with a variety of programming languages, including C, C#, C++, Fortran, Go, Java, JavaScript, Node.js, Python, Rust. It is based on the Electron framework, which is used to develop Node.js web applications that run on the Blink layout engine. Visual Studio Code employs the same editor component (codenamed "Monaco") used in Azure DevOps (formerly called Visual Studio Online and Visual Studio Team Services).

3.22 CODE

3.22.1 COLLECTION OF DATA

```
import os
import cv2
cap=cv2.VideoCapture(0)
directory='Image/'
while True:
_,frame=cap.read()
count = {
        'a': len(os.listdir(directory+"/A")),
        'b': len(os.listdir(directory+"/B")),
        'c': len(os.listdir(directory+"/C")),
        'd': len(os.listdir(directory+"/D")),
        'e': len(os.listdir(directory+"/E")),
        'f': len(os.listdir(directory+"/F")),
        'g': len(os.listdir(directory+"/G")),
        'h': len(os.listdir(directory+"/H")),
        'i': len(os.listdir(directory+"/I")),
        'j': len(os.listdir(directory+"/J")),
        'k': len(os.listdir(directory+"/K")),
        'l': len(os.listdir(directory+"/L")),
        'm': len(os.listdir(directory+"/M")),
        'n': len(os.listdir(directory+"/N")),
        'o': len(os.listdir(directory+"/O")),
        'p': len(os.listdir(directory+"/P")),
        'q': len(os.listdir(directory+"/Q")),
        'r': len(os.listdir(directory+"/R")),
        's': len(os.listdir(directory+"/S")),
        't': len(os.listdir(directory+"/T")),
        'u': len(os.listdir(directory+"/U")),
        'v': len(os.listdir(directory+"/V")),
        'w': len(os.listdir(directory+"/W")),
        'x': len(os.listdir(directory+"/X")),
        'y': len(os.listdir(directory+"/Y")),
        'z': len(os.listdir(directory+"/Z"))
        }
row = frame.shape[1]
```

```
col = frame.shape[0]
cv2.rectangle(frame,(0,40),(300,400),(255,255,255),2)
cv2.imshow("data",frame)
cv2.imshow("ROI",frame[40:400,0:300])
frame=frame[40:400,0:300]
interrupt = cv2.waitKey(10)
    if interrupt & 0xFF == ord('a'):
        cv2.imwrite(directory+'A/'+str(count['a'])+'.png',frame)
    if interrupt & 0xFF == ord('b'):
        cv2.imwrite(directory+'B/'+str(count['b'])+'.png',frame)
    if interrupt & 0xFF == ord('c'):
        cv2.imwrite(directory+'C/'+str(count['c'])+'.png',frame)
    if interrupt & 0xFF == ord('d'):
        cv2.imwrite(directory+'D/'+str(count['d'])+'.png',frame)
    if interrupt & 0xFF == ord('e'):
        cv2.imwrite(directory+'E/'+str(count['e'])+'.png',frame)
    if interrupt & 0xFF == ord('f'):
        cv2.imwrite(directory+'F/'+str(count['f'])+'.png',frame)
    if interrupt & 0xFF == ord('g'):
        cv2.imwrite(directory+'G/'+str(count['g'])+'.png',frame)
    if interrupt & 0xFF == ord('h'):
        cv2.imwrite(directory+'H/'+str(count['h'])+'.png',frame)
    if interrupt & 0xFF == ord('i'):
        cv2.imwrite(directory+'I/'+str(count['i'])+'.png',frame)
    if interrupt & 0xFF == ord('j'):
        cv2.imwrite(directory+'J/'+str(count['j'])+'.png',frame)
    if interrupt & 0xFF == ord('k'):
        cv2.imwrite(directory+'K/'+str(count['k'])+'.png',frame)
    if interrupt & 0xFF == ord('l'):
        cv2.imwrite(directory+'L/'+str(count['l'])+'.png',frame)
    if interrupt & 0xFF == ord('m'):
        cv2.imwrite(directory+'M/'+str(count['m'])+'.png',frame)
    if interrupt & 0xFF == ord('n'):
        cv2.imwrite(directory+'N/'+str(count['n'])+'.png',frame)
    if interrupt & 0xFF == ord('o'):
        cv2.imwrite(directory+'O/'+str(count['o'])+'.png',frame)
    if interrupt & 0xFF == ord('p'):
        cv2.imwrite(directory+'P/'+str(count['p'])+'.png',frame)
    if interrupt & 0xFF == ord('q'):
        cv2.imwrite(directory+'Q/'+str(count['q'])+'.png',frame)
```

```
    if interrupt & 0xFF == ord('r'):
        cv2.imwrite(directory+'R/'+str(count['r'])+'.png',frame)
    if interrupt & 0xFF == ord('s'):
        cv2.imwrite(directory+'S/'+str(count['s'])+'.png',frame)
    if interrupt & 0xFF == ord('t'):
        cv2.imwrite(directory+'T/'+str(count['t'])+'.png',frame)
    if interrupt & 0xFF == ord('u'):
        cv2.imwrite(directory+'U/'+str(count['u'])+'.png',frame)
    if interrupt & 0xFF == ord('v'):
        cv2.imwrite(directory+'V/'+str(count['v'])+'.png',frame)
    if interrupt & 0xFF == ord('w'):
        cv2.imwrite(directory+'W/'+str(count['w'])+'.png',frame)
    if interrupt & 0xFF == ord('x'):
        cv2.imwrite(directory+'X/'+str(count['x'])+'.png',frame)
    if interrupt & 0xFF == ord('y'):
        cv2.imwrite(directory+'Y/'+str(count['y'])+'.png',frame)
    if interrupt & 0xFF == ord('z'):
        cv2.imwrite(directory+'Z/'+str(count['z'])+'.png',frame)
cap.release()

cv2.destroyAllWindows()
```

3.22.2 FUNCTION

```
#import dependency

import cv2

import numpy as np

import os

import mediapipe as mp

mp_drawing = mp.solutions.drawing_utils

mp_drawing_styles = mp.solutions.drawing_styles

mp_hands = mp.solutions.hands

def mediapipe_detection(image, model):

image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # COLOR CONVERSION BGR 2 RGB

image.flags.writeable = False              # Image is no longer writeable

results = model.process(image)             # Make prediction
```

```
image.flags.writeable = True                # Image is now writeable
image=cv2.cvtColor(image, cv2.COLOR_RGB2BGR) #COLOR COVERSION RGB 2 BGR
return image, results
def draw_styled_landmarks(image, results):
if results.multi_hand_landmarks:
for hand_landmarks in results.multi_hand_landmarks:
mp_drawing.draw_landmarks(
image,
hand_landmarks,
mp_hands.HAND_CONNECTIONS,
mp_drawing_styles.get_default_hand_landmarks_style(),
mp_drawing_styles.get_default_hand_connections_style())
def extract_keypoints(results):
if results.multi_hand_landmarks:
for hand_landmarks in results.multi_hand_landmarks:
rh = np.array([[res.x, res.y, res.z] for res in hand_landmarks.landmark]).flatten() if hand_landmarks else np.zeros(21*3)
return(np.concatenate([rh]))
# Path for exported data, numpy arrays
DATA_PATH = os.path.join('MP_Data')
actions = np.array(['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'])
no_sequences = 10
sequence_length = 10
```

3.22.3 DATA

```
from function import *
from time import sleep
for action in actions:
for sequence in range(no_sequences):
try:
os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))
except:
pass
# cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_hands.Hands(
model_complexity=0,
min_detection_confidence=0.5,
min_tracking_confidence=0.5) as hands:
# NEW LOOP
# Loop through actions
for action in actions:
# Loop through sequences aka videos
for sequence in range(no_sequences):
# Loop through video length aka sequence length
for frame_num in range(sequence_length):
# Read feed
# ret, frame = cap.read()
frame=cv2.imread('Image/{}/{}.png'.format(action,sequence))
# frame=cv2.imread('{}{}.png'.format(action,sequence))
# frame=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
# Make detections
image, results = mediapipe_detection(frame, hands)
```

```
#print(results)
# Draw landmarks
draw_styled_landmarks(image, results)
# NEW Apply wait logic
if frame_num == 0:
cv2.putText(image, 'STARTING COLLECTION', (120,200),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE_AA)
cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
# Show to screen
cv2.imshow('OpenCV Feed', image)
cv2.waitKey(200)
else:
cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
# Show to screen
cv2.imshow('OpenCV Feed', image)
# NEW Export keypoints
keypoints = extract_keypoints(results)
npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
np.save(npy_path, keypoints)
# Break gracefully
if cv2.waitKey(10) & 0xFF == ord('q'):
break
# cap.release()
cv2.destroyAllWindows()
```

3.22.4 TRAINING MODEL

```
from function import *
```

```
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.callbacks import TensorBoard
label_map = {label:num for num, label in enumerate(actions)}
# print(label_map)
sequences, labels = [], []
for action in actions:
for sequence in range(no_sequences):
window = []
for frame_num in range(sequence_length):
res=np.load(os.path.join(DATA_PATH,action,str(sequence),
"{}.npy".format(frame_num)))
window.append(res)
sequences.append(window)
labels.append(label_map[action])
X = np.array(sequences)
y = to_categorical(labels).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)
model = Sequential()
model.add(LSTM(64,
return_sequences=True, activation='relu', input_shape=(10,63)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))
```

```
res = [.7, 0.2, 0.1]
model.compile(optimizer='Adam',loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
model.fit(X_train, y_train, epochs=200, callbacks=[tb_callback])
model.summary()
model_json = model.to_json()
with open("model.json", "w") as json_file:
json_file.write(model_json)
model.save('model.h5')
```

3.22.5 APP

```
from function import *
from keras.utils import to_categorical
from keras.models import model_from_json
from keras.layers import LSTM, Dense
from keras.callbacks import TensorBoard
json_file = open("model.json", "r")
model_json = json_file.read()
json_file.close()
model = model_from_json(model_json)
model.load_weights("model.h5")
colors = []
for i in range(0,20):
colors.append((245,117,16))
print(len(colors))
def prob_viz(res, actions, input_frame, colors,threshold):
output_frame = input_frame.copy()
for num, prob in enumerate(res):
cv2.rectangle(output_frame, (0,60+num*40), (int(prob*100), 90+num*40),
colors[num], -1)
cv2.putText(output_frame, actions[num], (0, 85+num*40),
```

```
cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2, cv2.LINE_AA)
return output_frame
# 1. New detection variables
sequence = []
sentence = []
accuracy=[]
predictions = []
threshold = 0.8
cap = cv2.VideoCapture(0)
# cap = cv2.VideoCapture("https://192.168.43.41:8080/video")
# Set mediapipe model
with mp_hands.Hands(
model_complexity=0,
min_detection_confidence=0.5,
min_tracking_confidence=0.5) as hands:
while cap.isOpened():
# Read feed
ret, frame = cap.read()
# Make detections
cropframe=frame[40:400,0:300]
# print(frame.shape)
frame=cv2.rectangle(frame,(0,40),(300,400),255,2)
#frame=cv2.putText(frame,"Active
Region",(75,25),cv2.FONT_HERSHEY_COMPLEX_SMALL,2,255,2)
image, results = mediapipe_detection(cropframe, hands)
# print(results)
# Draw landmarks
# draw_styled_landmarks(image, results)
# 2. Prediction logic
keypoints = extract_keypoints(results)
```

```
sequence.append(keypoints)
sequence = sequence[-10:]
try:
if len(sequence) == 10:
res = model.predict(np.expand_dims(sequence, axis=0))[0]
print(actions[np.argmax(res)])
predictions.append(np.argmax(res))
#3. Viz logic
if np.unique(predictions[-10:])[0]==np.argmax(res):
if res[np.argmax(res)] > threshold:
if len(sentence) > 0:
if actions[np.argmax(res)] != sentence[-1]:
sentence.append(actions[np.argmax(res)])
accuracy.append(str(res[np.argmax(res)]*100))
else:
sentence.append(actions[np.argmax(res)])
accuracy.append(str(res[np.argmax(res)]*100))
if len(sentence) > 1:
sentence = sentence[-1:]
accuracy=accuracy[-1:]
#Viz probabilities
# frame = prob_viz(res, actions, frame, colors,threshold)
except Exception as e:
# print(e)
pass
cv2.rectangle(frame, (0,0), (300, 40), (245, 117, 16), -1)
cv2.putText(frame,"Output: -"+' '.join(sentence)+".join(accuracy), (3,30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
# Show to screen
cv2.imshow('OpenCV Feed', frame)
```

```
# Break gracefully
if cv2.waitKey(10) & 0xFF == ord('q'):
break
cap.release()
cv2.destroyAllWindows()
```

CHAPTER – 4

RESULTS AND DISCUSSION

4.1 DATA COLLECTION

Raw image is collected through web cam of ROI detecting the part where only image of hand is required.

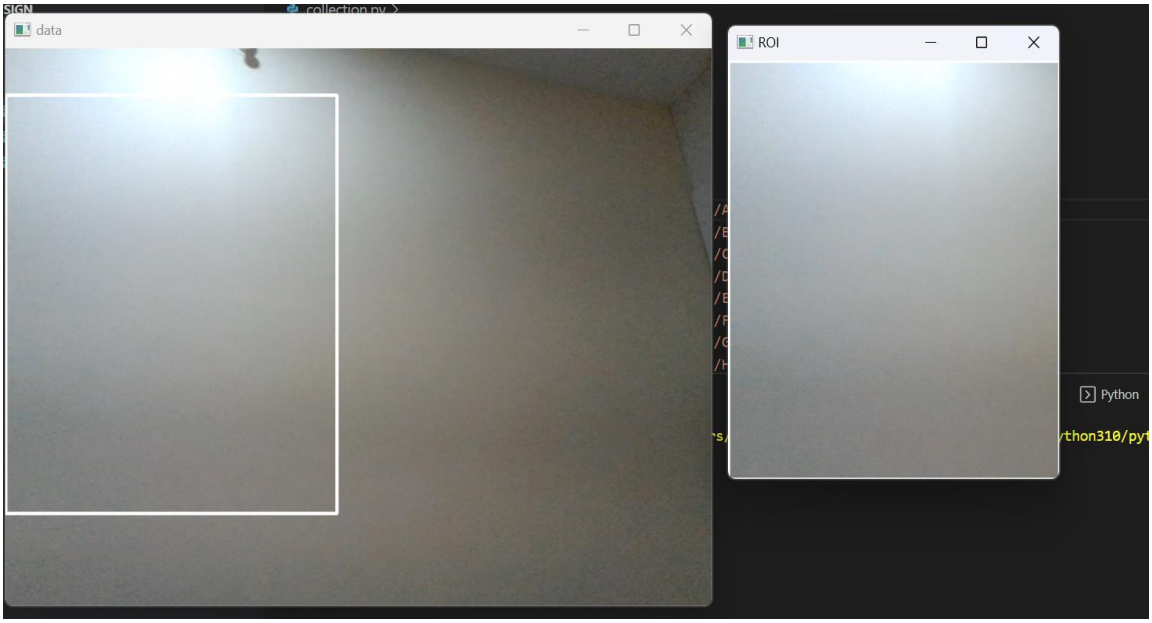


Fig. 4.1 A window having data and ROI layout

Here hand gesture for A is collected.

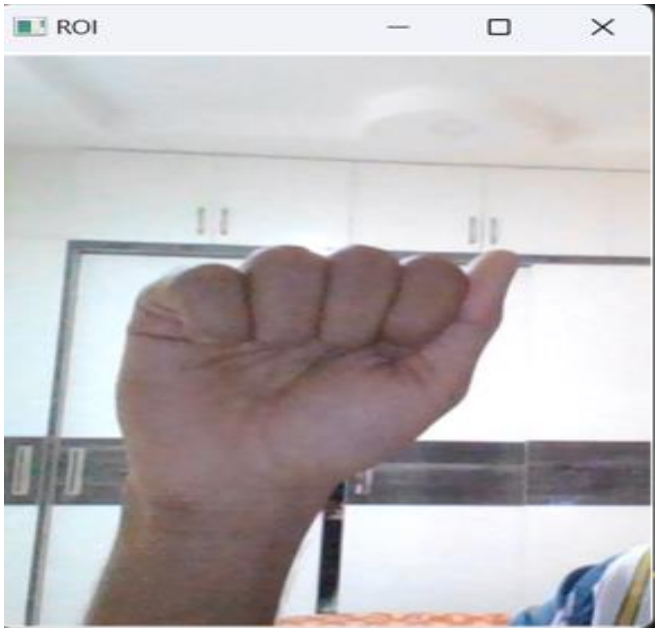


Fig. 4.2 Data collection of A through ROI

Then the image is functioned to various methods like grey scale image, blur image. This section is used to pinpoint the locations of the temporal restricts of this information after that segmentation. A webcam-captured hand image has been separated to get the hand region. The shapes, actions, and materials used in the various hand gestures vary & also, binary information of image.

4.2 DATA CONVERSION AND TRAINING

In this step the .png image is changed into a.npy file (matrix form), which includes a variety of properties to function under many circumstances. Here the data frames are collected.

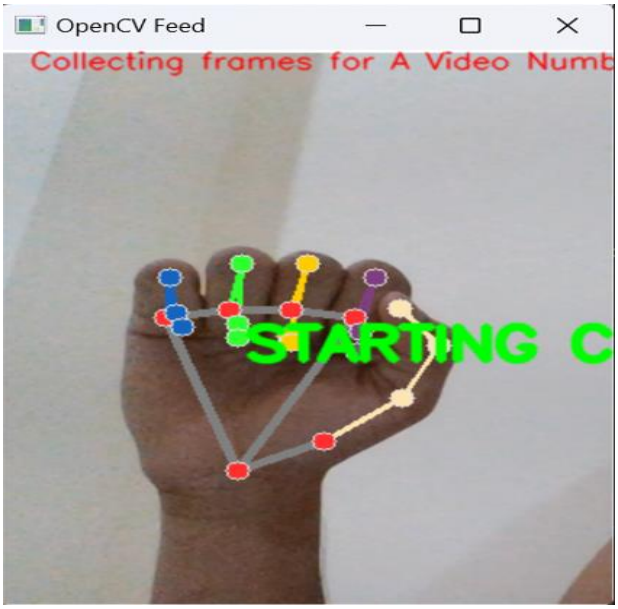


Fig. 4.3 Conversion of frames alphabet A

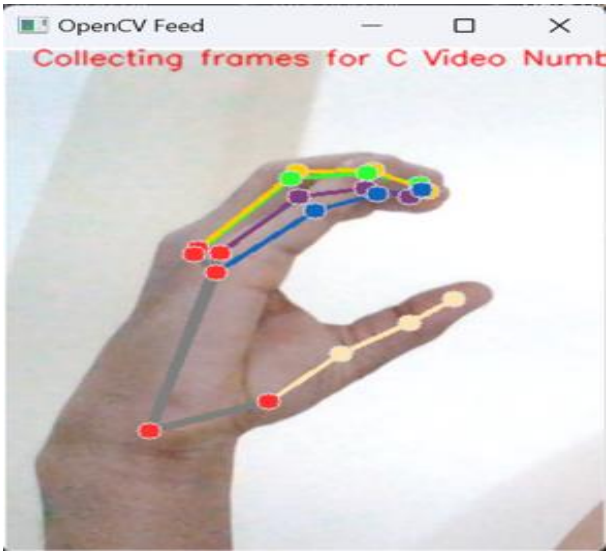


Fig. 4.4 Conversion of frames alphabet C

4.3 OUTPUT TESTING

Since they operate well with image data, CNNs, or artificial neural networks made up of convolutional layers, are the approach of choice for challenges involving image categorization. The categories training photographs were saved in the database. The image with the highest accuracy was generated as text after comparisons between the tested image and training dataset models.

In output image orange arrow represents alphabet and green arrow represents accuracy of image.

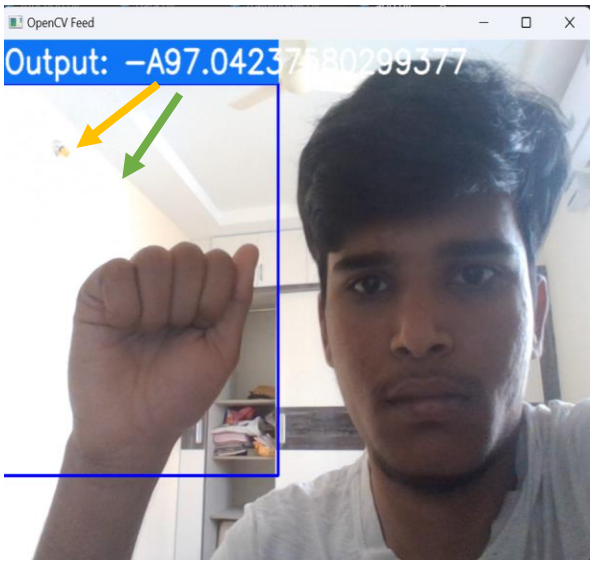


Fig. 4.5 Output of A

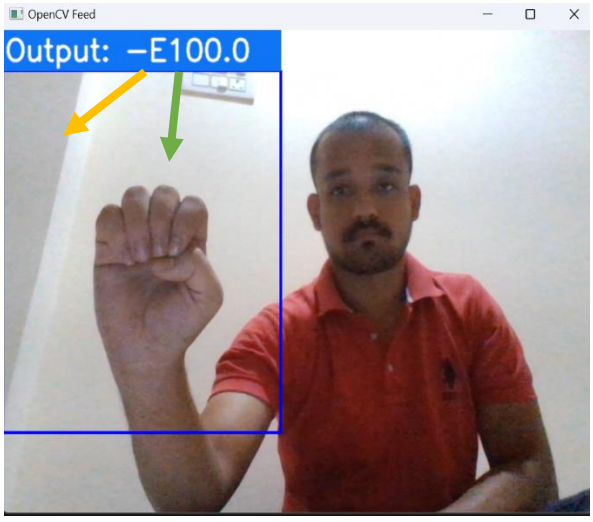


Fig. 4.6 Output of E

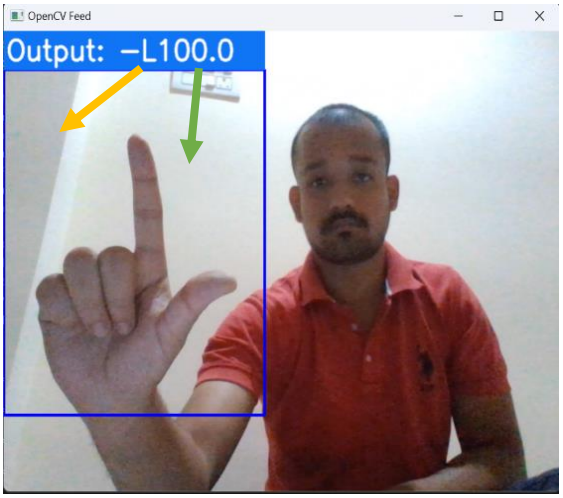


Fig. 4.7 Output of L

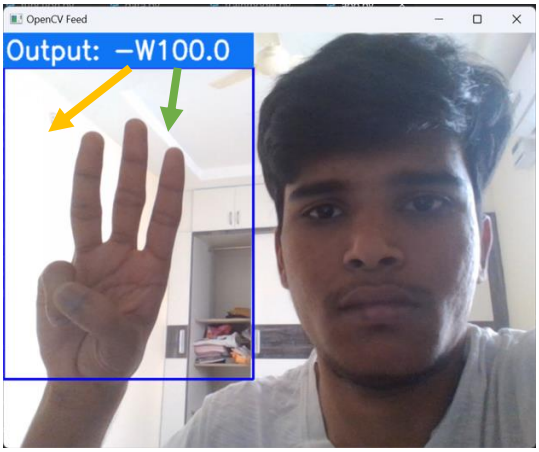


Fig. 4.8 Output of W

Some of the sample image outputs are given above.

A confusion matrix is a table that is used to define the performance of a classification algorithm. A confusion matrix visualizes and summarizes the performance of a classification algorithm. A confusion matrix is shown in Fig. 4.9, where benign tissue is called healthy and malignant tissue is considered cancerous. The laptop's embedded webcam gets utilized in our report, so it is a huge plus. Following is the confusion matrix of our findings.

						Predicted									Value										
		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
	A	147	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	2	0	0
	B	0	139	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	0
	C	0	0	152	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	D	0	0	0	145	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	E	0	0	0	0	152	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	F	0	0	0	0	0	135	0	0	0	0	0	4	0	0	0	0	0	1	0	0	2	10	0	0	0
C	G	0	0	0	0	0	0	150	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
o	H	1	0	0	0	0	0	7	143	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1
r	I	0	0	0	33	0	0	0	0	108	0	2	0	0	0	0	0	0	0	0	7	1	0	0	0	0
r	J	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	K	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	L	0	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0
t	M	0	0	0	0	0	0	0	0	0	0	2	0	152	0	0	0	0	0	0	0	0	0	0	0	0
	N	0	0	0	0	0	0	0	0	0	0	0	0	0	152	0	0	0	0	0	0	0	0	0	0	0
V	O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	154	0	0	0	0	0	0	0	0	0	0
a	P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0
l	Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	147	1	0	0	0	0	0	0	0
u	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	150	0	0	0	0	0	0	0
e	S	0	0	0	0	1	0	0	0	0	0	0	0	0	1	10	0	0	0	132	0	0	0	0	8	0
s	T	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	151	0	0	0	0	0
	U	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35	0	0	115	0	0	0	0
	V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	151	1	0	0
	W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	149	0	0
	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	148	0
	Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	151
	Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Algo 1

Fig. 4.9 Confusion Matrix

4.4 ADVANTAGES AND DISADVANTAGES

Advantages:

- Useful for deaf and dumb people.
- Quick expressing of messages.
- People can easily interpret the gesture of another person.
- Alternative computer interfaces.
- Used in sign language translation.
- Entertainment applications and automation systems.

Disadvantages:

- The model works well only in good lighting conditions.
- Plain background is needed for the model to detect with accuracy.
- Some gestures are difficult in understanding information might get distorted.
- Cannot make long explanation or conversation through it.
- It is one of the informal types of communication, where it is not suited for official purpose.

CHAPTER – 5

CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

This method has used images to discuss the gestures classification to recognition of gestures. The solution in this paper depends entirely on the shape parameters of the hand gesture. The implemented framework is effective in adjusting the lighting as well as the gesture orientation.

In this paper, we introduced a real-time system for transcribing text from American Sign Language (ASL) to English for sign language. The system recognises hand movements and converts them into text using a deep learning method that blends convolutional and recurrent neural networks. In both hand gesture recognition and phrase reconstruction, the system produced highly accurate results. The method, which may be expanded to other sign languages, holds the possibility to improve deaf people's accessibility and conversation. The classification of gestures and gesture recognition in this method has been discussed using photographs. The framework that has been put in place works effectively for altering both lighting and gesture orientation. By using a specific dataset, we were able to achieve a high accuracy of 97.0% in recognizing symbols.

5.2 FUTURE SCOPE

In order to achieve higher accuracy in recognizing gestures even in complex backgrounds, various background subtraction algorithms are being considered for implementation. Additionally, improvements in preprocessing are being explored to enhance the accuracy of gesture recognition in low light conditions. While there is still much room for research and development in CNN, three potential areas of focus include introducing more features of hierarchy and scale to improve model adaptability, improving the rate of dynamic gesture recognition, and reducing the dependence of the model on a large number of labeled data through unsupervised or semi-supervised learning. Looking to the future, a high-level semantic analysis will be applied to enhance the recognition capability of the system for complex human tasks.

REFERENCES

[1]. H. Cooper, B. Holt, and R. Bowden, “Sign language recognition,” in Visual Analysis of Humans, 2011.

[2]. F.S. Chen, C.M. Fu and C.L. Huang. "Hand gesture recognition using a real-time tracking method and hidden Markov models." Image and vision computing, vol. 21, no. 8, pp. 745-758, Aug. 2003.

[3]. G. Marin, F. Dominio, and P. Zanuttigh. "Hand gesture recognition with jointly calibrated leap motion and depth sensor." Multimedia Tools and Applications, vol. 75, no. 22, pp. 14991-15015, Nov. 2016.

[4]. P. Kumar, H. Gauba, P.P. Roy, and D.P. Dogra. "Coupled HMM-based multi-sensor data fusion for sign language recognition." Pattern Recognition Letters, vol. 86, pp. 1-8, Jan. 2017.

[5]. D. Lifeng, R. Jun, M. Qiushi, W. Lei, “The gesture identification based on invariant moments and SVM[J].” Microcomputer and Its Applications, vol. 31, no. 6, pp. 32-35, 2012.

[6]. M.A. Rahim, J. Shin, and M.R. Islam, “Human-Machine Interaction based on Hand Gesture Recognition using Skeleton Information of Kinect Sensor.” In Proceedings of the 3rd International Conference on Applications in Information Technology, ACM, pp. 75-79, Nov. 2018.

[7]. T. Yamashita, T. Watashe, “Hand posture recognition based on bottom-up structured deep convolutional neural network with curriculum learning.” IEEE international conference on image processing (ICIP), pp 853–857, Oct. 2014.

[8]. Y. Liao, P. Xiong, W. Min, W. Min, and J. Lu, “Dynamic Sign Language Recognition Based on Video Sequence with BLSTM-3D Residual Networks.” IEEE Access, 2019.

[9]. [Oscar Koller, Necati Camgoz, Hermann Ney, and Richard Bowden, “Weakly supervised learning with multi-stream cnn lstm-hmms to discover sequential parallelism in sign language videos,” IEEE Transactions on Pattern Analysis and Machine Intelligence, 04 2019.

[10]. S. Tornay, M. Razavi, N. C. Camgoz, R. Bowden, and M. Magimai.-Doss, “HMM-based approaches to model multichannel information in sign language inspired from articulatory features-based speech processing,” in Proc. of ICASSP,2019.

[11]. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition by Aurélien Géron

[12]. Practical Python and OpenCV+ Case Studies- An Introductory, Example Driven Guide to Image Processing and Computer Vision 4thEdition

[13]. T. Yang, Y. Xu, and "A. , Hidden Markov Model for Gesture Recognition", CMU-RI-TR-94 10, Robotics Institute, Carnegie Mellon Univ.,Pittsburgh,PA, May 1994.

[14]. Pujan Ziaie, Thomas M¨uller , Mary Ellen Foster , and Alois Knoll"A Na¨ive Bayes Munich,Dept. of Informatics VI, Robotics and Embedded Systems,Boltzmannstr. 3, DE-85748 Garching, Germany.

[15]. Lars Bretzner, Ivan Laptev, and Tony Lindeberg. 2002. Hand gesture recognitionusing multiscale color features, hieracrchichal models and particle in Ąltering. In proceedings the fifth IEEE International Conference on Automatic Face and Gesture Recognition, 423-428.

[16]. Nasser H. Dardas, and Nicolas D. Georganas. 2011. Real-time hand gesture detection and recognition using bag-of-features and support vector machine techniques. IEEE Transactions on Instrumentation and measurement, 60,11 (November 2011), 3592-3607.

[17]. Sebastian Nowozin, Pushmeet Kohli, and J. D. J. Shotton. 2017. Gesture detection and recognition. U.S. Patent 9,619,035, issued April 11.

[18]. Trygve Thomessen, Mihoko Niitsuma, Keita Suzuki, Takashi Hatano, and Hideki Hashimoto. 2015. Towards virtual presence based on multimodal man-machine communication: A remote operation support system for industrial robots. IFACPapersOnLine, 48,19 (January 2015), 172-177.

[19]. C. Gulcehre and Y. Bengio, "Knowledge matters: Importance of prior information for optimization", arXivpreprint arXiv:1301.4083, 2013.

[20]. Mohammed Waleed Kalous, Machine recognition of Auslan signs using PowerGloves: Towards large-lexicon recognition of sign language.