Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

# HAND GESTURE RECOGNITION USING CNN

# FINAL REPORT

| TEAM MEMBERS | |
|---|---|
| 18BEC0921 | UDUMULA ROHIT REDDY |
| 18BEC0972 | DWARSALA SUJITH KUMAR REDDY |

**Subject code & name:**

NEURAL NETWORKS AND FUZZYCONTROL
[ECE3009]

**Semester:** Winter Semester

**Slot**: G2+TG2

## CERTIFICATE

This is to certify that the project work entitled "Hand Gesture Recognition Using CNN". That is being submitted by a group of two for NEURAL NETWORKS AND FUZZY CONTROL [ECE3009] is a Record of Bonafede work done under Professor VENUGOPAL P supervision. The Contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted for any other CAL course.

Place: Vellore

Date: 7th JUNE , 2021

## ACKNOWLEDGEMENT

We would like to thank our professor VENUGOPAL P whom we are highly indebted to for giving us this opportunity to performthis project and for his help, and guidance throughout this project. We would like to express our gratitude to VIT University and the School of Electronics and Communication Engineering (SENSE). We would like to extend our gratitude to the Dean of SENSE and this prestigious University in supporting us and giving us an opportunity to carry out our studies at this University. Last but not the least; we would like to thank our Parents and my friends for supporting and helping us throughout our project

# Hand Gesture Recognition using CNN

## Abstract:

Communicating through hand gestures is a non-verbal way of communication, which is most necessary for disabled persons. This type of communication is very helpful for blind ,deaf people. Communication is sharing and conveying of information. We focus on detecting and recognizing the hand gestures. In this project is to create a model or algorithm capable of classifying 10 categories of images of different hand gestures such as a showing fist, palm, showing the thumb up, thumb down , ok, v, index, c, using Machine Learning real life this gesture recognition can be used to control or interact with devices without physically touching them the identification and recognition of posture and human behavior is also the subject of gesture recognition techniques. By using concept of gestures recognition it is possible to point a finger at that point and it will move accordingly and which could make conventional input on devices such that and even redundant. In this project, we attempt to design a neural network using back propagation algorithm. This project is to get a better and a deep understanding of the working of neural networks and how to apply these networks in machine learning in form of algorithms which make our life easy by having a variety of real-life applications. In this project, we attempt to use a Neural Network using CNN algorithms where we input dataset images of different hand gestures as input arrays and the Neural Network then processes them, recognizes them and gives output of which hand gesture was detected correctly.

## Introduction:

Communication is imparting, sharing and passing on of data, news, thoughts and emotions. Of them, gesture based communication is one of the method of non-verbal correspondence which is picking up catalyst and solid traction because of its applications in an enormous number of fields. The most noticeable utilization of this strategy is its use by distinctively handicapped people like hard of hearing and quiet individuals. They can speak with non-marking individuals without the assistance of an interpreter or mediator by this strategy. Some different applications are in the car area, travel area, gaming area and furthermore while opening a PDA. The sign motion acknowledgment should be possible in two different ways: static motion and dynamic motion. While conveying, the static signal utilizes hand shapes while the dynamic motion utilizes the developments of the hand. Hand motion acknowledgment is a method of understanding and afterward grouping the developments by the hands. However, the human hands have extremely complex explanations with the human body and subsequently a great deal of blunders can emerge. Hence it is hard to perceive the hand signals. As of now a great deal of strategies and advances are being utilized for sign and motion acknowledgment. Among them the most well-known ones utilized are Hand Glove Based Analysis, Microsoft Kinect Based Analysis, Support Vector Machines and Convolutional Neural Networks. One of the goal of these strategies is to connect the correspondence hole among discourse and hearing weakened individuals with the ordinary individuals and furthermore effective and smooth joining of these distinctively abled individuals in our general public. In this project we manufacture a constant correspondence framework utilizing the progressions in Machine Learning. Right now the frameworks in presence either take a shot at a little dataset and accomplish stable exactness or work on an enormous dataset with temperamental precision. We attempt to determine this issue by applying Convolutional Neural Network (CNN) on a genuinely huge dataset to accomplish a decent and stable precision

## Proposed Method:

In the perspective on the restrictions presented by the methodologies referenced, our framework would zero in on alleviating those ineptitudes. The framework to be assembled must be skilled to have the option to be conveyed on a portable or web application for far reach and simple availability thus, it must be lightweight and computationally able enough to perceive the signs fittingly. We propose a PC vision-based way to deal with perceive static hand motions. The framework would break down a video take care of and perceive the hand motion and afterward yield the right class mark. For each sign performed by the client, the framework will yield 1 among 10 gestures. The framework would take in a video feed which could be pre-recorded or coming live from an information gadget The framework would isolate each sign and yield the sign name in like manner. The significant difficulties distinguished were performing exact foundation deduction and accomplishing high precision all together for the framework to be utilized in figuring sentences. Foundation Subtraction required handling changing enlightenment and forefront commotion in input pictures. Activities, for example, Gaussian Mixture based division and Image Morphology are acted to diminish foundation clamor. From this time forward, our framework engineering utilizes three stages viz. Casing isolation, Image Processing, and Image acknowledgment.

## Methodology:

First we import all the libraries needed like:

- Tensorflow – Contains modules used for classification, perception and prediction
- Matplotlib – contains commands used to display the images

- Sklearn – contains commands used to manipulate or organize the datasets
- Cv2 – contains commands used to do operations on images
- Pandas – contains commands used to import datasets

## Data pre-processing:

- Then we give collab access to the google drive folder where the dataset is present
- Then we build a function for debugging and printing images, then we test it
- Once that is done we convert our dataset to an array with then converted into a NumPyarray with 8 bit width each for faster processing. Simulatanesly assigning labels for the images

## Constructing the model:

CNN's contain 3 components
convolution layers – To find out the significant part of the image (activation -RELU)

- Pooling layers – used to decrease size of the image to decrease processing time (Activation - RELU
- Dense layers – performs the classification from the previous layers (fully connected)
- Flatten layer- between the convolutional and the fully connected layers there will be a Flatten layer. Flattening layer transforms a two-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier.
- Dropout layer-A layer which prevents from overfitting Arranging 12 layers of our CNN as following:
- Convolution using ReLu
- Maxpooling layer
- Convolution layer using Relu
- Maxpooling layer
- Convolution using ReLu
- Maxpooling layer
- Flatten layer

- Dropout layer
- Dense layer using ReLu

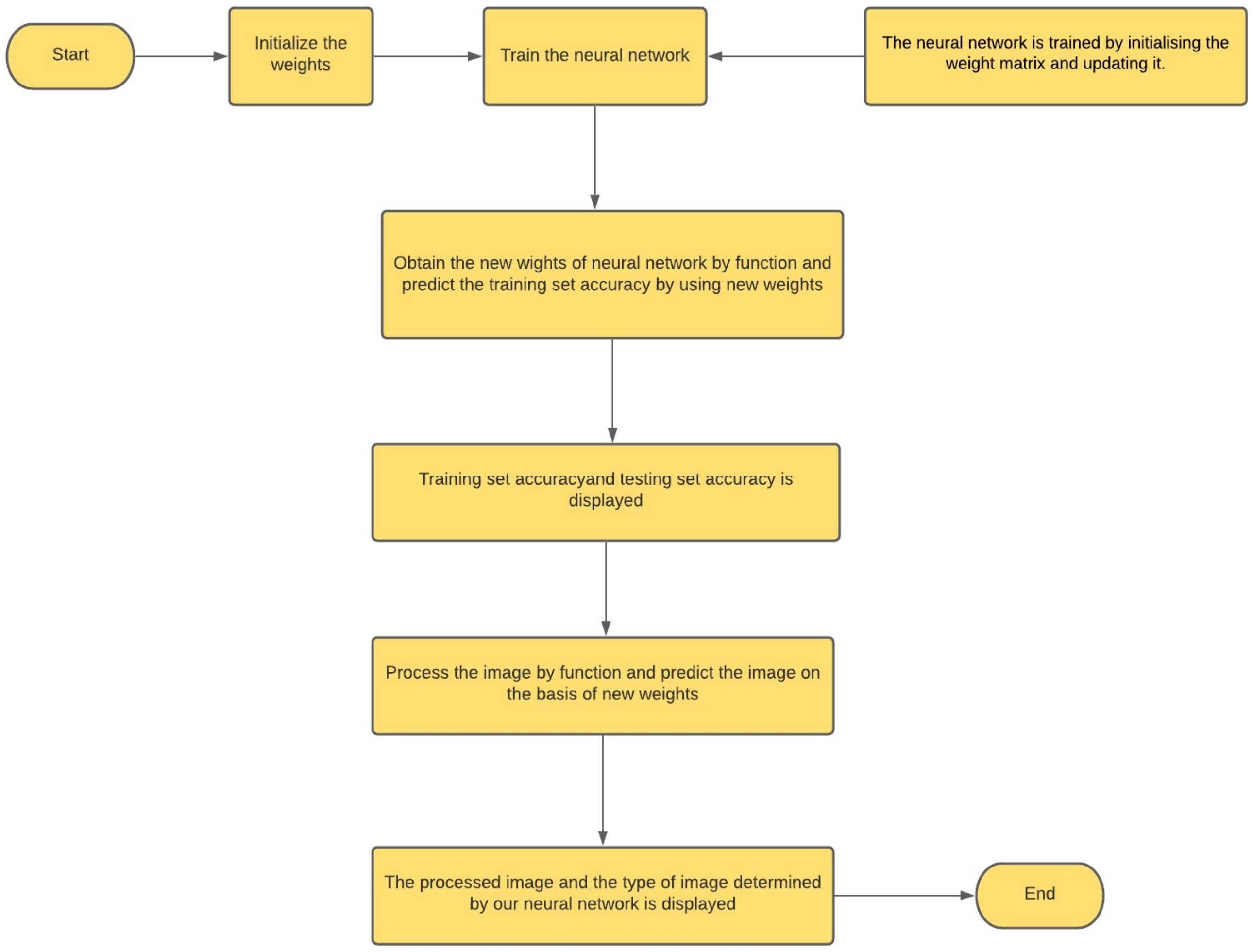
**Training:**

- First we configure the model to optimize and sparse categorical cross-entropy for calculatingloss  and only metric we want to know is accuracy
- Then we fit the dataset to the neural network and start training
- Once the training is done, we save that
  to h5 file

## Testing

- Using model.accuracy we get loss and accuracy
- Using.prediction we get the prediction and xtest prediction is equal to ytest then we know theneural network predicted correctly.
- The we prepare a function to plot images and for validation purposes
- Then print the evaluated data in a table format using pandas

## Block Diagram:

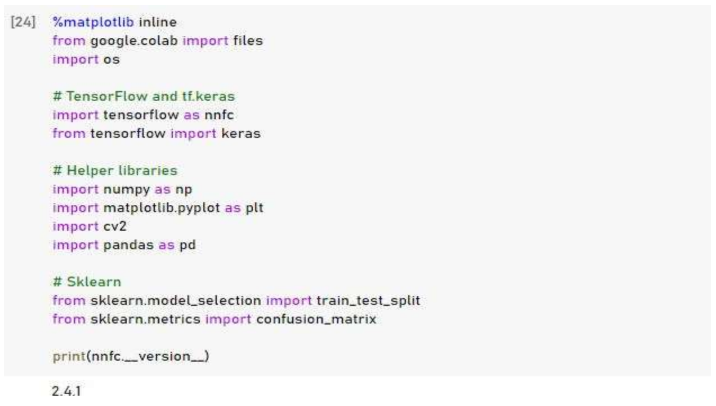

Start
Initialize the weights
Train the neural network
The neural network is trained by initialising the weight matrix and updating it.
Obtain the new wights of neural network by function and predict the training set accuracy by using new weights
Training set accuracyand testing set accuracy is displayed
Process the image by function and predict the image on the basis of new weights
The processed image and the type of image determined by our neural network is displayed
End

## Image Recognition:

In this stage framework is the picture acknowledgment stage. So as to accomplish higher precision when contrasted with existing frameworks and to keep the framework computationally lightweight, we utilize Convolutional neural organization (CNN) for picture acknowledgment. Convolutional neural organization are a class of feed-forward neural organizations normally utilized for visual examination assignments. They include neurons which go about as learnable boundaries having their own weight and predispositions. The whole neural organization learns with the assistance of a misfortune work, a learning rate is utilized to adjust the learning. The information layer takes in the information which gets spread through the different layers and a yield is created, the produced yield is contrasted and the real yield and the framework refreshes its loads and inclinations to address itself, this progression is critical and is known as backpropagation and this cycle done iteratively is called as preparing. The dataset comprised of 10 different categories of hand gestures. The proposed engineering of our CNN is represented in figure. It comprises of three convolutional layers with different number of channels, having halfway max-pooling layers and RELU initiations. The last three layers comprised of a leveling layer and completely associated layers with dropout layers in the middle of so as to maintain a strategic distance from overfitting. The last thick layer is of size relating to the quantity of class marks with SoftMax initiation. The contribution to the CNN will be the dim scale prepared picture resized to according to the dataset and the yield of the CNN would be a likelihood appropriation. For any picture feed into the CNN, it yields a likelihood dissemination. The hub containing the most elevated likelihood esteem is considered as the yield hub and the right mark against that hub is yielded. In this manner the framework figures out what sign the client performed.

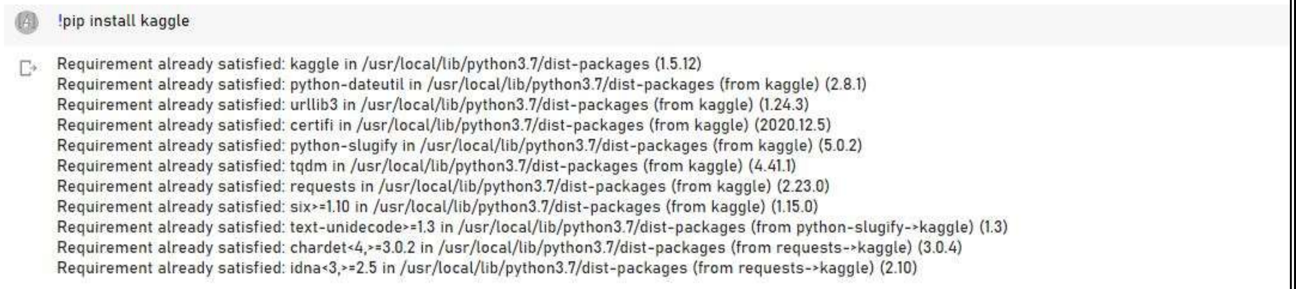
## Code:

### Importing libraries



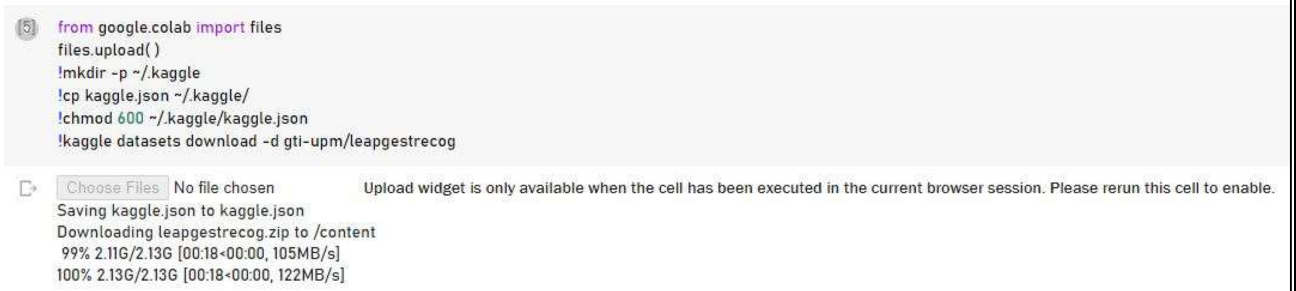
```
[24] %matplotlib inline
     from google.colab import files
     import os

     # TensorFlow and tf.keras
     import tensorflow as nnfc
     from tensorflow import keras

     # Helper libraries
     import numpy as np
     import matplotlib.pyplot as plt
     import cv2
     import pandas as pd

     # Sklearn
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import confusion_matrix

     print(nnfc.__version__)

     2.4.1
```

## Dataset from Kaggle



```
!pip install kaggle
```

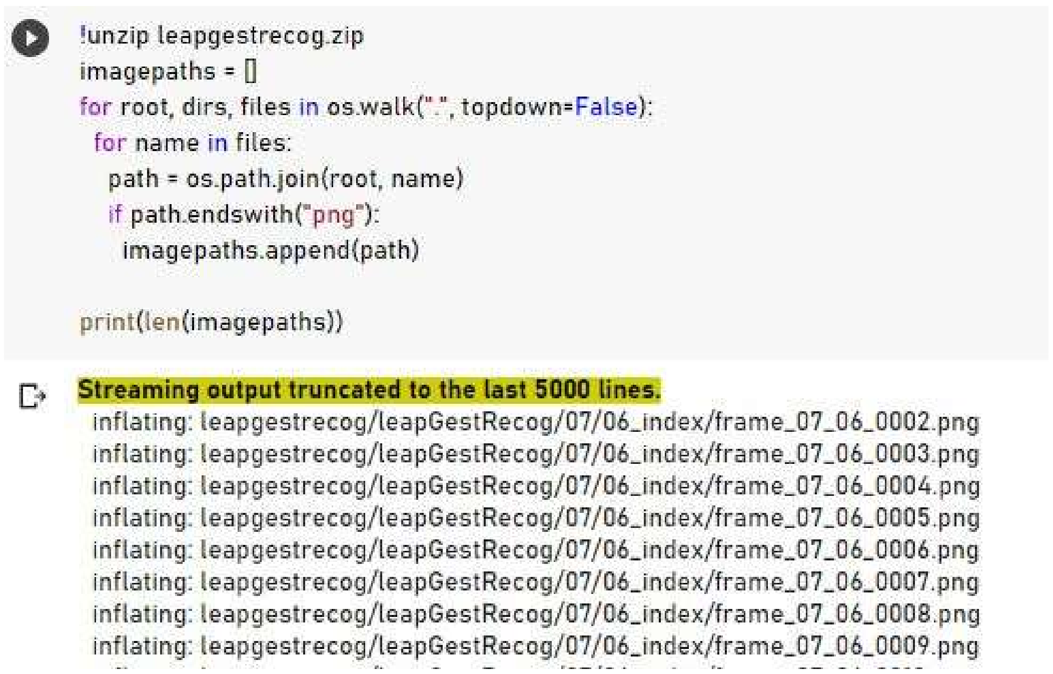
```
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages (1.5.12)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packages (from kaggle) (2.8.1)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from kaggle) (1.24.3)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from kaggle) (2020.12.5)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packages (from kaggle) (5.0.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from kaggle) (4.41.1)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from kaggle) (2.23.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages (from kaggle) (1.15.0)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle) (2.10)
```



```
[5] from google.colab import files
files.upload( )
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download -d gti-upm/leapgestrecog
```

```
Choose Files  No file chosen          Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
Downloading leapgestrecog.zip to /content
 99% 2.11G/2.13G [00:18<00:00, 105MB/s]
100% 2.13G/2.13G [00:18<00:00, 122MB/s]
```

## Unzipping images and accessing all paths for images

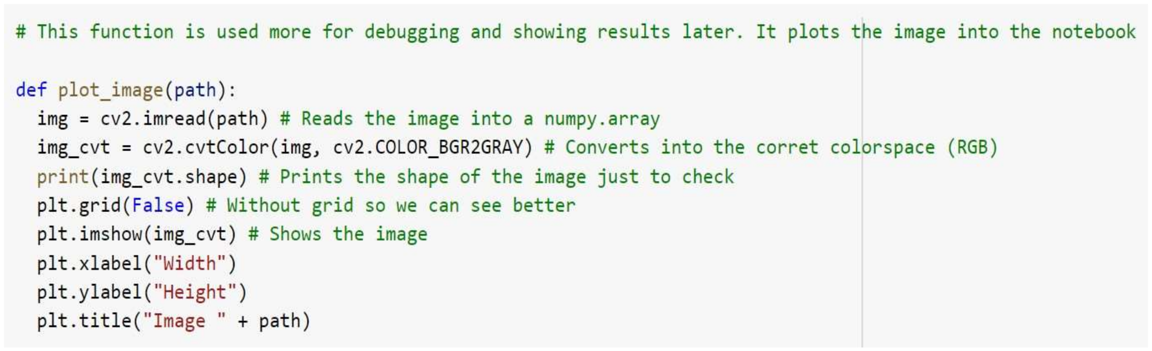


```
!unzip leapgestrecog.zip
imagepaths = []
for root, dirs, files in os.walk(".", topdown=False):
  for name in files:
    path = os.path.join(root, name)
    if path.endswith("png"):
      imagepaths.append(path)

print(len(imagepaths))
```

```
Streaming output truncated to the last 5000 lines.
  inflating: leapgestrecog/leapGestRecog/07/06_index/frame_07_06_0002.png
  inflating: leapgestrecog/leapGestRecog/07/06_index/frame_07_06_0003.png
  inflating: leapgestrecog/leapGestRecog/07/06_index/frame_07_06_0004.png
  inflating: leapgestrecog/leapGestRecog/07/06_index/frame_07_06_0005.png
  inflating: leapgestrecog/leapGestRecog/07/06_index/frame_07_06_0006.png
  inflating: leapgestrecog/leapGestRecog/07/06_index/frame_07_06_0007.png
  inflating: leapgestrecog/leapGestRecog/07/06_index/frame_07_06_0008.png
  inflating: leapgestrecog/leapGestRecog/07/06_index/frame_07_06_0009.png
```

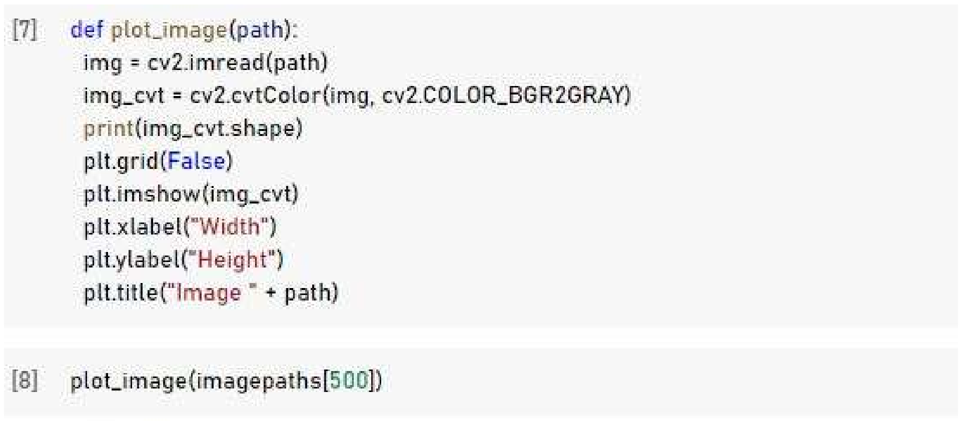
This dataset whole contains 40000 images

## Storing the images in array & converting into the Gray scale



```
# This function is used more for debugging and showing results later. It plots the image into the notebook

def plot_image(path):
  img = cv2.imread(path) # Reads the image into a numpy.array
  img_cvt = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Converts into the corret colorspace (RGB)
  print(img_cvt.shape) # Prints the shape of the image just to check
  plt.grid(False) # Without grid so we can see better
  plt.imshow(img_cvt) # Shows the image
  plt.xlabel("Width")
  plt.ylabel("Height")
  plt.title("Image " + path)
```

## Plotting a random image for checking whether images are loaded are not



```
[7]  def plot_image(path):
       img = cv2.imread(path)
       img_cvt = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
       print(img_cvt.shape)
       plt.grid(False)
       plt.imshow(img_cvt)
       plt.xlabel("Width")
       plt.ylabel("Height")
       plt.title("Image " + path)

[8]  plot_image(imagepaths[500])
```

```
(240, 640)
```

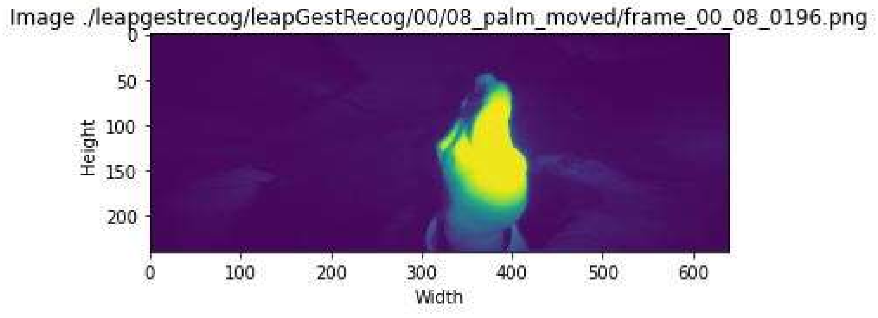
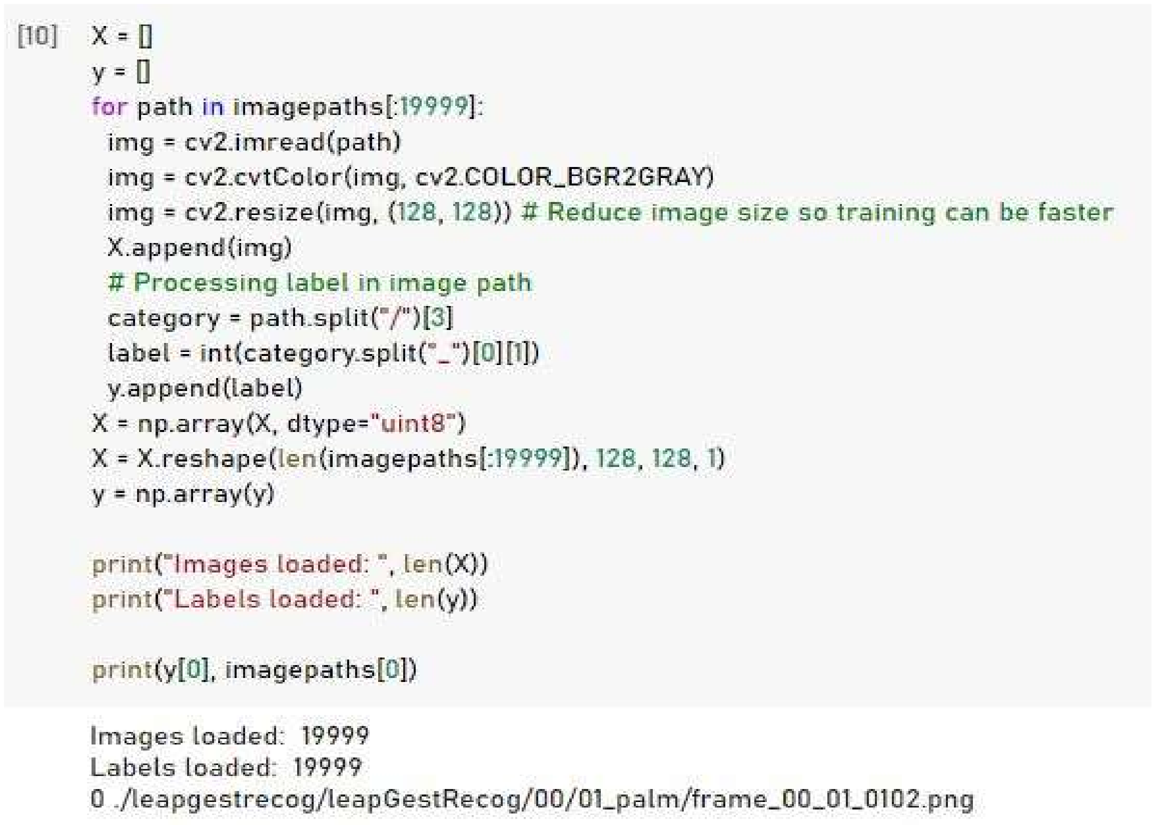


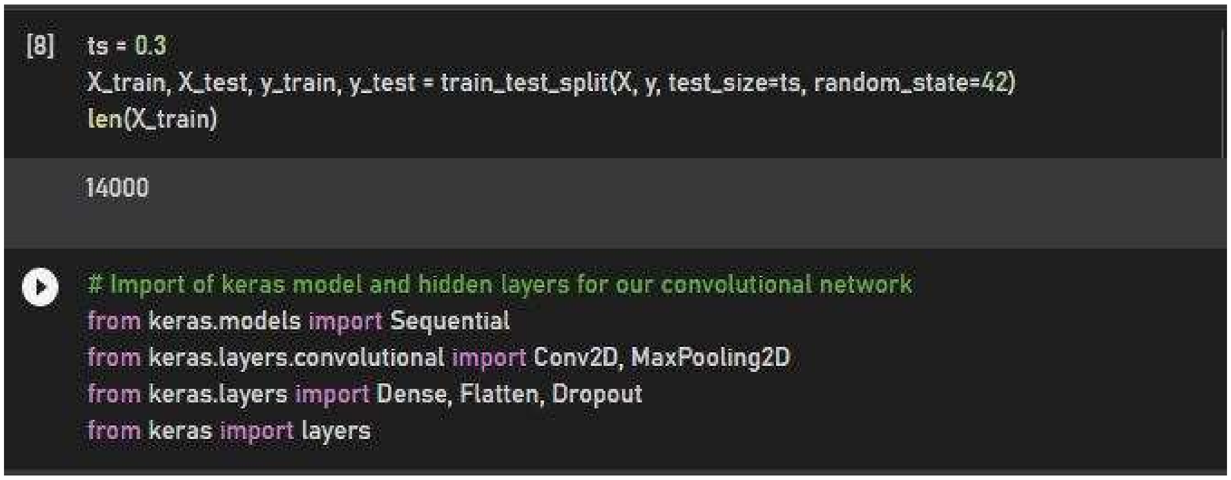
Image ./leapgestrecog/leapGestRecog/00/08_palm_moved/frame_00_08_0196.png
Height
0
50
100
150
200
0
100
200
300
400
500
600
Width

## Resizing and Reshaping the images



```
[10]  X = []
      y = []
      for path in imagepaths[:19999]:
        img = cv2.imread(path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        img = cv2.resize(img, (128, 128)) # Reduce image size so training can be faster
        X.append(img)
        # Processing label in image path
        category = path.split("/")[3]
        label = int(category.split("_")[0][1])
        y.append(label)
      X = np.array(X, dtype="uint8")
      X = X.reshape(len(imagepaths[:19999]), 128, 128, 1)
      y = np.array(y)

      print("Images loaded: ", len(X))
      print("Labels loaded: ", len(y))

      print(y[0], imagepaths[0])

      Images loaded:  19999
      Labels loaded:  19999
      0 ./leapgestrecog/leapGestRecog/00/01_palm/frame_00_01_0102.png
```

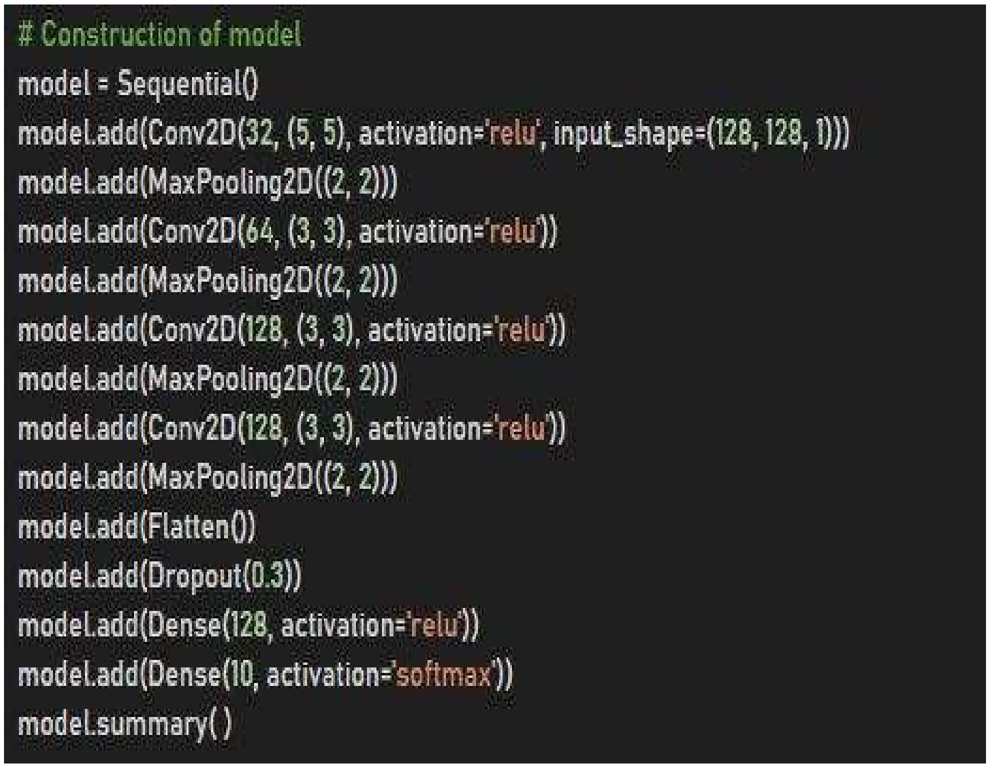
## Splitting the dataset for training and testing



```
[8]   ts = 0.3
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=ts, random_state=42)
      len(X_train)

      14000

      # Import of keras model and hidden layers for our convolutional network
      from keras.models import Sequential
      from keras.layers.convolutional import Conv2D, MaxPooling2D
      from keras.layers import Dense, Flatten, Dropout
      from keras import layers
```

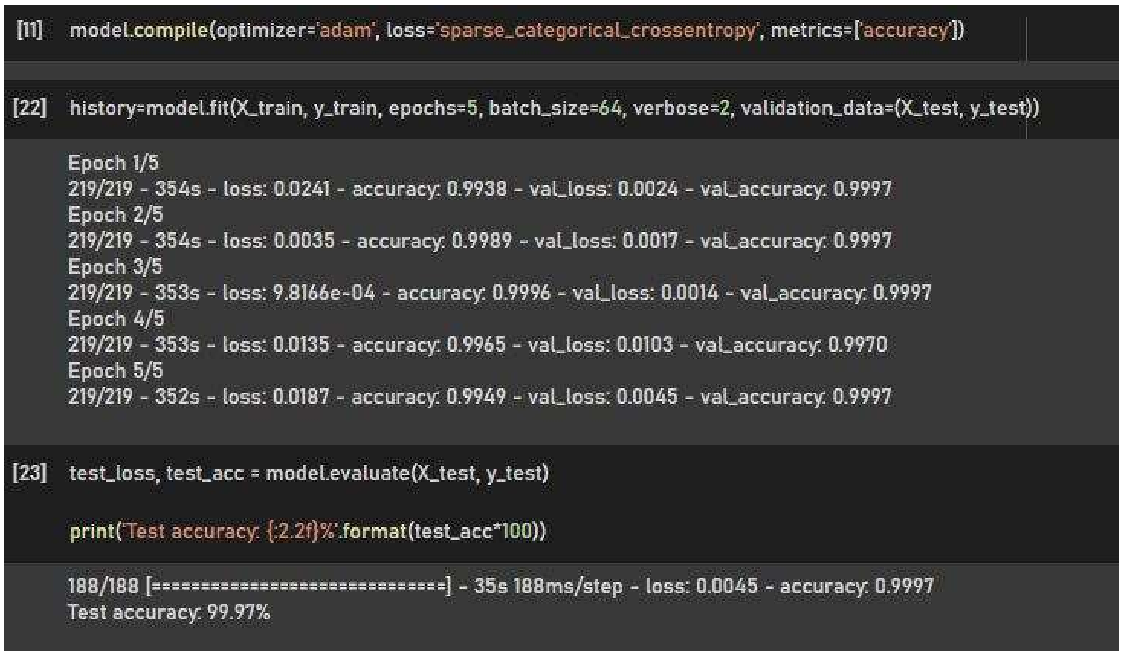
30% images for testing and 70% for training

## Creating a CNN model



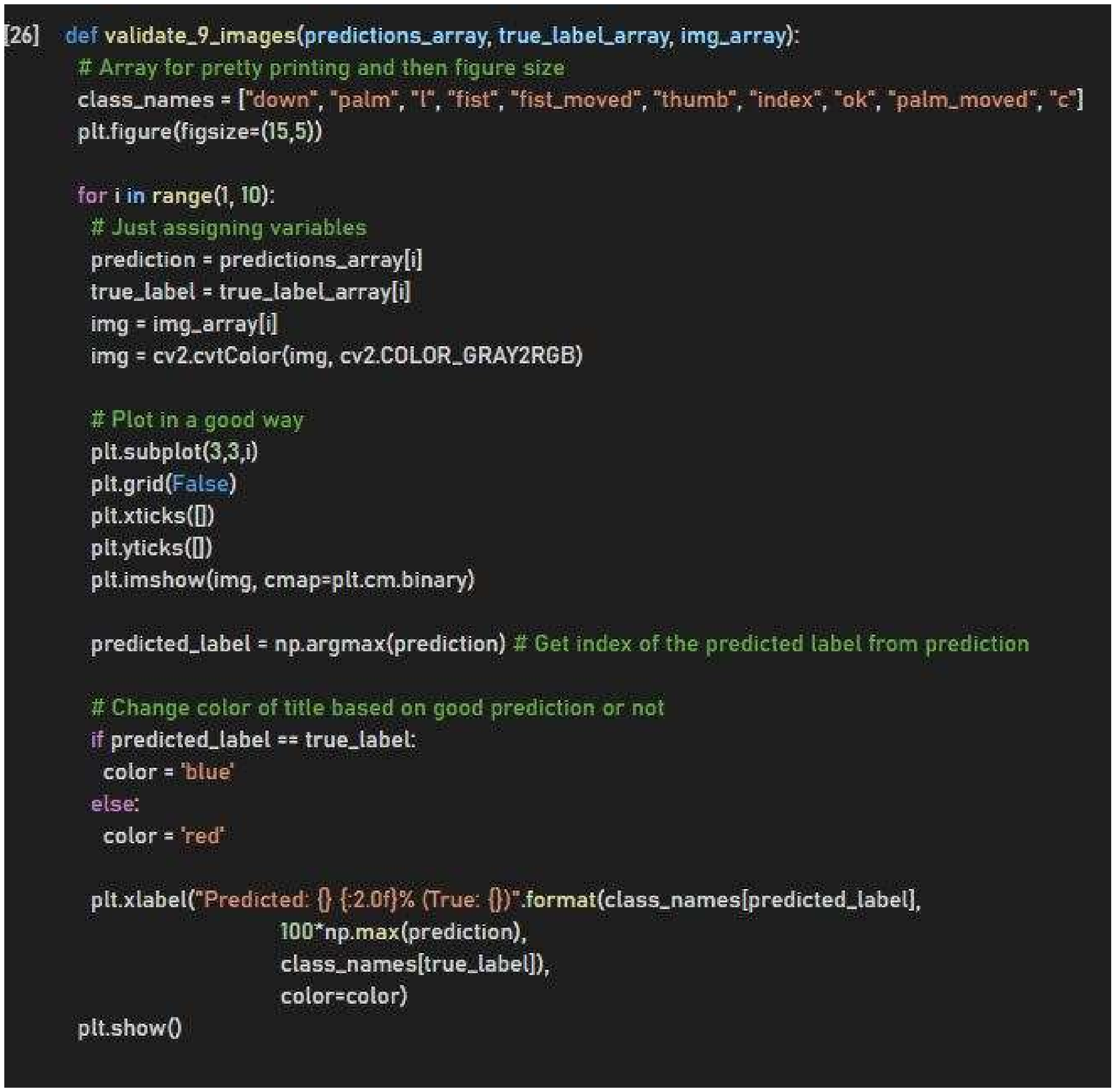
```
# Construction of model
model = Sequential()
model.add(Conv2D(32, (5, 5), activation='relu', input_shape=(128, 128, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dropout(0.3))
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary( )
```

**Training and fitting the data set into the model** and **Metrics accuracy score**



```
[11]  model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

[22]  history=model.fit(X_train, y_train, epochs=5, batch_size=64, verbose=2, validation_data=(X_test, y_test))

      Epoch 1/5
      219/219 - 354s - loss: 0.0241 - accuracy: 0.9938 - val_loss: 0.0024 - val_accuracy: 0.9997
      Epoch 2/5
      219/219 - 354s - loss: 0.0035 - accuracy: 0.9989 - val_loss: 0.0017 - val_accuracy: 0.9997
      Epoch 3/5
      219/219 - 353s - loss: 9.8166e-04 - accuracy: 0.9996 - val_loss: 0.0014 - val_accuracy: 0.9997
      Epoch 4/5
      219/219 - 353s - loss: 0.0135 - accuracy: 0.9965 - val_loss: 0.0103 - val_accuracy: 0.9970
      Epoch 5/5
      219/219 - 352s - loss: 0.0187 - accuracy: 0.9949 - val_loss: 0.0045 - val_accuracy: 0.9997

[23]  test_loss, test_acc = model.evaluate(X_test, y_test)

      print('Test accuracy: {:2.2f}%'.format(test_acc*100))

      188/188 [==============================] - 35s 188ms/step - loss: 0.0045 - accuracy: 0.9997
      Test accuracy: 99.97%
```

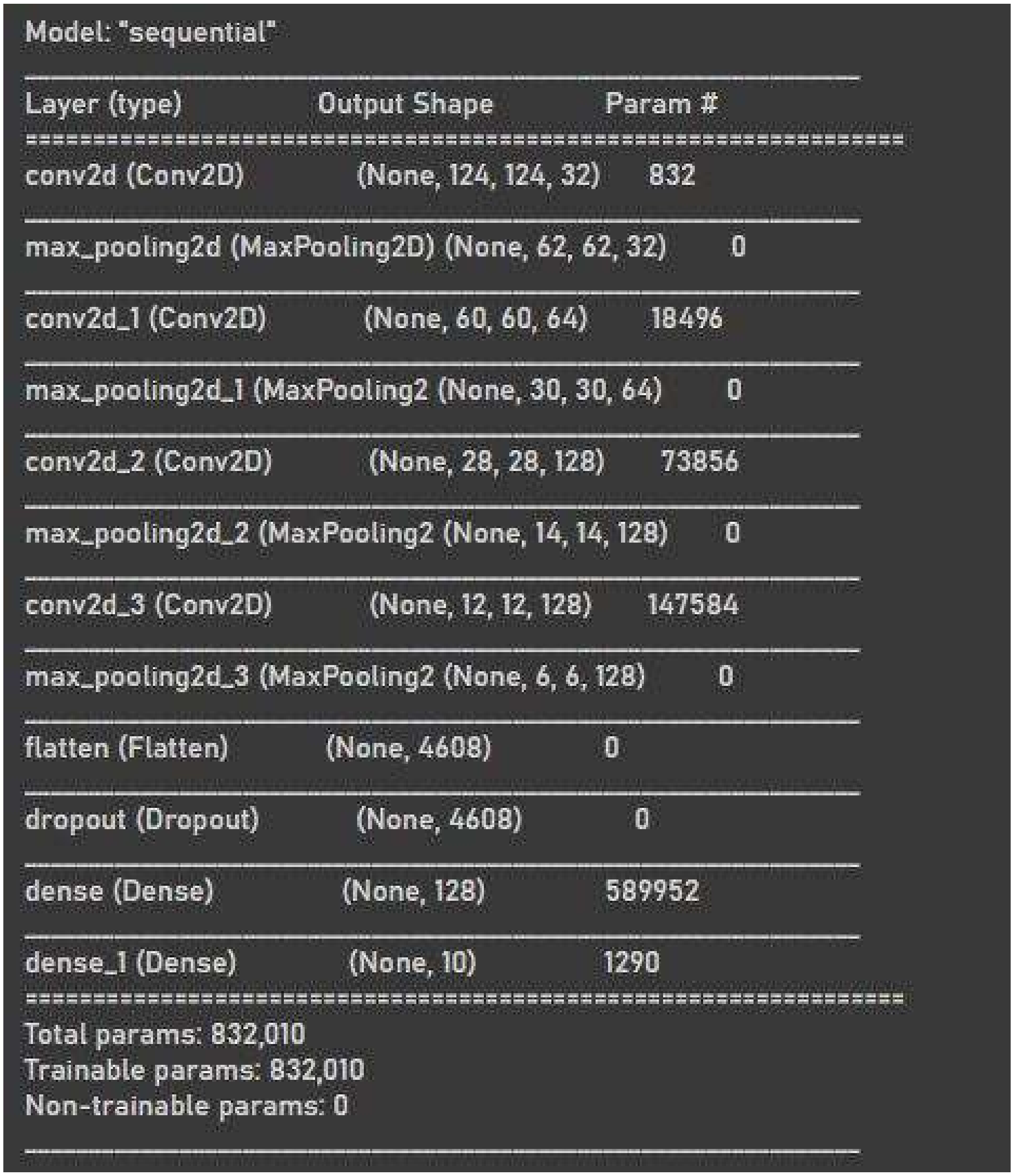
## Plotting predicted and true images



```
[26]  def validate_9_images(predictions_array, true_label_array, img_array):
        # Array for pretty printing and then figure size
        class_names = ["down", "palm", "l", "fist", "fist_moved", "thumb", "index", "ok", "palm_moved", "c"]
        plt.figure(figsize=(15,5))

        for i in range(1, 10):
          # Just assigning variables
          prediction = predictions_array[i]
          true_label = true_label_array[i]
          img = img_array[i]
          img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)

          # Plot in a good way
          plt.subplot(3,3,i)
          plt.grid(False)
          plt.xticks([])
          plt.yticks([])
          plt.imshow(img, cmap=plt.cm.binary)

          predicted_label = np.argmax(prediction) # Get index of the predicted label from prediction

          # Change color of title based on good prediction or not
          if predicted_label == true_label:
            color = 'blue'
          else:
            color = 'red'

          plt.xlabel("Predicted: {} {:2.0f}% (True: {})".format(class_names[predicted_label],
                                        100*np.max(prediction),
                                        class_names[true_label]),
                                        color=color)
        plt.show()
```

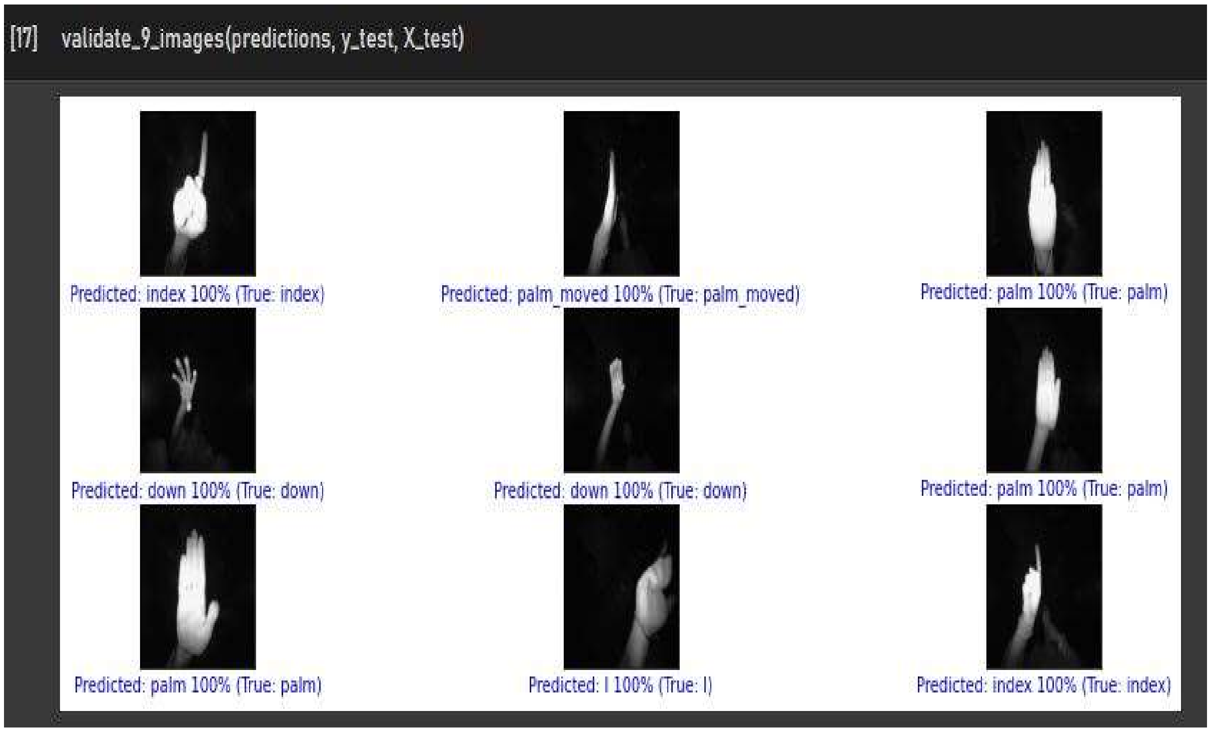
## Results

### Summary of CNN model



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 124, 124, 32)      832
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 62, 62, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 60, 60, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dropout (Dropout)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 128)               589952
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 832,010
Trainable params: 832,010
Non-trainable params: 0
_________________________________________________________________
```

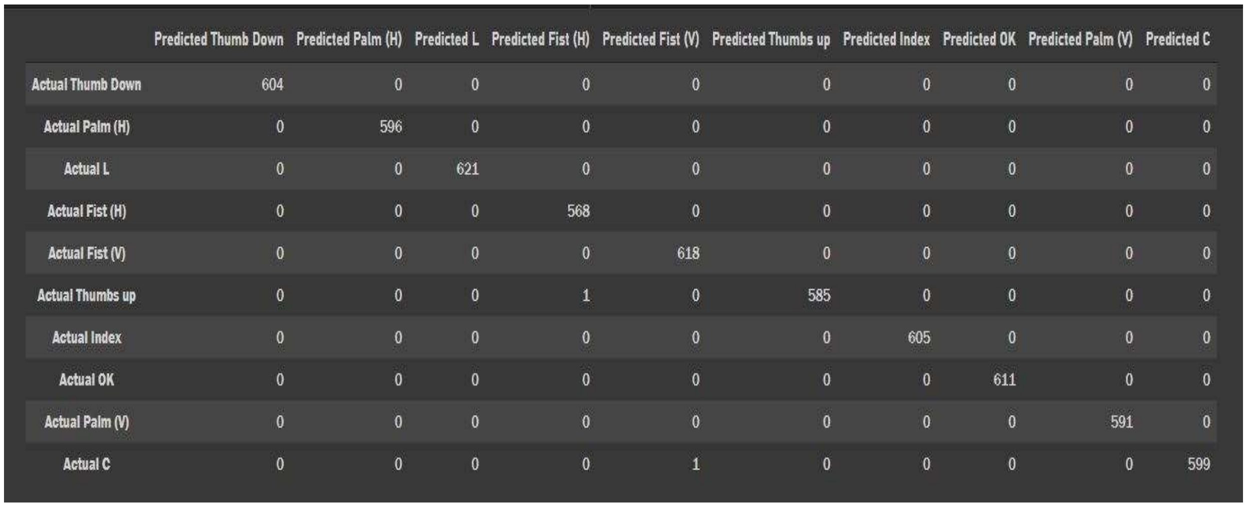
## Predicted vs True images



```
[17]  validate_9_images(predictions, y_test, X_test)
```

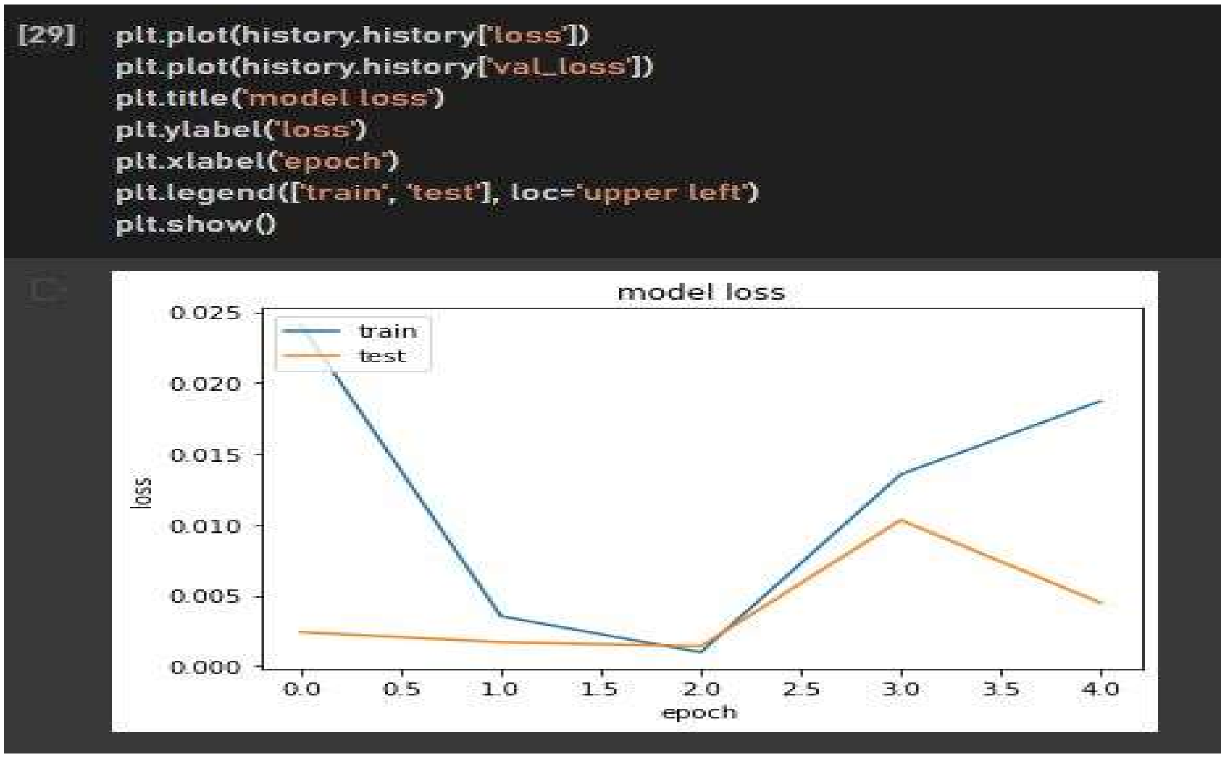
```
Predicted: index 100% (True: index)    Predicted: palm_moved 100% (True: palm_moved)    Predicted: palm 100% (True: palm)
Predicted: down 100% (True: down)      Predicted: down 100% (True: down)                Predicted: palm 100% (True: palm)
Predicted: palm 100% (True: palm)      Predicted: l 100% (True: l)                      Predicted: index 100% (True: index)
```

## Confusion Matrix

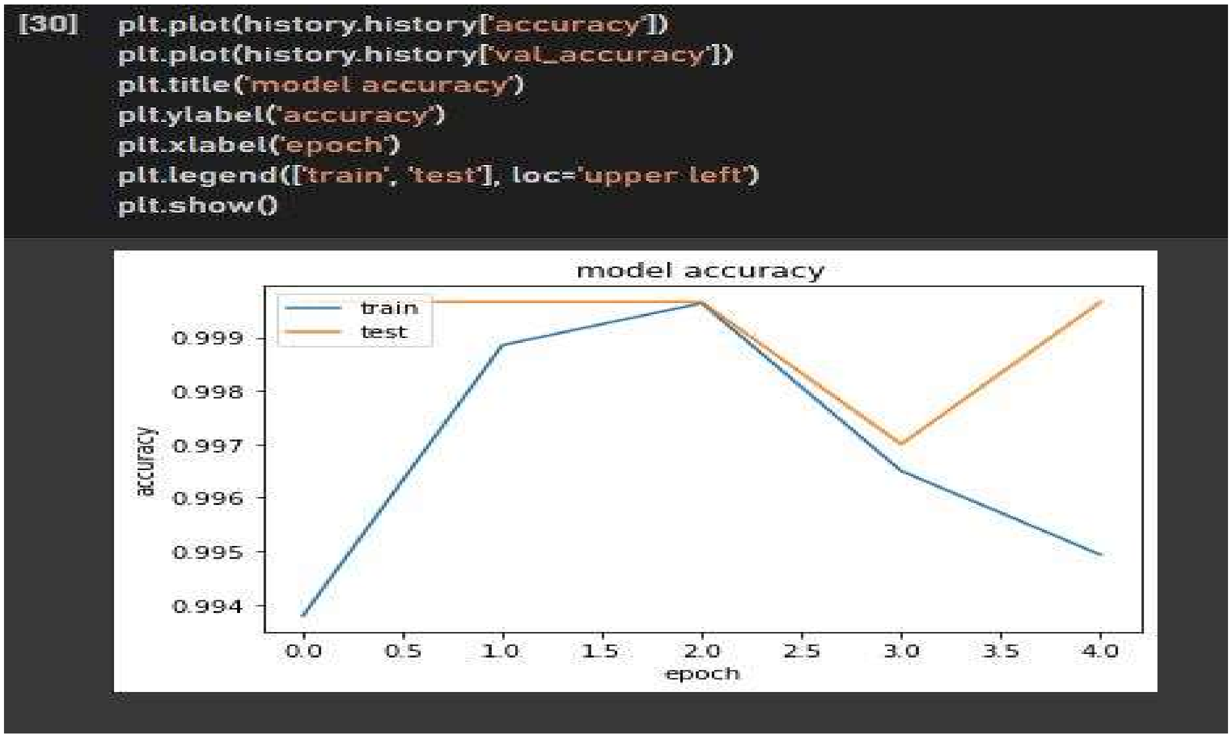


| | Predicted Thumb Down | Predicted Palm (H) | Predicted L | Predicted Fist (H) | Predicted Fist (V) | Predicted Thumbs up | Predicted Index | Predicted OK | Predicted Palm (V) | Predicted C |
|---|---|---|---|---|---|---|---|---|---|---|
| Actual Thumb Down | 604 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Actual Palm (H) | 0 | 596 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Actual L | 0 | 0 | 621 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Actual Fist (H) | 0 | 0 | 0 | 568 | 0 | 0 | 0 | 0 | 0 | 0 |
| Actual Fist (V) | 0 | 0 | 0 | 0 | 618 | 0 | 0 | 0 | 0 | 0 |
| Actual Thumbs up | 0 | 0 | 0 | 1 | 0 | 585 | 0 | 0 | 0 | 0 |
| Actual Index | 0 | 0 | 0 | 0 | 0 | 0 | 605 | 0 | 0 | 0 |
| Actual OK | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 611 | 0 | 0 |
| Actual Palm (V) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 591 | 0 |
| Actual C | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 599 |

## Plots of model accuracy and model loss



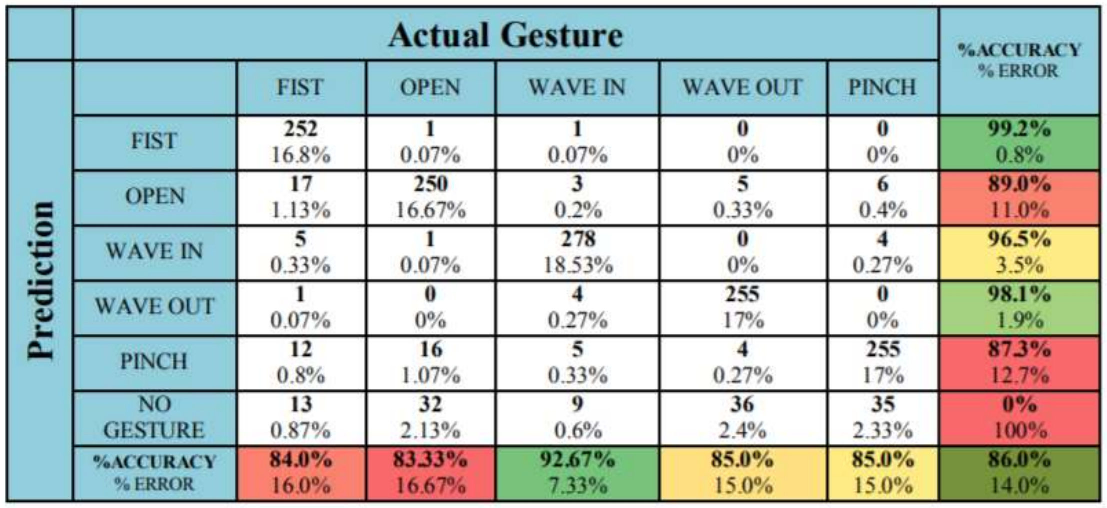
```
[29]  plt.plot(history.history['loss'])
      plt.plot(history.history['val_loss'])
      plt.title('model loss')
      plt.ylabel('loss')
      plt.xlabel('epoch')
      plt.legend(['train', 'test'], loc='upper left')
      plt.show()
```



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

## Comparison with other models:

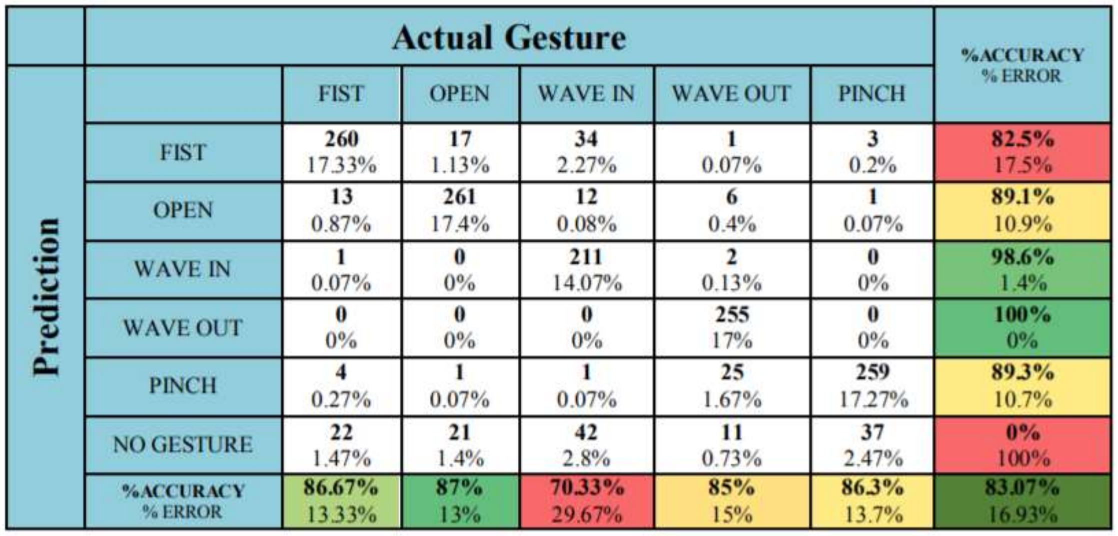
Confusion matrix of the proposed model(machine learning)



| | | Actual Gesture | | | | | %ACCURACY % ERROR |
|---|---|---|---|---|---|---|---|
| | | FIST | OPEN | WAVE IN | WAVE OUT | PINCH | |
| Prediction | FIST | 252 16.8% | 1 0.07% | 1 0.07% | 0 0% | 0 0% | 99.2% 0.8% |
| | OPEN | 17 1.13% | 250 16.67% | 3 0.2% | 5 0.33% | 6 0.4% | 89.0% 11.0% |
| | WAVE IN | 5 0.33% | 1 0.07% | 278 18.53% | 0 0% | 4 0.27% | 96.5% 3.5% |
| | WAVE OUT | 1 0.07% | 0 0% | 4 0.27% | 255 17% | 0 0% | 98.1% 1.9% |
| | PINCH | 12 0.8% | 16 1.07% | 5 0.33% | 4 0.27% | 255 17% | 87.3% 12.7% |
| | NO GESTURE | 13 0.87% | 32 2.13% | 9 0.6% | 36 2.4% | 35 2.33% | 0% 100% |
| | %ACCURACY % ERROR | 84.0% 16.0% | 83.33% 16.67% | 92.67% 7.33% | 85.0% 15.0% | 85.0% 15.0% | 86.0% 14.0% |

Accuracy= 86%

Confusion matrix of the MYO ARMBAND



| | | Actual Gesture | | | | | %ACCURACY % ERROR |
|---|---|---|---|---|---|---|---|
| | | FIST | OPEN | WAVE IN | WAVE OUT | PINCH | |
| Prediction | FIST | 260 17.33% | 17 1.13% | 34 2.27% | 1 0.07% | 3 0.2% | 82.5% 17.5% |
| | OPEN | 13 0.87% | 261 17.4% | 12 0.08% | 6 0.4% | 1 0.07% | 89.1% 10.9% |
| | WAVE IN | 1 0.07% | 0 0% | 211 14.07% | 2 0.13% | 0 0% | 98.6% 1.4% |
| | WAVE OUT | 0 0% | 0 0% | 0 0% | 255 17% | 0 0% | 100% 0% |
| | PINCH | 4 0.27% | 1 0.07% | 1 0.07% | 25 1.67% | 259 17.27% | 89.3% 10.7% |
| | NO GESTURE | 22 1.47% | 21 1.4% | 42 2.8% | 11 0.73% | 37 2.47% | 0% 100% |
| | %ACCURACY % ERROR | 86.67% 13.33% | 87% 13% | 70.33% 29.67% | 85% 15% | 86.3% 13.7% | 83.07% 16.93% |

Accuracy= 83.04%

Here we can see that our accuracy % is much higher than these models and hence we have used this model and got an accuracy of 99.97%

## Conclusion

Based on the results presented in the previous section, we can conclude that our algorithm successfully classifies different hand gestures images with enough confidence (>95%) based on a Deep Learning model.

The accuracy of our model is directly influenced by a few aspects of our problem. The gestures presented are reasonably distinct, the images are clear and without background. Also, there is a reasonable quantity of images, which makes our model more robust. The drawback is that for different problems, we would probably need more data to stir the parameters of our model into a better direction. Moreover, a deep learning model is very hard to interpret, given its abstractions.

## References

Hand Gesture Recognition Using Infrared Imagery Provided by Leap Motion Controller By Tomas Mantecon(B), Carlos R. delBlanco, Fernando Jaureguizar, and Narciso Garcıa , published ACIVS 2016, LNCS 10016, pp. 47–57, 2016 - DOI: 10.1007/978-3- 319-48680-2 5

Hand gesture recognition using neural network based techniques by Vladislava Bobić, Predrag Tadić, Goran Kvaščev on 2016 13th Symposium on Neural Networks and Applications (NEUREL) SAVA Center, Belgrade, Serbia, November 22-24, 2016

Kinect Depth Image Processing for Hand Motion Recognition using Backpropagation Neural Network by Syamsiar Kautsar, Purwadi A. Darwito, and Sryang T. Sarena on The 1st International Seminar on Science and Technology August 5th 2015, Postgraduate Program Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia

Hand Gesture Recognition Using Infrared Imagery Provided by Leap Motion Controller By Tomas Mantecon(B), Carlos R. delBlanco, Fernando Jaureguizar, and Narciso Garcıa , published ACIVS 2016, LNCS 10016, pp. 47–57, 2016 - DOI: 10.1007/978-3- 319-48680-2

Hand gesture recognition using neural network based techniques by Vladislava Bobić, Predrag Tadić, Goran Kvaščev on 2016 13th Symposium on Neural Networks and Applications (NEUREL) SAVA Center, Belgrade, Serbia, November 22-24, 2016

Kinect Depth Image Processing for Hand Motion Recognition using Backpropagation Neural Network by Syamsiar Kautsar, Purwadi A. Darwito, and Sryang T. Sarena on The 1st International Seminar on Science and Technology August 5th 2015, Postgraduate Program Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia

Wu, Xiao Yan. "A hand gesture recognition algorithm based on DC-CNN." *Multimedia Tools and Applications* (2019): 1-13.

Al-Hammadi, Muneer, et al. "Hand gesture recognition using 3D-CNN model." *IEEE Consumer Electronics Magazine* 9.1 (2019): 95-101.

Neethu, P. S., R. Suguna, and Divya Sathish. "An efficient method for human hand gesture detection and recognition using deep learning convolutional neural networks." *Soft Computing* (2020): 1-10.

Lin, Hsien-I., Ming-Hsiang Hsu, and Wei-Kai Chen. "Human hand gesture recognition using a convolution neural network." *2014 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2014.

Varun, Kollipara Sai, I. Puneeth, and T. Prem Jacob. "Hand gesture recognition and implementation for disables using CNN'S." *2019 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2019.

Mohanty, Aparna, Sai Saketh Rambhatla, and Rajiv Ranjan Sahay. "Deep gesture: static hand gesture recognition using CNN." *Proceedings of International Conference on Computer Vision and Image Processing*. Springer, Singapore, 2017.

Chen, Lin, et al. "Hand gesture recognition using compact CNN via surface electromyography signals." *Sensors* 20.3 (2020): 672.