**C H A P T E R 1**

Introduction

**[1.1] Introduction**

Hand gesture recognition is one of the most advanced domains in which computer vision and artificial intelligence has helped to improve communication. From levers, switches, buttons, and now touch displays, the way we interact has been a constant focus in how we design the machines we use in our daily lives. Currently, it seems like the next evolution of human-computer interaction is focused on interacting with computers without physical touch, such as talking or gesturing instead of manually clicking a button or turning a switch. There are many products that are already used by many that emphasize wireless control, such as voice assistants. For our project we attempted to figure out the ways we could utilize the gestures in a way that was actually useful and easy to use. We created a program that uses image feed from your webcam to detect your hand gestures and then perform appropriate behavior based on it. Touchless Human-Machine Interface (HMI) is a field with applications in robotics, computer gaming and sign-language interpretation. A person’s gesture must be recognized by a device for effective dialogue between a human and a computer. A well-known use of hand gesture recognition is the identification of human data, namely, sign language. Sign language is a visual language in which ideas are communicated by a series of expressive hand motions in a certain order. Moreover, touchless HMI is very useful in sterile environments such as doctors or surgeons needing to interact with reports with computers without introducing contamination. Most of the time, buttons, or touch screens are wrapped in plastic and the surgeons need to change their gloves each time they have to use the computers. It is quite common for surgeons to ask colleagues or nurses, who are in another room to interact with the computers for moving images. The progress of the gesture recognition systems plays a vital role in the development of computer and human interaction. By definition, gestures originate from the body and include movement of fingers, hands, arms, legs, eyes, and even facial expressions. So, Hand gestures recognition is a subset of the larger gesture recognition problem in Computer Science. The main aim of this real-time hand gesture touchless recognition application is to classify and recognize the gestures and then perform the particular task /movement assigned to them using the Deep Learning method. This is possible with the help of touchless gesture recognition applications where you can access the images without coming into contact with the screen by showing the gestures interfaced in your machine.

**C H A P T E R 2**

Literature Review

For sign language recognition in the human-computer relationship, many approaches are present in the literature. The basic goal of these technologies is to facilitate communication by the accurate interpretation of the user’s gestures/signs. The following steps are included in this methodology: capture and preprocessing, gesture representation, feature extraction, and categorization.

**[2.1.]****Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks**

**Author:** [**Okan Köpüklü**](https://paperswithcode.com/author/okan-kopuklu-1)**,** [**Ahmet Gunduz**](https://paperswithcode.com/author/ahmet-gunduz)**,** [**Neslihan Kose**](https://paperswithcode.com/author/neslihan-kose)**,** [**Gerhard Rigoll**](https://paperswithcode.com/author/gerhard-rigoll)

**Dated: 29 Jan 2019**

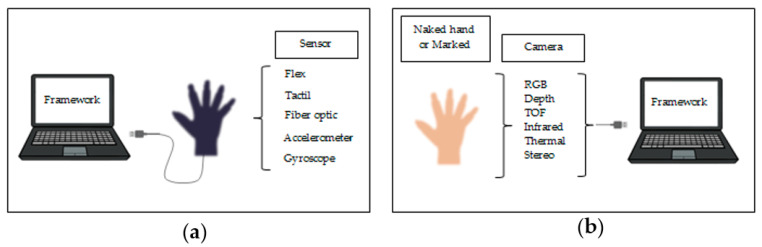
Real-time recognition of dynamic hand gestures from video streams is a challenging task since (i) there is no indication when a gesture starts and ends in the video, (ii) performed gestures should only be recognized once, and (iii) the entire architecture should be designed considering the memory and power budget. In this work, they have addressed these challenges by proposing a hierarchical structure enabling offline-working convolutional neural network (CNN) architectures to operate online efficiently by using a sliding window approach. The proposed architecture consists of two models: (1) A detector which is a lightweight CNN architecture to detect gestures and (2) a classifier which is a deep CNN to classify the detected gestures. In order to evaluate the single-time activations of the detected gestures, they propose to use Levenshtein distance as an evaluation metric since it can measure misclassifications, multiple detections, and missing detections at the same time. We evaluate our architecture on two publicly available datasets - EgoGesture and NVIDIA Dynamic Hand Gesture Datasets - which require temporal detection and classification of the performed hand gestures.

The general workflow of the proposed two-model hierarchical architecture. Sliding windows with stride s run through incoming video frames where a detector queue is placed at the very beginning of the classifier queue. If the detector recognizes an action/gesture, then the classifier is activated. The detector’s output is post-processed for a more robust performance, and the final decision is made using a single-time activation block where only one activation occurs per performed gesture. 1) Detector: The purpose of the detector is to distinguish between gesture and no gesture classes by running on a sequence of images, which detector queue masks. Its main and only role is to act as a switch for the classifier model, meaning that if it detects a gesture, then the classifier is activated and fed by the frames in the classifier queue. [1]

**[2.2.] Hand-Gesture Recognition based on Computer Vision**

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

**Dated: 23 July 2020**

****

As illustrated in figure (1) , hand gestures for human–computer interaction (HCI) started with the invention of the data glove sensor. It offered simple commands for a computer interface. The gloves used different sensor types to capture hand motion and position by detecting the correct coordinates of the location of the palm and fingers. Various sensors using the same technique based on the angle of bending were the curvature sensor, angular displacement sensor, optical fiber transducer, flex sensors and accelerometer sensor. These sensors exploit different physical principles according to their type.

Although the techniques mentioned above have provided good outcomes, they have various limitations that make them unsuitable for the elderly, who may experience discomfort and confusion due to wire connection problems. In addition, elderly people suffering from chronic disease conditions that result in loss of muscle function may be unable to wear and take off gloves, causing them discomfort and constraining them if used for long periods. These sensors may also cause skin damage, infection or adverse reactions in people with sensitive skin or those suffering burns. Moreover, some sensors are quite expensive.

These drawbacks led to the development of promising and cost-effective techniques that did not require cumbersome gloves to be worn. These techniques are called camera vision-based sensor technologies. With the evolution of open-source software libraries, it is easier than ever to detect hand gestures that can be used under a wide range of applications like clinical operations, sign language, robot control, virtual environments, home automation, personal computer and tablet, gaming. These techniques essentially involve replacement of the instrumented glove with a camera. Different types of camera are used for this purpose, such as RGB camera, time of flight (TOF) camera, thermal cameras or night vision cameras.

In figure (2), the camera vision based sensor is a common, suitable and applicable technique because it provides contactless communication between humans and computers. Different configurations of cameras can be utilized, such as monocular, fisheye, TOF and IR. However, this technique involves several challenges, including lighting variation, background issues, the effect of occlusions, complex background, processing time traded against resolution and frame rate and foreground or background objects presenting the same skin color tone or otherwise appearing as hands. [2]

**[2.3.] Hand Gesture Recognition using Deep Learning Neural Networks**

**Author: Norah Meshari Alnaim**

**Dated:** **December 2019**

Various studies have previously been conducted relating to hand gestures. Some studies proposed different techniques to implement the hand gesture experiments. For image processing there are multiple tools to extract features of images, as well as Artificial Intelligence which has varied classifiers to classify different types of data. 2D and 3D hand gestures request an effective algorithm to extract images and classify various mini gestures and movements. This research discusses this issue using different algorithms. To detect 2D or 3D hand gestures, this research proposed image processing tools such as Wavelet Transforms and Empirical Mode Decomposition to extract image features. The Artificial Neural Network (ANN) classifier which used to train and classify data besides Convolutional Neural Networks (CNN). These methods were examined in terms of multiple parameters such as execution time, accuracy, sensitivity, specificity, positive predictive value, negative predictive value, positive likelihood, negative likelihood, receiver operating characteristic, area under ROC curve and root mean square. This research discusses four original contributions in the field of hand gestures. The first contribution is an implementation of two experiments using 2D hand gesture video where ten different gestures are detected in short and long distances using an iPhone 6 Plus with 4K resolution. The experiments are performed using WT and EMD for feature extraction while ANN and CNN for classification. The second contribution comprises 3D hand gesture video experiments where twelve gestures are recorded using a holoscopic imaging system camera. The third contribution pertains to experimental work carried out to detect seven common hand gestures. Finally, disparity experiments were performed using the left and the right 3D hand gesture videos to discover disparities. The results of comparison show the accuracy results of CNN being 100% compared to other techniques. CNN is clearly the most appropriate method to be used in a hand gesture system. [3]

**[2.4.] Gesture-Based Interaction Using Touchless Technology for Medical Application**

**Author: Abdelkrim Belhaoua, Jean-Pierre Radouz, Florian Pereme**

**Dated: Barcelona, Spain, July 13 - 14, 2015**

In this paper, we present a touchless human-machine interaction dedicated to a science museum for children, in the short-term, and surgeons in an operating room in the long-term. In fact, surgeons need to review medical images during minimally invasive surgery without interacting directly with the medical devices. Touchless technology is an interesting solution despite the many challenges as low accuracy, bad lighting conditions, occlusion and real time computing. In order to perform such an application, an educational game, so-called Anatomia, is developed using Kinect and interacting with a 3D model of a human body. Many tests and surveys are carried out from children, adults and surgeons to evaluate the touchless technology and improve its ergonomics.

Keywords: Touchless technology, Kinect, minimally invasive surgery, interaction.

Anatomia

Anatomia is an educational game for children (8-15 years old) using Kinect 2 and interacting with a 3D model of a human body, provided by IRCAD (Research Institute against Digestive Cancer). The basic idea is that the children tend to be more interested in the medical field. Can the children point out a heart on the 3D model of the human body? This game brings them some entertaining and stimulating games to improve their medical knowledge. The quizzes include questions about the digestive system and the respiratory system (Figure 1). The selection of the organs is performed using the identiﬁcation of the right hand-state (open/fristed). This application falls within the framework of a research program in collaboration with IRCAD (Re- search Institute against cancers of the digestive system) and the LeVaisseau Museum of Strasbourg where the application was installed in September 2015.[4]

**[2.5.] Hand-gesture-based sterile interface for the operating room using contextual cues for the navigation of radiological images**

**Author: Mithun George Jacob, Juan Pablo Wachs**

**Dated: 2012 Dec 18**

This presents a method to improve the navigation and manipulation of radiological images through a sterile hand gesture recognition interface based on attentional contextual cues. Computer vision algorithms were developed to extract intention and attention cues from the surgeon's behavior and combine them with sensory data from a commodity depth camera. The developed interface was tested in a usability experiment to assess the effectiveness of the new interface. An image navigation and manipulation task was performed, and the gesture recognition accuracy, false positives and task completion times were computed to evaluate system performance. Experimental results show that gesture interaction and surgeon behavior

analysis can be used to accurately navigate, manipulate and access MRI images, and therefore this modality could replace the use of keyboard and mice-based interfaces.

Step 1: lexicon generation—An ethnographic study was conducted with 10 surgeons from the University's School of Veterinary Medicine to collect a set of gestures natural for the primary user of the system (clinicians and surgeons). First, surgeons were asked to specify functions they perform on MRI images in typical surgeries that would be useful in the OR. When asked about gestures that could be effective if the interface were only controlled via hand or body gestures, each surgeon provided a set of gestures corresponding to each aforementioned function. Each surgeon clearly showed the gesture assigned to the function (requiring one or both hands), which was recorded.

Step 2: development of gesture recognition software—Skeletal joints were tracked using a software library (OpenNI, V.1.3.2.3) using the Kinect sensor, which fits a skeleton model to the user.

Step 3: validation of the technology—Three experiments were conducted to validate the main hypothesis that contextual information can be used to accurately detect gestures evoked by the user. [5]

**[2.6.] A Touchless system for image visualization during surgery**

**Author:** [**C Massaroni**](https://pubmed.ncbi.nlm.nih.gov/?term=Massaroni+C&cauthor_id=30441652)**,** [**F Giurazza**](https://pubmed.ncbi.nlm.nih.gov/?term=Giurazza+F&cauthor_id=30441652)**,** [**M Tesei**](https://pubmed.ncbi.nlm.nih.gov/?term=Tesei+M&cauthor_id=30441652)**,** [**E Schena**](https://pubmed.ncbi.nlm.nih.gov/?term=Schena+E&cauthor_id=30441652)**,** [**F Corvino**](https://pubmed.ncbi.nlm.nih.gov/?term=Corvino+F&cauthor_id=30441652)**,** [**M Meneo**](https://pubmed.ncbi.nlm.nih.gov/?term=Meneo+M&cauthor_id=30441652)**,** [**L Corletti**](https://pubmed.ncbi.nlm.nih.gov/?term=Corletti+L&cauthor_id=30441652)**,** [**R Niola**](https://pubmed.ncbi.nlm.nih.gov/?term=Niola+R&cauthor_id=30441652)**,** [**R Setola**](https://pubmed.ncbi.nlm.nih.gov/?term=Setola+R&cauthor_id=30441652)

**Dated: 2018 Jul**

Today clinicians may access large medical datasets, but very few systems have been designed to allow a practical and efficient exploration of data directly in critical medical environments such as operating rooms (OR). This work aims to assess during tests in laboratory and clinical settings a Surgery Touchless System (STS). This system allows clinicians to interact with medical images by using two different approaches: a gesture recognition and a voice recognition based system. These two methods are based on the use of a Microsoft Kinect and of a selective microphone, respectively. The STS allows navigating in a specifically designed interface, to perform several tasks, among others, to manipulate biomedical images. In this article, we assessed both the recognition approaches in the laboratory with 5 users. In addition, the STS was tested using only the voice-based recognition approach in clinical settings. The assessment was performed during three procedures by two interventional radiologists. The five volunteers and the 2 radiologists filled two questionnaires to assess the system. The system usability was positively evaluated in laboratory tests. From clinical trials emerged that the STS was considered safe and useful by both the radiologists: they used the system an averaged number of times of 10 and 15 for patients, and found the system useful. These promising results allow considering this system useful for providing information not otherwise accessible and limiting the impact of human error during the operation. Future work will be focused on the use of the STS on a high number and different types of procedure. [6]

**[2.7.] Real-Time Hand Gesture Interface for Browsing Medical Images**

**Author: Juan Wachs, Helman Stern, Yael Edan, Michael Gillam, Craig Feied, Mark Smith, Jon Handler**

**Dated:3 March 2007**

A gesture interface is developed for users, such as doctors/surgeons, to browse medical images in a sterile medical environment. A vision-based gesture capture system interprets user’s gestures in real-time to manipulate objects in an image visualization environment. A color distribution model of the gamut of colors of the user's hand or glove is built at the start of each session resulting in an independent system. The gesture system relies on real-time robust tracking of the user’s hand based on a color-motion fusion model, in which the relative weight applied to the motion and color cues are adaptively determined according to the state of the system. Dynamic navigation gestures are translated to commands based on their relative positions on the screen.

A web-camera placed above a screen captures a sequence of images of the hand. The hand is tracked by a tracking module which segments the hand from the background using color and motion cues. This is followed by black/white thresholding.

The location of the hand in each image is represented by the 2D coordinates of its centroid. This spatio-temporal data is mapped into a ‘flick gesture’.

A flick gesture is the rapid movement of the hand from a neutral position to a specific direction. The direction of the ‘flick’ is used to navigate through a visual image data browser. Other actions/commands such as: zoom, rotate are recognized by extracting features from the spatio-temporal data of the gestures. With these actions/commands doctors can bring up X-rays images, select a patient record from the database or move objects and windows on the screen. A two-layer architecture is used.

Real time feedback and operation - during surgery the system should be fast and enable the surgeon to obtain visual feedback of the evoked gestures. [7]

**[2.8.] Classifiers for hand gesture recognition**

**Author:** [**Helman Stern**](https://www.researchgate.net/profile/Helman-Stern)

**Dated: November 2010**

The use of hand gestures provides an attractive alternative to cumbersome interface devices for human -machine interaction. The current research addresses real time gesture classification and aims to develop an algorithm capable of accurate classification of gestural control commands. Two different classifiers were developed for classifying a gesture vocabulary of eight dynamic hand gestures. The classifiers developed were: K-means + rule-based classifier and longest common subsequence (LCS) classifier. An experiment was performed to determine the recognition accuracy of the classifiers in which a test set of 180 trajectories were classified. The obtained accuracies are 90 and 94 percent for the K-means and LCS classifiers, respectively. [8]

**[2.9] How is Machine Learning and Deep Learning different from each other?**

In practical terms, deep learning is just a subset of machine learning. In fact, deep learning is machine learning and functions in a similar way (hence why the terms are sometimes loosely interchanged). However, its capabilities are different.

While basic machine learning models do become progressively better at performing their specific functions as they take in new data, they still need some human intervention. If an AI algorithm returns an inaccurate prediction, then an engineer has to step in and make adjustments. With a deep learning model, an algorithm can determine whether or not a prediction is accurate through its own neural network—no human help is required.

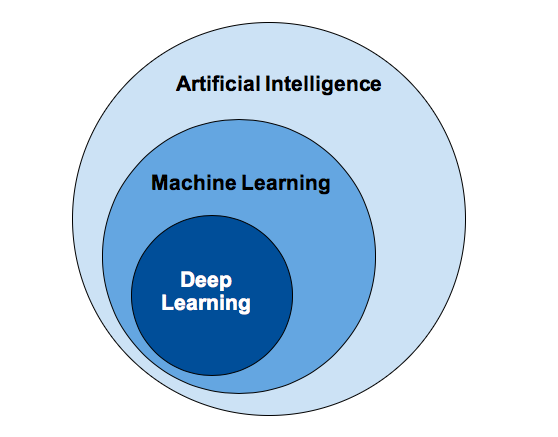
****

Figure (1)

Machine Learning and Deep Learning are the two main concepts of Data Science and the subsets of Artificial Intelligence. Most people think of machine learning, deep learning, and artificial intelligence as the same buzzwords. But in actuality, all these terms are different but related to each other.

Machine learning is a part of artificial intelligence and growing technology that

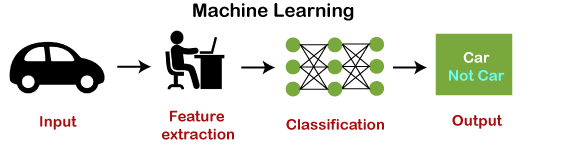
enables machines to learn from past data and perform a given task automatically.

“Machine Learning allows the computers to learn from the experiences on its own, use statistical methods to improve the performance and predict the output without being explicitly programmed.”

Some useful ML algorithms are:

* Decision Tree algorithm
* Naïve Bayes
* Random Forest
* K-means clustering
* KNN algorithm
* Apriori Algorithm, etc.

### How does Machine Learning work?

The working of machine learning models can be understood by the example of identifying the image of a cat or dog. To identify this, the ML model takes images of both cat and dog as input, extracts the different features of images such as shape, height, nose, eyes, etc., applies the classification algorithm, and predicts the output. Consider the below image:

Deep Learning is the subset of machine learning or can be said as a special kind of machine learning. It works technically in the same way as machine learning does, but with different capabilities and approaches. It is inspired by the functionality of human brain cells, which are called neurons, and leads to the concept of artificial neural networks. It is also called a deep neural network or deep neural learning.

In deep learning, models use different layers to learn and discover insights from the data.

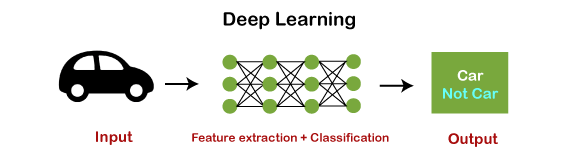
Some popular applications of deep learning are self-driving cars, language translation, natural language processing, etc.

Some popular deep learning models are:

* Convolutional Neural Network
* Recurrent Neural Network
* Autoencoders
* Classic Neural Networks, etc.

### How Deep Learning Works?

We can understand the working of deep learning with the same example of identifying cat vs. dog. The deep learning model takes the images as the input and feed it directly to the algorithms without requiring any manual feature extraction step. The images pass to the different layers of the artificial neural network and predict the final output.

Consider the below image:

**Summary**

From the above several reviews we understood the importance of hand gestures in today's world. As the world is growing in different technologies we are also growing through and evolving through buttons then touchscreens and then now touchless exploration. We can use human-computer interaction in various applications. The main application is sign language which can help several needy people. But the main focus we considered after going all through the reviews and research is ‘the touchless exploration of medical images in the field of medicine’, this is the highlighted point that we have worked upon, and the reason that we have chosen this application is considering the doctors views and point.

Concerned doctors, nowadays the disruption of the environment is high due to which spread of germs and viruses is also high. Floods, the Corona virus and the related variant that is spreading all over the world is breathtaking. The reason is contamination of dust, germs and viruses. The doctors are front leaders to save lives, so to think about them we have built an application, that is they can explore the medical image of their concern touchless without coming into contact with the screen.

We have used Convolution Neural Network which is part of deep learning as an main architecture. We have collected the dataset, resized it to the same size and then pushed into the internal layers. The internal layers then extract the features required on its own process and classify it accordingly. Here we used multi classification because we have four classes.

To perform this we required to install several packages and libraries to complete the work efficiently. In this project we have worked upon image manipulation in real time. After working upon it we have almost got the full accuracy. Soon we will be extending this onto the video manipulation, that is by extracting the images from the video and then working upon it.

The best part of our project is we have not used any kind glove that consists of sensors. There are no such wire things or the portability issue. As these gloves and sensors cost high here, our project is cost friendly. Soon further effective features will be working and implemented into it.

**C H A P T E R 3**

System Implementation

**[3.1.] Convolutional Neural Network Architecture**

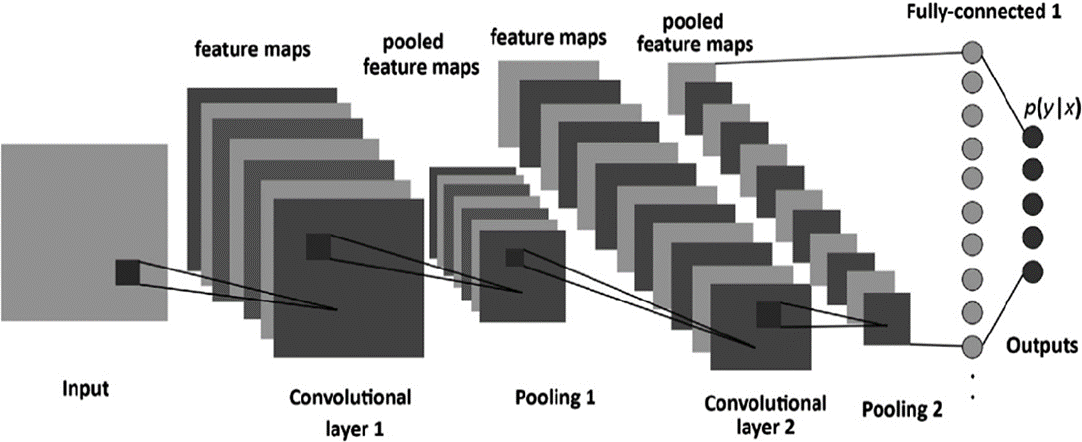


 Figure (2)

A convolutional neural network is a specific kind of neural network with multiple layers. It processes data that has a grid-like arrangement then extracts important features. One huge advantage of using CNNs is that you don't need to do a lot of pre-processing on images. With most algorithms that handle image processing, the filters are typically created by an engineer based on heuristics. CNNs can learn what characteristics in the filters are the most important. That saves a lot of time and trial and error work since we don't need as many parameters.

It doesn't seem like a huge savings until you are working with high resolution images that have thousands of pixels. The convolutional neural network algorithm's main purpose is to get data into forms that are easier to process without losing the features that are important for figuring out what the data represents. This also makes them great candidates for handling huge datasets.

A big difference between a CNN and a regular neural network is that CNNs use convolutions to handle the math behind the scenes. A convolution is used instead of matrix multiplication in at least one layer of the CNN. Convolutions take two functions and return a function.

CNNs work by applying filters to your input data. What makes them so special is that CNNs are able to tune the filters as training happens. That way the results are fine-tuned in real time, even when you have huge data sets, like with images. Since the filters can be updated to train the CNN better, this removes the need for hand-created filters. That gives us more flexibility in the number of filters we can apply to a data set and the relevance of those filters. Using this algorithm, we can work on more sophisticated problems like face recognition.

One of things that prevents a lot of problems from using CNNs is a lack of data. While networks can be trained with relatively few data points (~10,000 >), the more data there is available, the better tuned the CNN will be. To clean and label in order for the CNN to be able to use them. That's what makes them so expensive to work with.

An RGB image is nothing but a matrix of pixel values having three planes whereas a grayscale image is the same but it has a single plane.

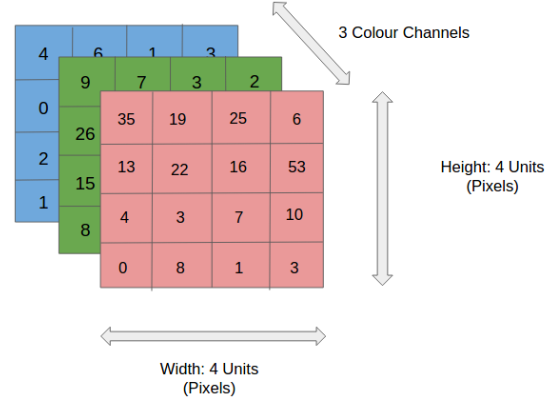


Figure (3)

## **[3.2.] How Convolutional Neural Networks Work?**

Convolutional neural networks are based on neuroscience findings. They are made of layers of artificial neurons called nodes. These nodes are functions that calculate the weighted sum of the inputs and return an activation map. This is the convolution part of the neural network.

Each node in a layer is defined by its weight values. When you give a layer some data, like an image, it takes the pixel values and picks out some of the visual features.

When you're working with data in a CNN, each layer returns activation maps. These maps point out important features in the data set. If you give the CNN an image, it'll point out features based on pixel values, like colors, and give you an activation function.

Usually with images, a CNN will initially find the edges of the picture. Then this slight definition of the image will get passed to the next layer. Then that layer will start detecting things like corners and color groups. Then that image definition will get passed to the next layer and the cycle continues until a prediction is made.

As the layers get more defined, this is called max pooling. It only returns the most relevant features from the layer in the activation map. This is what gets passed to each successive layer until you get the final layer.

Let’s go layer-wise to get deep insights about CNN.

First, there a few things to learn from layer 1 that isstriding and padding, let’s see each of them in brief with examples

In the input matrix of 5×5 and a filter of matrix 3x3, filter is a set of weights in a matrix applied on an image or a matrix to obtain the required features.

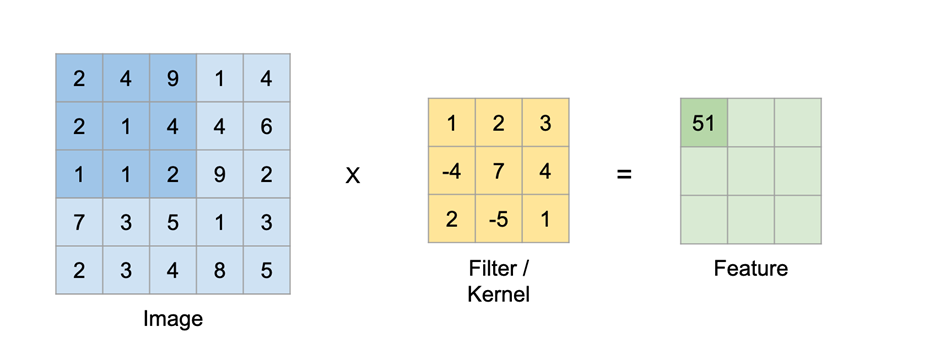
We always take the sum or average of all the values while doing a convolution.

Figure (4)

Here the input is of size 5×5 after applying a 3×3 kernel or filters you obtain a 3×3 output feature map.

**[3.2.1.] Padding**

While applying convolutions we will not obtain the output dimensions the same as input we will lose data over borders so we append a border of zeros and recalculate the convolution covering all the input values.

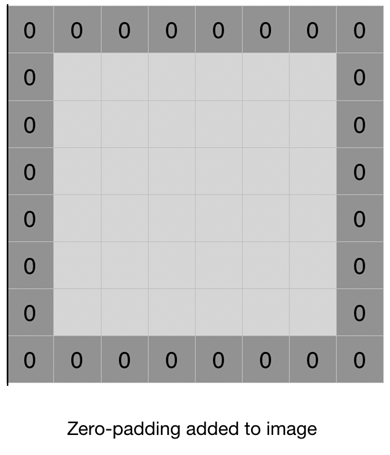


Figure (5)

When performing a standard convolution operation, the image shrinks by a factor equivalent to the filter size plus one. If we take an image of width and height 6, and a filter of width and height 3, the image shrinks by the following factor.

**6 - 3 +1 =4**

**6−3+1=4**

The reason for the shrinking image is that a 3×3 filter cannot slide all three of its columns over the first two horizontal pixels in the image. The same problem exists with regard to the rows and the vertical pixels.

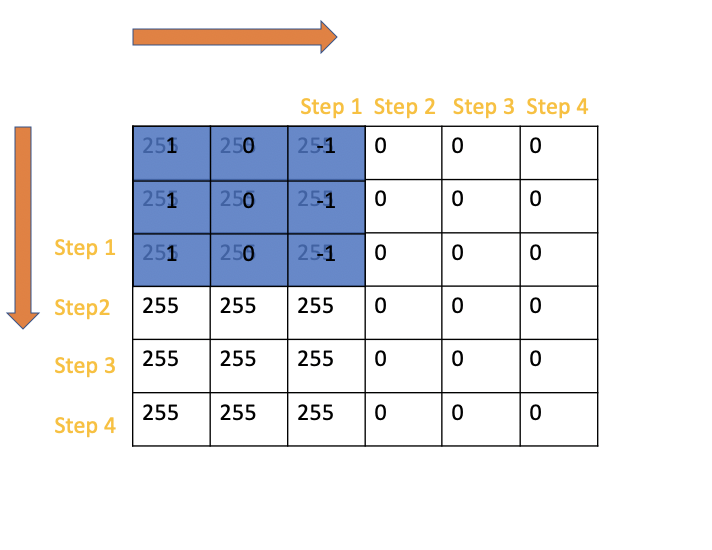


Figure (6)

There are only 4 steps left for the filter until it reaches the end of the image, both vertically and horizontally. As a consequence, the resulting image will only have 4×4 dimensions instead of 6×6. The general formula for calculating the shrinkage of the image dimensions m x m based on the kernel size f x f, can be calculated as follows:

(m\times m) \* (f\times f) = (m-f+1)\*(m-f+1)

(*m*×*m*)∗(*f*×*f*)=(*m*−*f*+1)∗(*m*−*f*+1)

**[3.2.2.] Striding**

Sometimes we do not want to capture all the data or information available so we skip some neighboring cells. Here the input matrix or image is of dimensions 8×8 with a filter of 4x4.

Stride is a component, or tuned for the compression of images and video data. Stride is a parameter of the neural network's filter that modifies the amount of movement over the image or video. For example, if a neural network's stride is set to 1, the filter will move one pixel, or unit, at a time. The size of the filter affects the encoded output volume, so stride is often set to a whole integer, rather than a fraction or decimal.

Basically, a computer reads an image from left to right and from top to bottom. Therefore, it starts from the top-left corner all the way to the bottom-right corner.

We define how far the filter moves from one position to the next position by “stride”. Let’s look at an example. The red square is a filter. The computer is going to use this filter to scan the image.

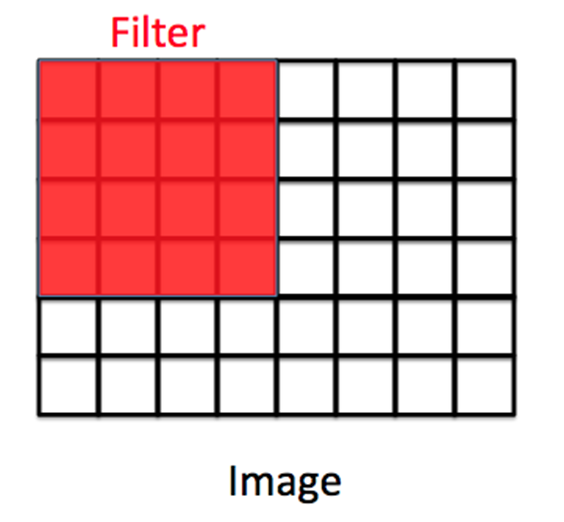


Figure (7.1)

If stride = 1, the filter will move one pixel.

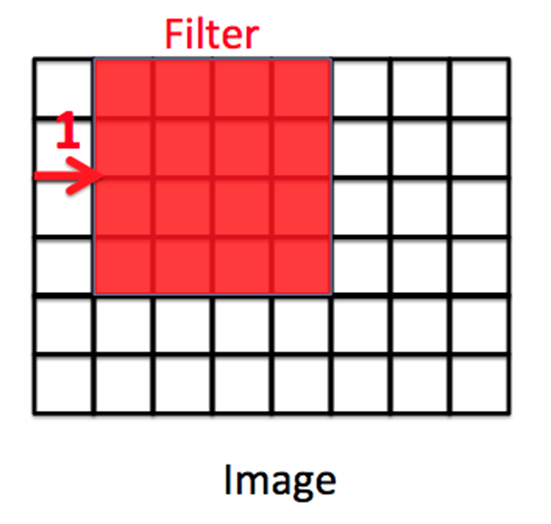


Figure (7.2)

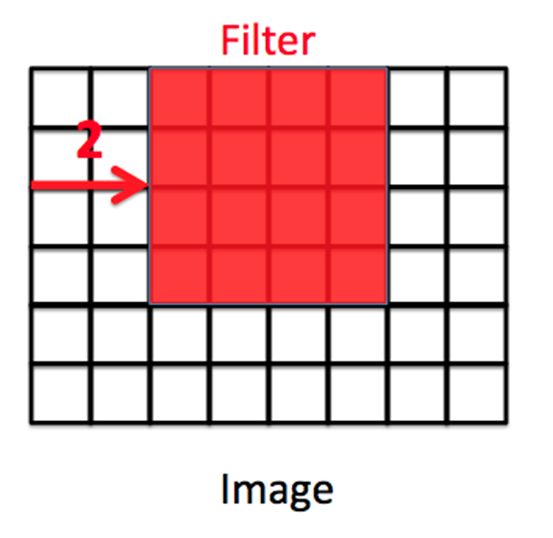
If stride = 2, the filter will move two pixels.

Figure (7.3)

**[3.2.3.] imageInputLayer:**

An image InputLayer is the place you initialize the size of input image, here, 100-by-100-by-1 is used. These numbers represent height, width, and the number of channels. In this case, input data is a grayscale image.

**[3.2.4] Convolutional Layer:**

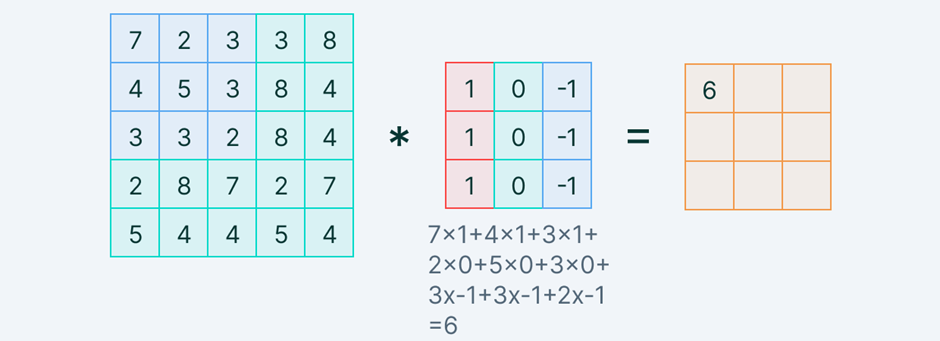
In a convolutional neural network, a convolutional layer is responsible for the systematic application of one or more filters to an input.

The multiplication of the filter to the input image results in a single output. The input is typically three-dimensional images (e.g., rows, columns and channels), and in turn, the filters with fewer rows and columns than the input image. As such, the filter is repeatedly applied to each part of the input image, resulting in a two-dimensional output map of activations, called a feature map.

Keras provides an implementation of the convolutional layer called a Conv2D.

It requires that you specify the expected shape of the input images in terms of rows (height), columns (width), and channels (depth) or [rows, columns, channels].

The filter contains the weights that must be learned during the training of the layer. The filter weights represent the structure or feature that the filter will detect and the strength of the activation indicates the degree to which the feature was detected.

Figure (8)

In the convolution layer, we move the filter/kernel to every possible position on the input matrix. Element-wise multiplication between the filter-sized patch of the input image and filter is done, which is then summed.

**[3.2.5.] Pooling Layers:**

We can use the pooling layers with the convolutional layer so the feature map generated by the convolutional layer is high dimensional can be reduced in the low dimensional and rest computational work will cost a low amount of effort. Pooling layers summarizes the featured map so that the model will not need to be trained on precisely positioned features. This makes a model more reliable and robust.

**Max Pooling**

Max pooling is an operation that selects the maximum elements from the region of the feature map covered by the filter. Thus, the output after max pooling layer would be a feature map containing the most prominent features of the previous feature map.

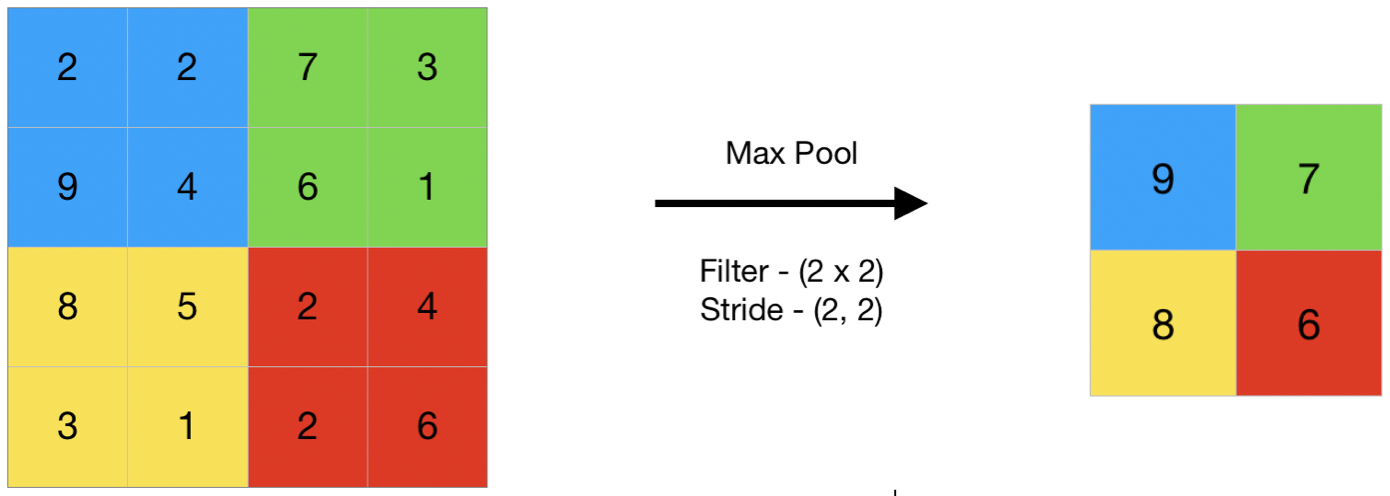


Figure (9.1)

**Average Pooling**

Average pooling computes the average of the elements present in the region of the feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.

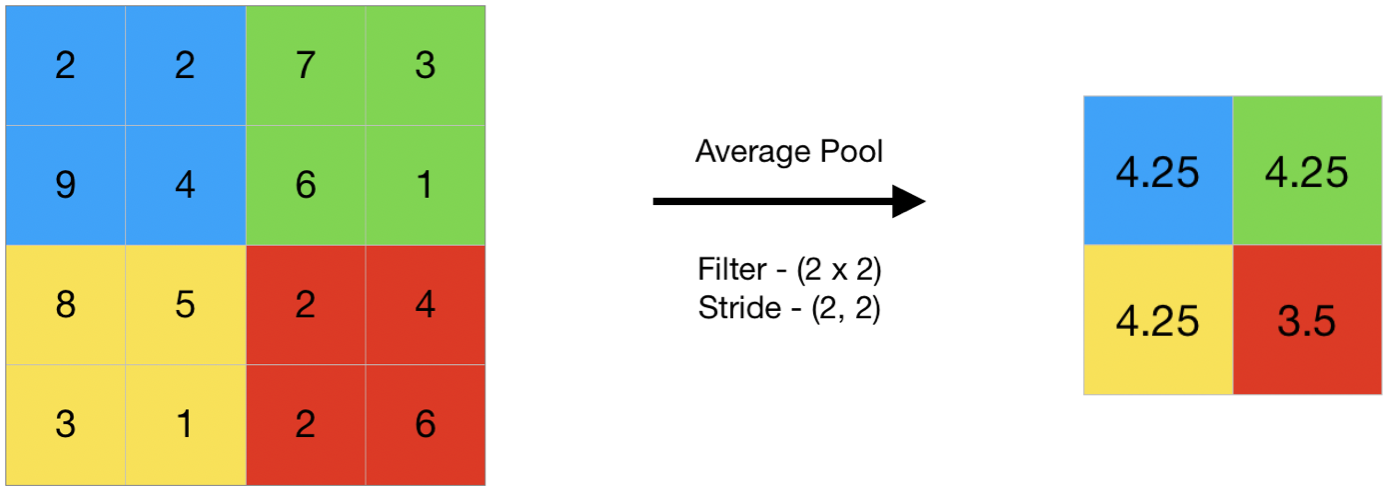


Figure (9.2)

**Min Pooling**

In min pooling, the layer operates with the most non-prominent feature of the feature map provided by the convolutional layer. More basically we can say it selects the minimum valued element from the region captured by the filter in any feature map.

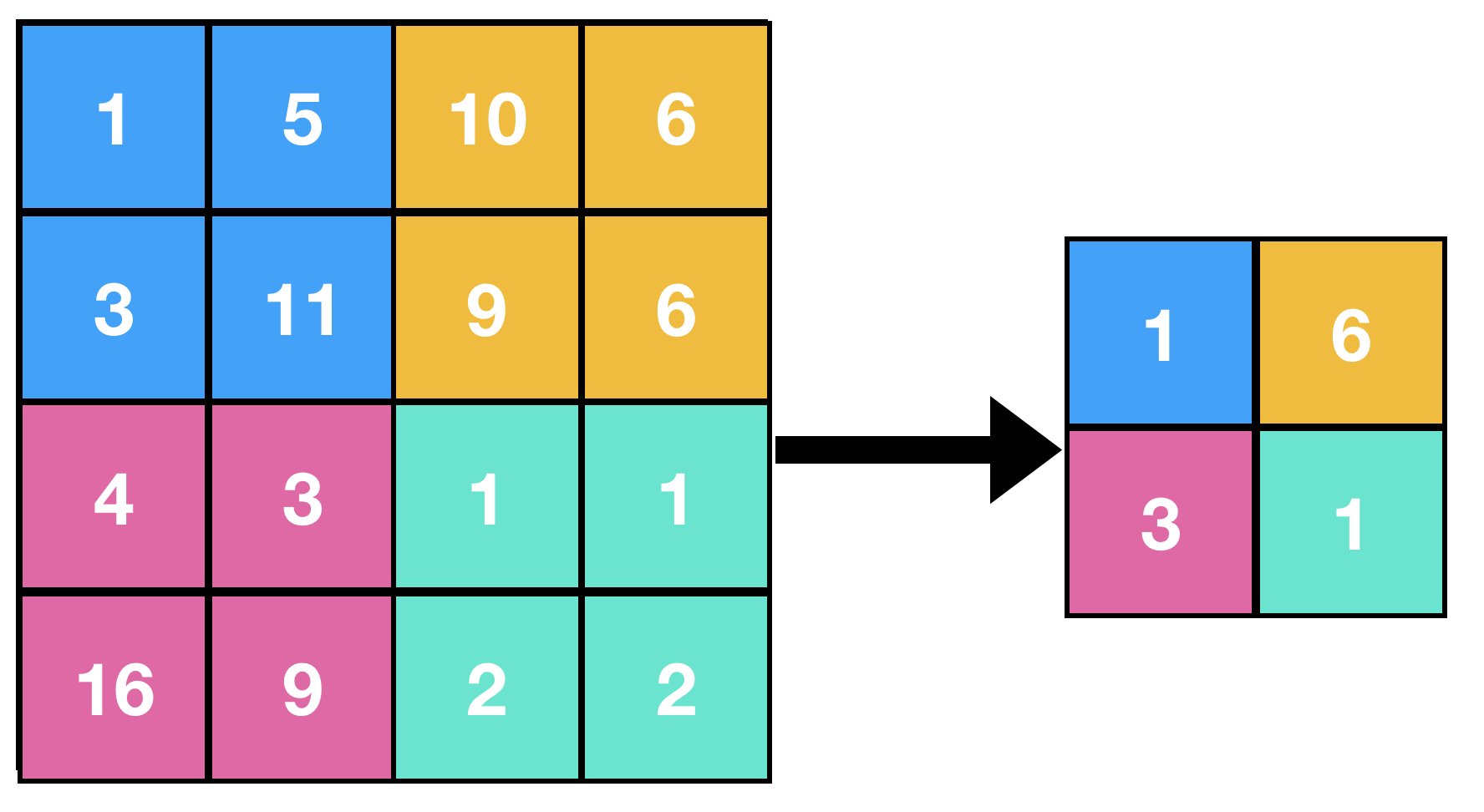


Figure (9.3)

**[3.2.6.] Activation function:**

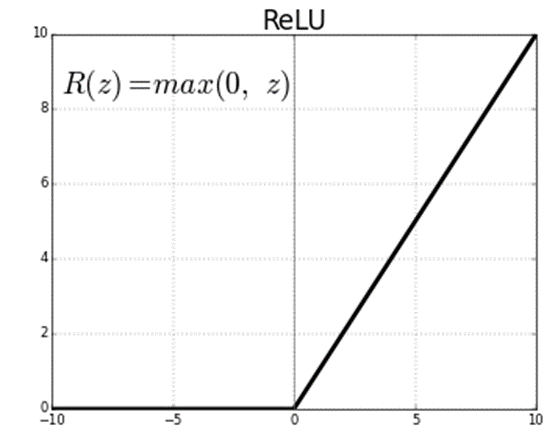
The activation functions we used here are relu and softmax functions. The activation function is a node that is put at the end of or in between Neural Networks.They help to decide if the neuron would fire or not. 

Figure (10)

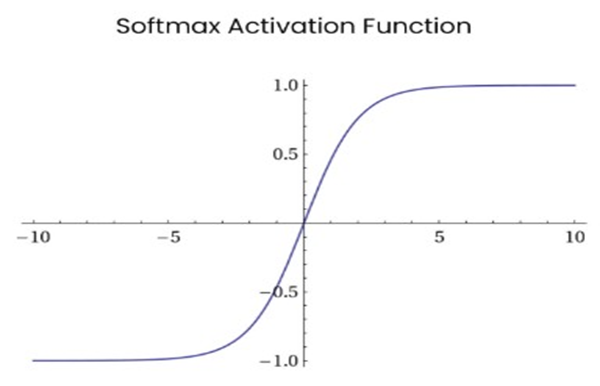


Figure (11)

**Rectified Linear Unit (ReLU):**

This function simply returns 0 if your value is negative else it returns the same value you gave, nothing but eliminates negative outputs and maintains values between 0 to +infinity.

The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time. This means that the neurons will only be deactivated if the output of the linear transformation is less than 0.

Now how does ReLU transform its input? It uses this simple formula:

f(x)=max(0,x)

The ReLU function is its derivative and both are monotonic. The function returns 0 if it receives any negative input, but for any positive value x, it returns that value back. Thus, it gives an output that has a range from 0 to infinity.

Let us define a ReLU function

def ReLU(x):

if x>0:

return x

else:

return 0

ReLU is used as a default activation function and nowadays it is the most commonly used activation function in neural networks, especially in CNNs. It consists of no heavy computation as there is no complicated math. The model can, therefore, take less time to train or run. One more important property that we consider the advantage of using ReLU activation function is sparsity. Usually, a matrix in which most entries are 0 is called a sparse matrix and similarly, we desire a property like this in our neural networks where some of the weights are zero. Sparsity results in concise models that often have better predictive power and less overfitting/noise. In a sparse network, it’s more likely that neurons are actually processing meaningful aspects of the problem.

**[3.2.7.] Flattening Layer**

Flattening and fully-connected layers are what we have at the last stage of CNN, which means you’re almost there. What are we doing? Image processing. For what? Classifying ‘the gestures.’ We are making a classification model, which means these processed data should be good input to the model. It needs to be in the form of a 1-dimensional linear vector. Rectangular or cubic shapes can’t be direct inputs. And this is why we need a flattening and fully connected layer.

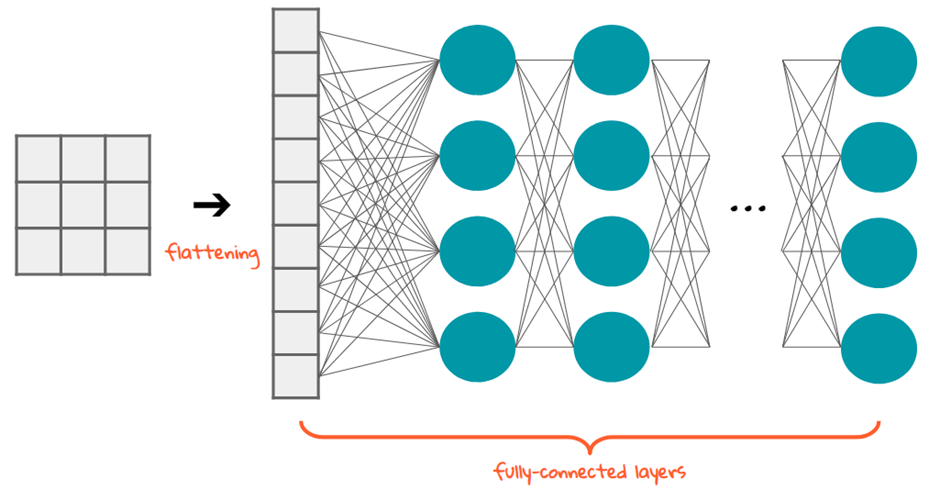


Figure (12)

Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the convolution layer to create a single lang feature network. And it is connected to the final classification model, which is called a fully connected layer.

So, we’ve got the pooled layer, pooled feature map. After we apply the convolution operation to our image and then we apply pooling to the results of the convolution which is the convolved image.

The flattening step is a refreshingly simple step involved in building a convolutional neural network. It involves taking the pooled feature map that is generated in the pooling step and transforming it into a one-dimensional vector. Here is a visual representation of what this process looks like:

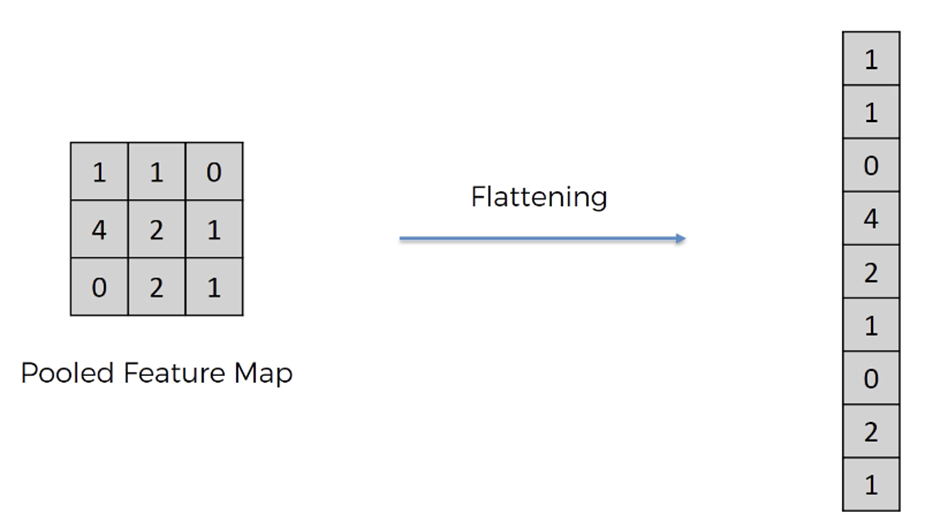


Figure (13)

The purpose is that we want to later input this into a neural network for further processing.

**[3.2.8.]** **Fully Connected Layer:**

Fully connected layers follow max pooling layer. In this layer, all the neurons of all layers are interconnected to the previous layer. The given input argument for this layer is 4, which indicates 4 classes. Fully Connected Layer is simply, feed forward neural network. Fully Connected Layers form the last few layers in the network. The input to the fully connected layer is the output from the *final* Pooling Convolutional Layer, which is *flattened* and then fed into the fully connected layer.

**[3.2.9] Softmax Activation Function:**

It is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output class. This function generates an output that ranges between values 0 and 1 and with the sum of the probabilities being equal to 1.

A CNN model which aims at classifying an image as either a Zoom-in, Zoom-out, Rotation, Information (4 possible outcomes/classes). The last (fully-connected) layer of the CNN outputs a vector of logits, L, that is passed through a Softmax layer that transforms the logits into probabilities, P. These probabilities are the model predictions for each of the 4 classes.

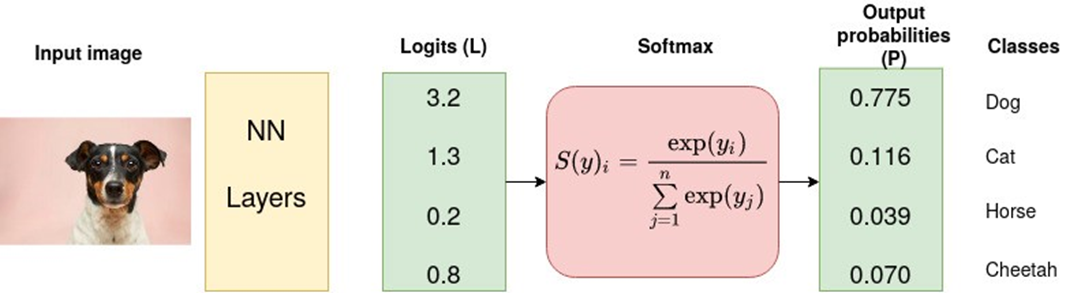


Figure (14)

**Summary**

**[3.3.] Block Diagram:**

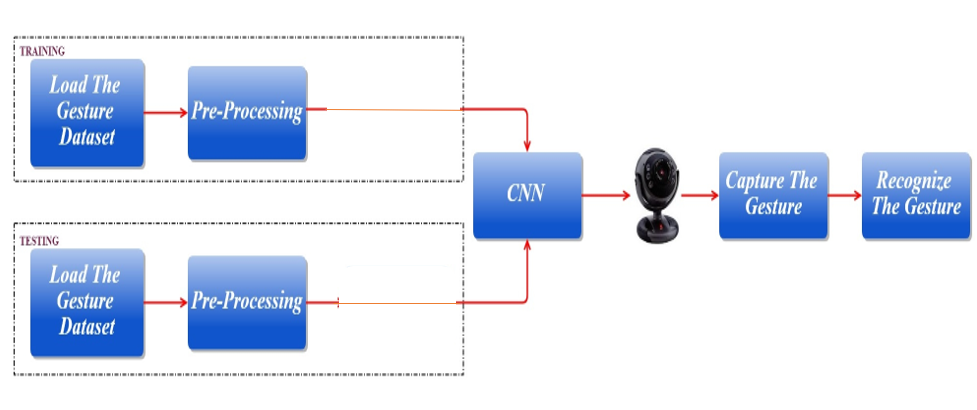
****

Figure (15)

**[3.4.1] Hand Gestures Collected:**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **ZOOM IN** | **ZOOM OUT** | **INFORMATION** | **ROTATION** |

Figure (16)

These are the hand gestures dataset collected. We have worked on these four gestures.

* The first gesture mentioned above is Zoom-in which will lessen the size of the image i.e. decrease the resolution of the image.
* The second gesture that is Zoom-out will zoom the image i.e. increase size of the image.
* The third gesture that is information will give the detailed information of the image, what is the image about. The id number of the image. The person's name of whom this image is related to.
* The last gesture i.e. forth is Rotation which will rotate the image according to your convenience that is clockwise or anticlockwise.

**[3.4.2.] Dataset after resizing the images to the same size:**

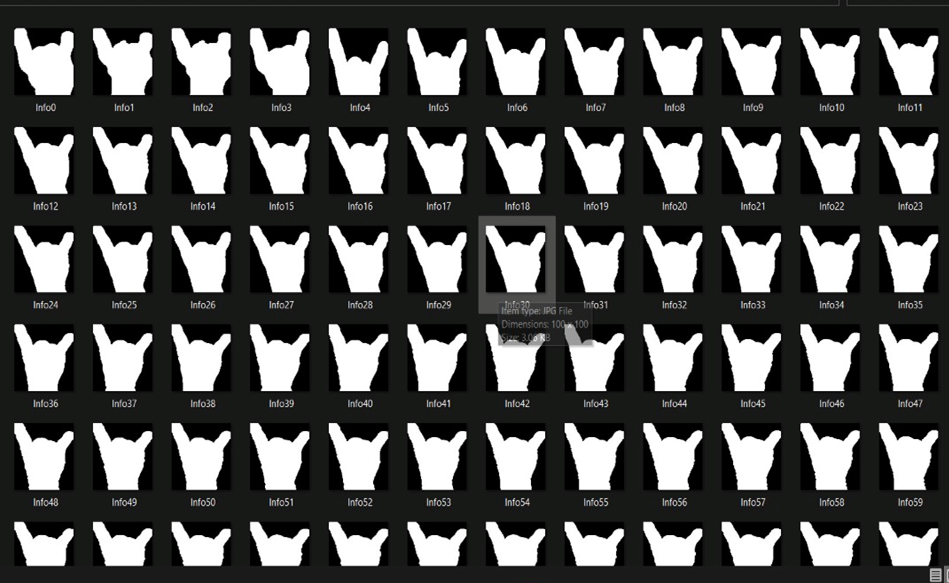


Figure (16.1)



Figure (16.2)



Figure (16.3)



Figure (16.4)

**[3.5.] Block diagram of the model:**

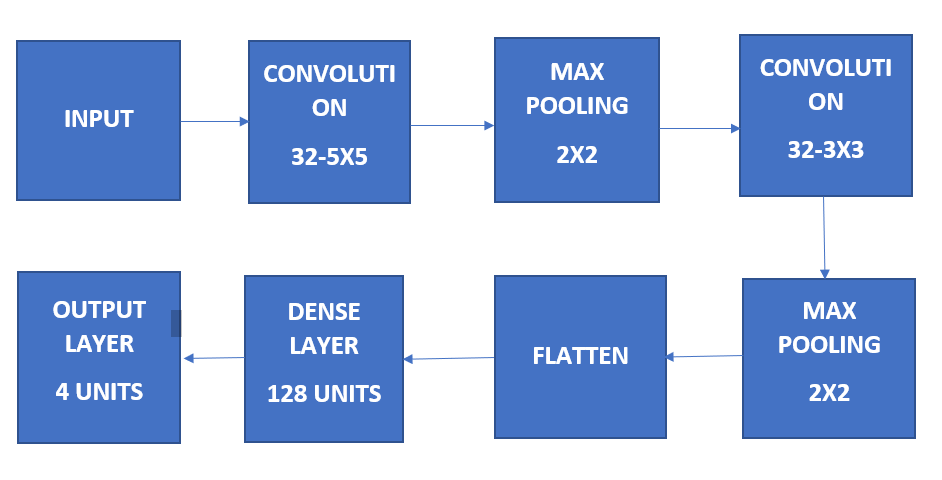


Figure (17)

* Initially the dataset is loaded into the network for training. Pre-processing is done before the feature is extracted. The training is done in Convolutional neural networks. After training an input image is given by capturing from a webcam. The given input image is tested for recognizing the gesture. A confusion matrix is produced according to the produced output with its mean accuracy.
* A ConvNet is a popular machine learning algorithm. Which is one of the techniques of Deep learning and is a learning model used to execute classification tasks through images. CNNs specifically give better results for identifying patterns in an image, which leads to recognition of hand gestures. The advantage of CNN is it doesn't require any feature extraction to train the model. CNN is invariant to the scaling and rotation.
* The first step is to gather the data. The data must be labeled. If the dataset is not labeled, this can be time consuming as you would have to manually create new labels for each category of images. Another method is to create new labels and only move pictures into their proper labels, and create a classifier like the one we will have and have that machine classify the images.
* The data set that is required is then pre-processed as per the requirement. The feature extraction takes place i.e., the main features which are shown in the dataset images are extracted for the classification to happen. Then these images are given as an input to CNN i.e., Convolutional Neural Network. And this Convolutional Neural Network has many functioning layers and each layer in it has its own specifications to perform.
* Training data set would contain 85–90% of the total labeled data. This data would be used to train our machine about the different types of images we have. Validation data set would contain 5–10% of the total labeled data. This will test how well our machine performs against known labeled data. This testing data will be used to test how well our machine can classify data it has never seen.
* The first convolution layer we introduce is of 32 filters and the filter size is 5X5 matrix as shown in the above block.
* The following layer is max pooling layer i.e. pulls the maximum values and the filter size used is 2X2. After this hidden layer there's an activation function named ReLu which decides which neurons in the layer should be activated.
* Here now we can insert ‘n’ number of hidden layers, so, there again is a convolution layer with 32 filters and the filter size is 3X3.
* Following this convolution layer there is a max pooling layer of filter size 2X2. Followed by the flattening layer where the data is converted into a 1-dimensional array for inputting it to the next layer. In the same way as the previous we use rule as the activation function in between the hidden layers.
* Then there is a dense layer with 128 units, i.e. these units mentioned are the number of neurons in the hidden layer. Then there is a final stage that is an output layer with 4 units. Here we have four units in the last layer because we have four classes to be classified at the final stage. At the last layer we use the softmax activation function. As per the multi class classification, the number of classes will decide the number of units in the outer layer.

**C H A P T E R 4**

Experimental Results and Discussions

**[4.1] The packages and libraries needed to install to efficiently work on the architecture of the code**

import os

import numpy as np

import scipy

import sklearn

import keras

from keras.models import Sequential

from keras.layers import Conv2D

from keras.layers import MaxPooling2D

from keras.layers import Flatten

from keras.layers import Dense

from tool.gesture\_recognition import MotionDetector

import cv2

from keras.preprocessing import image

from keras.preprocessing.image import ImageDataGenerator

from keras.models import model\_from\_json

import h5py

import matplotlib.pyplot as plt

import imutils

from imutils import paths

from PIL import Image

import keyboard

1. **import os**

The os module is a part of the standard library, or stdlib, within Python 3. This means that it comes with your Python installation, but you still must import it. The OS module in python provides functions for interacting with the operating system. This module provides a portable way of using operating system-dependent functionality.

1. **import numpy as np**

* NumPy is an open-source numerical Python library.
* NumPy contains a multi-dimensional array and matrix data structures.
* It can be utilized to perform a number of mathematical operations on arrays such as trigonometric, statistical, and algebraic routines. Therefore, the library contains a large number of mathematical, algebraic, and transformation functions.
* NumPy is an extension of Numeric and Numarray.
* Numpy also contains random number generators.
* NumPy is a wrapper around a library implemented in C.
* Pandas objects rely heavily on NumPy objects. Essentially, Pandas extends Numpy.

1. **import keras**

Keras is an Open Source Neural Network library written in Python that runs on top of Theano or Tensorflow. It is designed to be modular, fast and easy to use. It was developed by François Chollet, a Google engineer. Keras doesn’t handle low-level computation. Instead, it uses another library to do it, called the “Backend.

Keras is a high-level API wrapper for the low-level API, capable of running on top of TensorFlow, CNTK, or Theano. Keras High-Level API handles the way we make models, defining layers, or set up multiple input-output models. In this level, Keras also compiles our model with loss and optimizer functions, training process with fit function. Keras in Python doesn’t handle Low-Level API such as making the computational graph, making tensors or other variables because it has been handled by the “backend” engine.

Backend is a term in Keras that performs all low-level computation such as tensor products, convolutions and many other things with the help of other libraries such as Tensorflow or Theano. So, the “backend engine” will perform the computation and development of the models. Tensorflow is the default “backend engine” but we can change it in the configuration.

Tensorflow is the rising star in the deep learning framework. Developed by Google’s Brain team it is the most popular deep learning tool. With a lot of features, researchers contribute to help develop this framework for deep learning purposes.

1. **from keras.models import Sequential**

The core idea of Sequential API is simply arranging the Keras layers in a sequential order and so, it is called Sequential API. Most of the CNN also has layers in sequential order and the data flows from one layer to another layer in the given order until the data finally reaches the output layer.

A CNN model can be created by simply calling Sequential() API as specified below −

from keras.models import Sequential

model = Sequential()

1. **from keras.layers import Conv2D**

Keras Conv2D is a 2D Convolution Layer, this layer creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs.

Kernel: In image processing the kernel is a convolution matrix or masks which can be used for blurring, sharpening, embossing, edge detection, and more by doing a convolution between a kernel and an image.

**The Keras Conv2D class constructor has the following arguments:**

keras.layers.Conv2D(filters, kernel\_size, strides=(1, 1),

padding='valid', data\_format=None, dilation\_rate=(1, 1),

activation=None, use\_bias=True, kernel\_initializer='glorot\_uniform',

bias\_initializer='zeros', kernel\_regularizer=None,

bias\_regularizer=None, activity\_regularizer=None,

kernel\_constraint=None, bias\_constraint=None)

**filters**

* Mandatory Conv2D parameter is the number of filters that convolutional layers will learn from.
* It is an integer value and also determines the number of output filters in the convolution.

**kernel\_size**

* This parameter determines the dimensions of the kernel. Common dimensions include 1×1, 3×3, 5×5, and 7×7 which can be passed as (1, 1), (3, 3), (5, 5), or (7, 7) tuples.
* It is an integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window.

**strides**

* This parameter is an integer or tuple/list of 2 integers, specifying the “step” of the convolution along with the height and width of the input volume.
* Its default value is always set to (1, 1) which means that the given Conv2D filter is applied to the current location of the input volume and the given filter takes a 1-pixel step to the right and again the filter is applied to the input volume and it is performed until we reach the far right border of the volume in which we are moving our filter.

**padding**

* The padding parameter of the Keras Conv2D class can take one of two values: ‘valid’ or ‘same’.
* Setting the value to “valid” parameter means that the input volume is not zero-padded and the spatial dimensions are allowed to reduce via the natural application of convolution.

**data\_format**

* This parameter of the Conv2D class can be either set to “channels\_last” or “channels\_first” value.
* The TensorFlow backend to Keras uses channels last ordering whereas the Theano backend uses channels first ordering.
* Usually we are not going to touch this value as Keras as most of the times we will be using TensorFlow backend to Keras.
* It defaults to the image\_data\_format value found in your Keras config file at ~/.keras/Keras.json.

**activation**

* The activation parameter to the Conv2D class is simply a convenience parameter which allows you to supply a string, which specifies the name of the activation function you want to apply after performing the convolution.

**use\_bias**

* This parameter of the Conv2D class is used to determine whether a bias vector will be added to the convolutional layer.
* By default, its value is set as True.

**kernel\_initializer**

* This parameter controls the initialization method which is used to initialize all the values in the Conv2D class before actually training the model.
* It is the initializer for the kernel weights matrix.

**bias\_initializer**

* Whereas the bias\_initializer controls how the bias vector is actually initialized before the training starts.
* It is the initializer for the bias vector.

1. **from keras.layers import MaxPooling2D**

Max pooling operation for 2D spatial data. Downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window (of size defined by pool\_size) for each channel of the input. The window is shifted by strides along each dimension

1. **from keras.layers import Flatten**

This is where Keras flatten comes to save us. This function converts the multi-dimensional arrays into flattened one-dimensional arrays or single-dimensional arrays. It takes all the elements in the original tensor (multi-dimensional array) and puts them into a single-dimensional array. not that this does not include the batch dimension.

1. **from keras.layers import Dense**

The dense layer is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer. The dense layer is found to be the most commonly used layer in the models.

In the background, the dense layer performs a matrix-vector multiplication. The values used in the matrix are actually parameters that can be trained and updated with the help of backpropagation.

The output generated by the dense layer is an ‘m’ dimensional vector. Thus, a dense layer is basically used for changing the dimensions of the vector. Dense layers also apply operations like rotation, scaling, translation on the vector.

### **Syntax**

**keras.layers.Dense(units, activation=None, use\_bias=True**

1. **from keras.preprocessing import image**

Keras Preprocessing is the data preprocessing and data augmentation module of the Keras deep learning library. It provides utilities for working with image data, text data, and sequence data.

1. **from keras.preprocessing.image import ImageDataGenerator**

Keras ImageDataGenerator class provides a quick and easy way to augment your images. It provides a host of different augmentation techniques like standardization, rotation, shifts, flips, brightness change, and many more.

#### **1. Random Rotations**

Image rotation is one of the widely used augmentation techniques and allows the model to become invariant to the orientation of the object.

ImageDataGenerator class allows you to randomly rotate images through any degree between 0 and 360 by providing an integer value in the rotation\_range argument.

#### **2. Random Zoom**

The zoom augmentation either randomly zooms in on the image or zooms out of the image. ImageDataGenerator class takes in a float value for zooming in the zoom\_range argument. You could provide a list with two values specifying the lower and the upper limit. Else, if you specify a float value, then zoom will be done in the range [1-zoom\_range,1+zoom\_range]. Any value smaller than 1 will zoom in on the image. Whereas any value greater than 1 will zoom out on the image.

#### **3. Random Flips**

Flipping images is also a great augmentation technique and it makes sense to use it with a lot of different objects. ImageDataGenerator class has parameters horizontal\_flip and vertical\_flip for flipping along the vertical or the horizontal axis. However, this technique should be according to the object in the image.

1. **from keras.models import model\_from\_json**

* to\_json() − Returns the model as a json object.
* model\_from\_json() − Accepts json representation of the model and creates a new model.

**Python JSON**

JSON stands for JavaScript Object Notation. JSON is a lightweight data format used for data interchange between multiple different languages. It is easy to read for humans and easily parsed by machines.

It’s pretty common for websites to return JSON from API so that the information is easy to parse by different programming languages. In Python, the text of JSON is read as a quoted-string which contains the value in key-value mapping within { }. Once parsed, it is available as a dictionary object in Python.

Python comes with a built-in package called json for encoding and decoding JSON data.

For working with json type files, first you need to import the json library.

import json

1. **from imutils import paths**

imutils is a Python basic image processing functional package to do image translation, rotation, resizing, skeletonization, or blur amount detection. imutils also check to find functions if you already have NumPy, SciPy, Matplotlib, and OpenCV installed.

1. **from PIL import Image**

To load the image, we simply import the image module from the pillow and call the Image.open(), passing the image filename.

Instead of calling the Pillow module, we will call the PIL module to make it backward compatible with an older module called Python Imaging Library (PIL). That’s why our code starts with “from PIL import Image” instead of “from Pillow import Image”.

1. **import keyboard**

Python provides a library named keyboard which is used to get full control of the keyboard. It’s a small Python library which can hook global events, register hotkeys, simulate key presses and much more.

* It helps to enter keys, record the keyboard activities and block the keys until a specified key is entered and simulate the keys.
* It captures all keys, even onscreen keyboard events are also captured.
* Keyboard module supports complex hotkeys.
* Using this module we can listen and send keyboard events.

**[4.2.] Resultant summary of the classifier**

|  |  |  |
| --- | --- | --- |
| Layer (Type) | Output shape | #Para |
| Conv2d\_1 (conv2D) | (None,96,96,32) | 2432 |
| max\_pooling2d\_1(MaxPooling 2 | (None,48,48,32) | 0 |
| Conv2d\_2 (conv2D) | (None,46,46,32) | 9248 |
| max\_pooling2d\_2(MaxPooling 2 | (None,23,23,32) | 0 |
| Flatten\_1 (flatten) | (None, 16928) | 0 |
| Dense\_1 (Dense) | (None, 128) | 2166912 |
| Dense\_2 (Dense) | (None, 4) | 516 |

Table (1)

Total params: 2,179,108

Trainable params: 2,179,108

Non-trainable params: 0

* The first column consists of types of layers and the second column consists of output shape after the completion of processing on the hidden layers.
* In row 1 the initialization of the network is with the convolution layer i.e. conv2D\_1. Here, (Convolution2D(32, 3, 3, input\_shape = (64, 64, 3), activation = 'relu')), Convolution2D(32, 3, 3, input\_shape = (64, 64, 3), activation = 'relu')) The first parameter tells us about the number of filters used in our convolution operation. Then the second parameter specifies the size of the convolutional filter in pixels. Filter size may be determined by the CNN architecture you are using – for example, VGGNet exclusively uses (3, 3) filters. If not, use a 5×5 or 7×7 filter to learn larger features and then quickly reduce to 3×3. The Third parameter specifies how the convolutional filter should step along the x-axis and the y-axis of the source image. In most cases, it’s okay to leave the strides parameter with the default (1, 1). However, you may increase it to (2, 2) to reduce the size of the output volume. The Fourth parameter is the activation parameter which specifies the name of the activation function you want to apply after performing convolution.

We get the output shape of (96,96) having 32 as the number of filters and None mentioned because batch size is not initialized.

* classifier.add(MaxPooling2D(pool\_size = (2, 2)))

The second row shows the layer type as MaxPooling layer where we take a 2x2 matrix we’ll have minimum pixel loss and get a precise region where the feature are located., then the output shape we get is (None,48,48,32)

* classifier.add(Conv2D(32, (3, 3))

Here the convolution layer has 32 filters and the size of filters is 3X3. The output shape then we get is (46,46,32) and the output parameter we get is 9248.

* classifier.add(MaxPooling2D(pool\_size = (2, 2)))

Again, the next parameter is related to MaxPooling2D where its filter size is 2X2. The output shape we get is(23,23,32) with output parameter zero.

* Flattening transforms a two-dimensional matrix of features into a vector of features. Here the output shape dimension of the flattening layer is (None, 16928). In this step we need to create a fully connected layer, and to this layer we are going to connect the set of nodes we got after the flattening step, these nodes will act as an input layer to these fully-connected layers. As this layer will be present between the input layer and output layer, we can refer to it as a hidden layer.
* classifier.add(Dense(units = 128, activation = 'relu'))

Dense is the function to add a fully connected layer, ‘units’ is where we define the number of nodes that should be present in this hidden layer, these units value will be always between the number of input nodes and the output nodes but the art of choosing the most optimal number of nodes can be achieved only through experimental tries. Though it’s a common practice to use a power of 2. And the activation function will be a rectifier function.

* classifier.add(Dense(units = 4, activation = 'softmax'))

It’s time to initialize our output layer, which should contain four nodes, as it is multi classification. The final layer contains only four nodes, and we will be using a softmax activation function for the final layer.

**[4.3.] Model Accuray:**

Training set images found 3350 images belonging to 4 classes.

Testing set images found 991 images belonging to 4 classes.

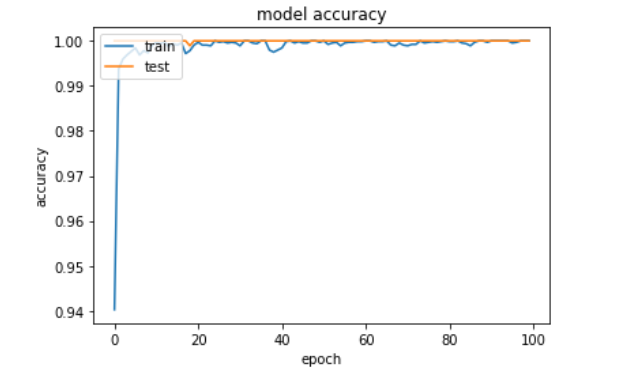
Number of epochs: 100

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Epoch | Time (ms/step) | loss | accuracy | val\_loss | val\_acc |
| 1 | 681ms | 0.1601 | 0.9404 | 5.92e-05 | 1.0000 |
| 2 | 444ms | 0.0218 | 0.9937 | 2.866e-04 | 1.0000 |
| 3 | 412ms | 0.0121 | 0.9959 | 1.0354e-04 | 1.0000 |
| 4  .  .  . | 419ms  .  .  . | 0.0085  .  .  . | 0.9969  .  .  . | 8.0297e-06  .  .  . | 1.0000  .  . |

Table (2)

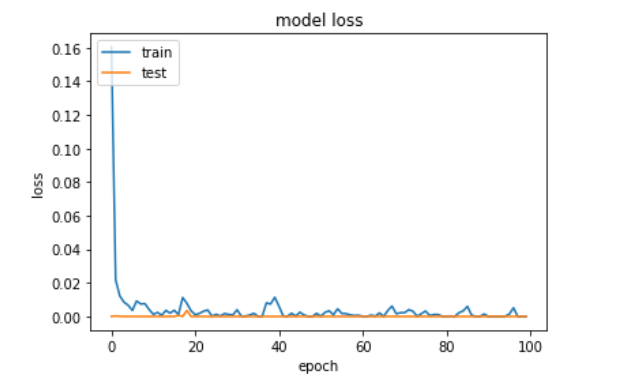
As shown in the above table the number of epochs is set to 100. As the number of the epoch goes on increasing the time required per step of epoch decreases. And the model loss value goes on decreasing from 0.1 to 0.0001. The accuracy increases from 94% to 99.8% i.e. almost 100%. The validation loss also decreases and the validation accuracy is almost constant, that is 1.

**[4.4.] History for accuracy**

****

Graph (1.1)

In the above graph (1.1), The plot of the graph is epoch vs accuracy. On the x-axis we have epoch and, on the y-axis, we have accuracy. The blue line shows the training accuracy and the orange line shows the testing accuracy. Total number of epochs we have is 100. As the number of steps of epoch proceeds, the accuracy increases. As shown above in the graph that means the accuracy is almost 99-100%.



Graph (1.2)

In the above graph (1.2), The plot of the graph is epoch vs loss. On the x- axis we have epoch and on the y- axis we have loss, the orange line shows the testing loss and the blue line shows the training loss.

As shown above, as the steps of epoch increase the loss value is decreasing, we don't have loss greater than 0.01. That means the accuracy is almost 100% and the loss is also nearly negligible. Hence, the network trained shows the proper results with high accuracy.

**C H A P T E R 5**

Conclusion & Future Scope

**[5.1.] Conclusion**

Nowadays, applications need several kinds of image patterns as source input of information for analyzing. Serval features of input are to be extracted to perform various applications or tasks. When transformed from one form to another such as digitizing, scanning, communicating and storing etc. degradation occurs. Therefore, the output has to undertake a process called image enhancement which contains a group of methods that seek to develop the visual presence of an image.

Variable features of gestures are extracted for different input gestures, which indicate the actual nature of the code. The extracted features are enough to represent the given task.

In the project, we consider a vision-based system that can interpret a user’s gesture in real time to manipulate windows & objects within a medical data visualization. A hand segmentation procedure first captures the picture of gesture, extracts the feature. Dynamic navigation gestures are translated to commands based on their relative position on the screen. Static gesture poses are identified to execute non-directional commands. This is accomplished by using open palm, feast, victory symbol like features to represent the zoom in, zoom out, rotation, image information. These features are then input to the classification.

The hand region is detected from the background by the background subtraction method. Then, the palm and finger are segmented. On the basis of segmentation, the fingers in the hand image are discovered and recognized. The recognition of hand gestures is accomplished by a simple CNN classifier. The performance of our method is evaluated on a data set of hand images. The experimental results show that our approach performs well & is fit for the real-time applications.

The performance of the proposal method highly depends on the result of hand detection. If there are moving objects with the color similar to that of the skin, the object exits in the result of hand detection & then degrades the performance of the hand gesture recognition. The utilization of gesture recognition technology in the medical environment has been progressing. This project is about the case study on touchless communication between humans & computers to avoid the risk of contamination of germs during surgery.

**[5.2.] Future Scope:**

Hand Gesture recognition is moving at tremendous speed for the futuristic products and services and major companies are developing technology based on hand gesture systems that includes companies like Microsoft, Samsung, Sony and it includes the devices lik e Laptop, Hand held devices, Professional and LED lights. The verticals include where the Gesture technology is and will be evident are Entertainment, Artificial Intelligence, Education and Medical and Automation fields. And with a lot of Research and Development in the field of Gesture Recognition Field, the use and adoption will become more cost effective and cheaper. It’s a brilliant feature turning data into features with a mix of technology and Human wave. Smart phones have been experiencing an enormous amount of Gesture Recognition Technology with look and views and working to manage the Smartphone in reading, viewing and that includes what we call touch less gestures. Google Glass has been embedded into smart televisions nowadays as well, which can easily be controlled and managed by Voice and Hand options. In the medical fields Hand Gestures may also be experienced in terms of Robotic Nurse and medical assistance. As the Technology is always revolving and changing the future is quite unpredictable but we have to be certain the future of Gesture Recognition is here to stay with more and more eventful and Life touching experiences.

Additional future work will include recognition of dynamic two-handed manipulation gestures for zooming an image, rotating an image etc. We are interested as well as, to experiment with large gesture vocabularies to enhance the interaction flexibility to the system.

The proposed hand gesture recognition system used to recognize the tasks given to computers for processing the image can be extended to recognize gestures and facial expressions.

**References**

[1] Poluka Srilatha, Tiruveedhula Saranya, Advancements in Gesture Recognition Technology, IOSR Journal of VLSI and Signal Processing (IOSR-JVSP), 4(1), 2014, 2319-4197

[2] Minoo Hamissi , Karim Faez, Real-Time Hand Gesture Recognition Based on the Depth Map for Human Robot Interaction, International Journal of Electrical and Computer Engineering (IJECE) , 3(6),2013, 2088-8708

[3] Prof Kamal K Vyas, Amita Pareek, Dr Sandhya Vyas, Gesture Recognition and Control, International Journal on Recent and Innovation Trends in Computing and Communication, 1(7), 2321 –8169

[4] Hervé Lahamy \* and Derek Litchi, “Real Time Hand Gesture Recognition using Range Cameras”.

[5] Monuri Hemantha, M.V.Srikant, “Simulation of Real Time Hand Gesture recognition for Physically Impaired”, Proc. of the International Journal of Advanced Research in Computer and Comm Engineering Vol. 2, Issue 11, November’13.

[6] Joyeeta Singha, Karen Das, “Hand Gesture Recognition Based on Karhunen-Loeve Transform”.

[7] Tin Hninn Hninn Maung ,”Real-Time Hand Tracking and Gesture Recognition System Using Neural Networks”, World Academy of Science, Engineering and Technology 50 2009

[8] Pavlovic, et al. visual interpretation of hand gesture for human computer interaction: a review, IEEE Trans. On Pattern anal. Mach. Intel. 19(7), pp 677 695, 1997.

[9] X. Zabulis, H. Baltzakis, A. Argyrosz, “Vision-based Hand Gesture Recognition for Human-Computer Interaction”, Proc. Institute of Computer Science Foundation for Research and Technology - Hellas (FORTH) Heraklion, Crete, Greece

[10] E.Stergiopoulou and N.Papamarkos: “A New Technique on Hand Gesture Recognition”, Proc of the IEEE International Conference on Image Processing, 2657-2660, 2006.

[11] Arpita Ray Sarkar, G. Sanyal & S. Majumder: “Hand Gesture Recognition Systems: A Survey”, Proc of the International Journal of Computer Applications (0975 – 8887) Volume 71– No.15, May 2013 Shweta. K. Yewale & Pankaj. K. Bharne: “Hand Gesture Recognition Using Different Algorithms Based on Artificial Neural Network”

[12] Feng-Sheng Chen, Chih-Ming Fu, Chung-Lin Huang : “Hand gesture recognition using a real-time tracking method and hidden Markov model”

[13] Byung-Woo Min, Ho-Sub Yoon, Jung Soh, Yun-Mo Yangc and Toshiaki Ejima : “Hand Gesture Recognition Using Hidden Markov Models”, Proc. of the IEEE International conference on Systems, Man and Cybernetics, vol 5, pp. 4232 -4235, 1997.

[14] Yikai Fang, Jian Cheng, Kongqiao Wang and Hanqing Lu, “Hand Gesture Recognition Using Fast Multi-scale Analysis”, Proc. of the Fourth International Conference on Image and Graphics, pp 694-698, 2007.

[15] Chris Joslin, Ayman El-Sawah, Qing chen, Nicolas Georganas, “ Dynamic Gesture Recognition”, Proc. of the Instrumental and Measurement Technology Conference, pp 1706-1710, 2005.

[16] Prateem Chakraborty, Prashant Sarawgi, Ankit Mehrotra, Gaurav Agarwal, Ratika Pradhan, “Hand Gesture Recognition: A Comparative Study”, Proceedings of the International MultiConference of Engineers and Computer Scientists 2008 Vol I IMECS 2008, 19-21 March, 2008, Hong Kong

[17] Aditya Ramamoorthy et al. “Recognition of dynamic hand gestures “, page 1-13. Department of Electrical Engineering IIT New Delhi-110016 India. October 2002.