Hand Gesture Recognition

7th Semester Project Report

Project Supervisor: Dr. Sanchita Paul

	Presented by:
Akshay Desai	BE/10357/17
Kumari Nivedita	BE/10245/17
Snigdha Ansu	BE/10124/17

ACKNOWLEDGEMENT

Completing this project has been an absolute pleasure and a learning experience for all of us. Throughout the course of completion, we received a lot of help from people and we would like to take this opportunity to show our gratitude. We would like to show our gratitude to Dr. Sanchita Paul, our project guide, BIT Mesra for her guidance and support throughout. We would also like to thank Dr. I D Ram, our course lecturer, to introduce us to the topic of Neural Networks this semester which helped us with the concept of the project.

We are also grateful to our classmates and seniors for helping us numerous times that helped us to improve our project.
At last, we would like to thank Birla Institute of technology Mesra for giving us this opportunity to go along with this project.

INDEX

OBJECTIVE

- The aim is to build a hand gesture recognition model with deep learning. This model will classify images of different hand gestures, such as a fist, palm, thumb, and others.
- The basic idea is to use data to produce a model that is capable of returning the correct output. This should be done using a webcam in real time.

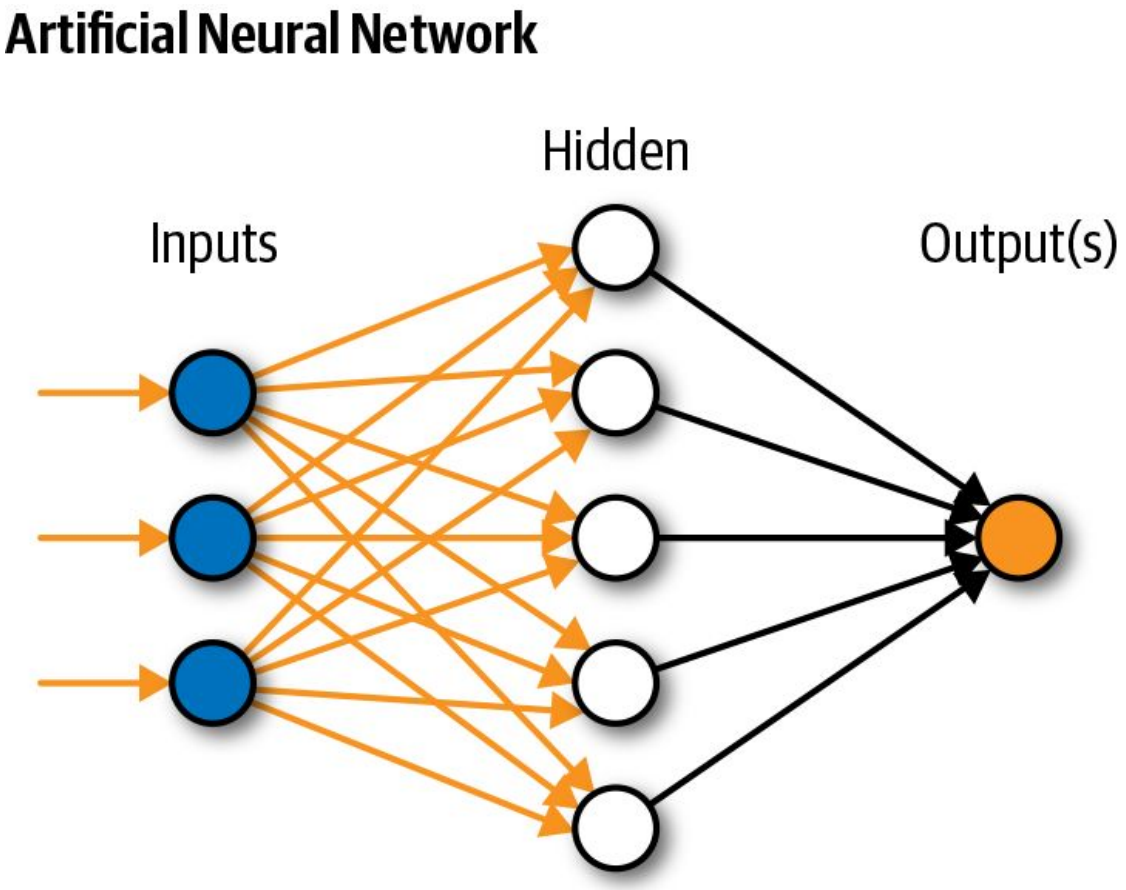
INTRODUCTION

- Machine Learning is very useful for a variety of real-life problems.
- It is commonly used for tasks such as classification, recognition, detection, and predictions.
- The machine learning algorithm which is used for Image Classification here is Convolutional Neural Network(or CNN).
- CNN is an algorithm for machines to understand the features of the image and remember the features to guess the label of the new image fed to the machine.

CONCEPTS USED

Neural Networks

Artificial neural networks (ANNs) are statistical models directly inspired by, and partially modeled on biological neural networks. They are capable of modeling and processing nonlinear relationships between inputs and outputs in parallel. Artificial neural networks are characterized by containing adaptive weights along paths between neurons that can be tuned by a learning algorithm that learns from observed data in order to improve the model.



- Deep Learning:

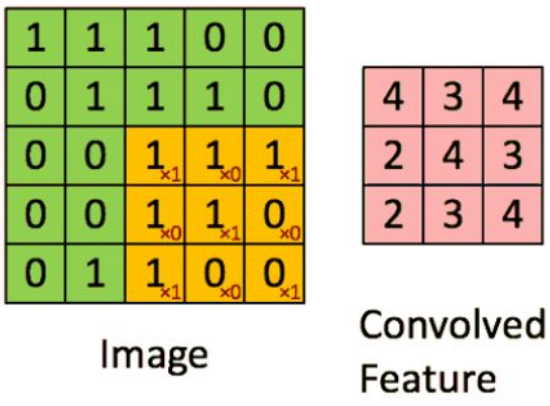
  Deep learning is a subset of machine learning where artificial neural networks, algorithms inspired by the human brain, learn from large amounts of data. Similarly to how we learn from experience, the deep learning algorithm would perform a task repeatedly, each time tweaking it a little to improve the outcome.

- Convolutional Neural Network:

  A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The preprocessing required in CNN is much lower as compared to other classification algorithms.

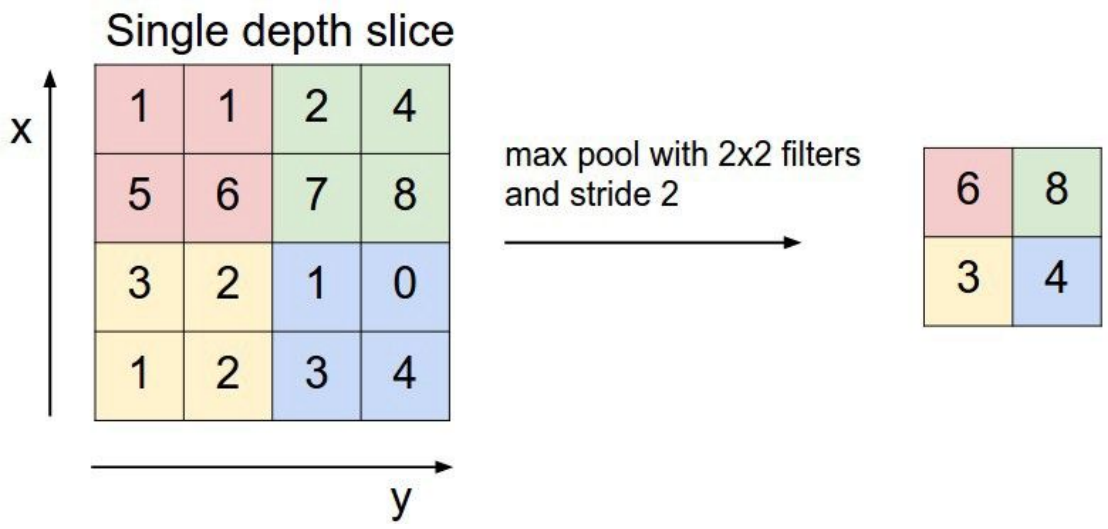
- Convolution filter:

  It extracts features from the input image using sliding matrices to preserve the spatial relations between the pixels. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. The result obtained, having information about that feature, is called a ***feature map***.



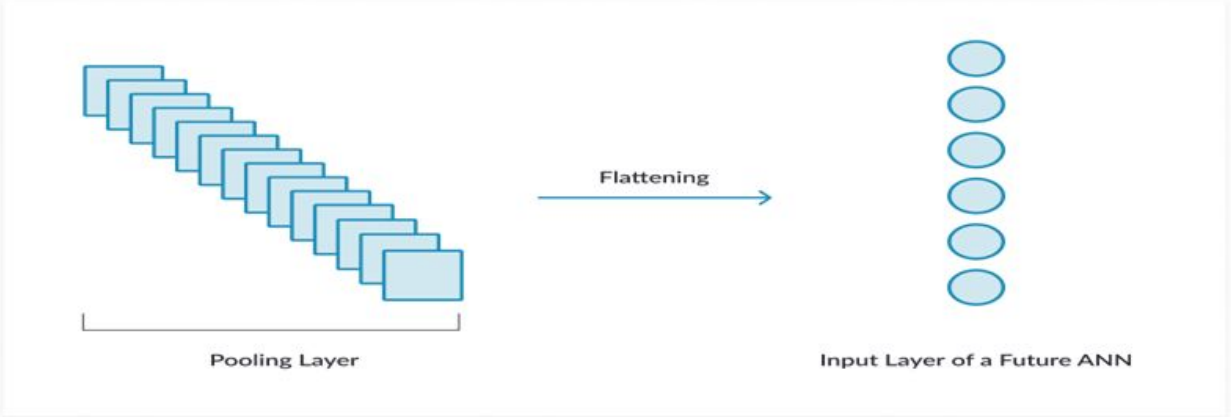
- Pooling:

  Pooling is used to reduce the dimensionality of each feature while retaining the most important information. Similar to the convolutional step, we apply a sliding function to our data. Different functions can be applied: max, sum, mean, etc. The max function usually performs better.



- Flattening :

  Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector. It is connected to the final classification model, which is called a fully-connected layer.



- Dense Layer:

  Dense (fully connected) layers perform classification on the features extracted by the convolutional layers and downsampled by the pooling layers.

- Dropout:

  Dropout reduces overfitting by randomly not updating the weights of some nodes. This helps prevent the NN from relying on one node in the layer too much.

- Activation Function:

  Activation functions are mathematical equations that determine the output of a neural network. The function is attached to each neuron in the network and determines whether it should be activated or not, based on whether each neuron's input is relevant for the model's prediction.

OpenCV

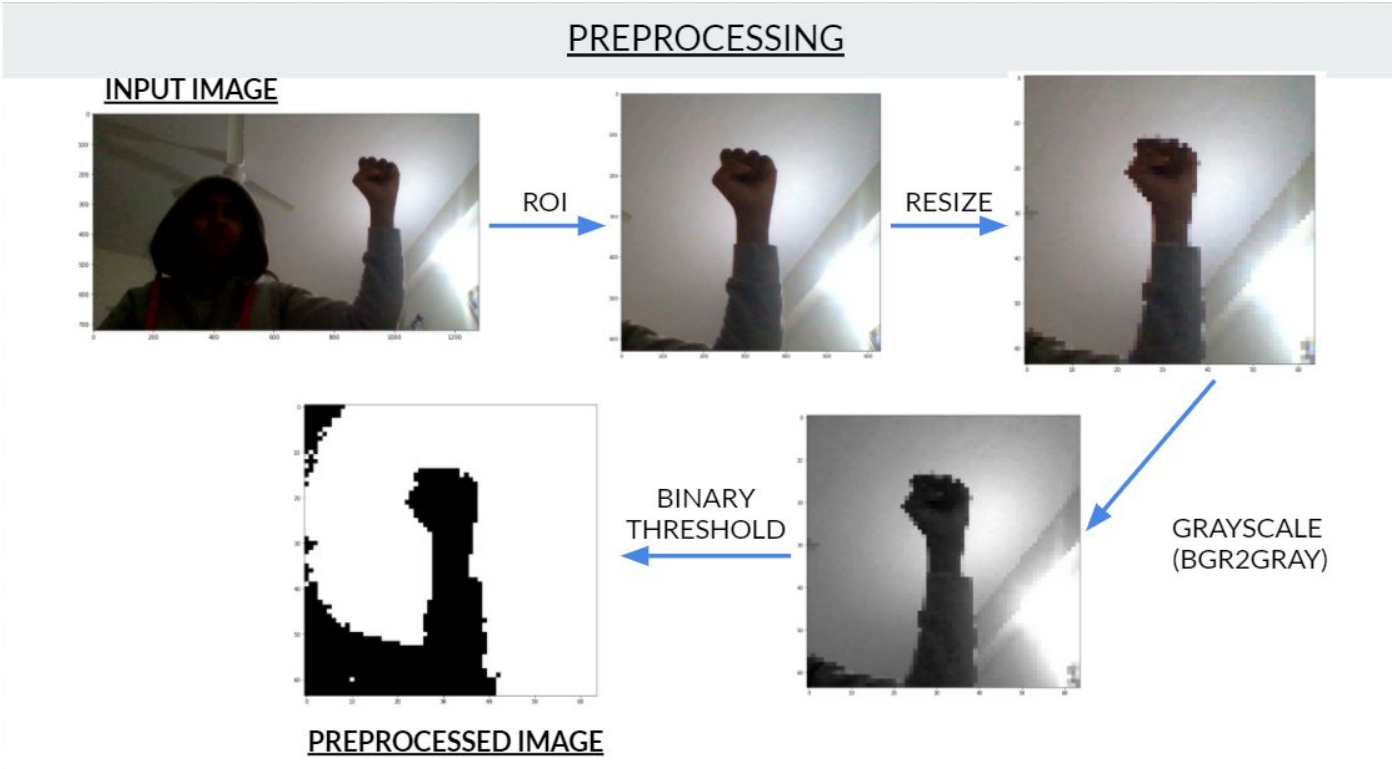
OpenCV is a computer vision library with Python. It is used as an image processing library in many computer vision real-time applications. There are thousands of functions available in OpenCV. These simple techniques are used to shape our images in our required format. As we know an image is a combination of pixels, for a color image we have three channels with pixels ranging from 0 to 225, and for black & white-colored images has only one change ranging from 0 to 1.

PROPOSED METHODOLOGY

- Building a dataset using OpenCV :
  - Taking images for different gestures manually and putting them into test and train datasets.
  - The training dataset contains 6000 images of 6 different classes of hand gestures as follows:

1.INDEX 2.PALM 3.C 4.OK 5.THUMBSUP 6.FIST

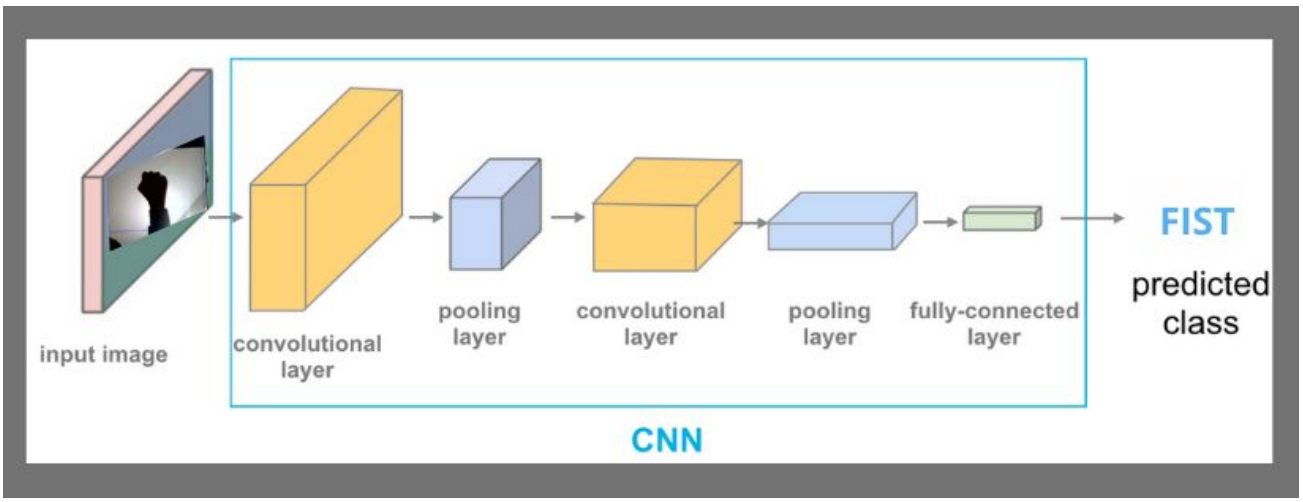
- Preprocessing the data :
  - Extracting the region of interest
  - Images have been resized to 64px x 64px uniform size.
  - Converted the images to grayscale.
  - Binary Thresholding applied for training the model.



- Creating a CNN and train the model :
  - Using a training dataset to train the CNN and test its accuracy.
  - CNN applies a series of filters to the raw pixel data of an image to extract and learn higher-level features, which the model uses for classification.
  - CNN has 3 main components :
    i. **Convolutional** layers - apply a specified number of convolution filters to the image.
    Ex - model.add(Convolution2D(32, (3, 3), input_shape=(64, 64, 1), activation='relu'))
    ii. **Pooling** layers - downsample the image data extracted by the convolutional layers to reduce the dimensionality of the feature map in order to decrease processing time.
    Ex - model.add(MaxPooling2D(pool_size=(2, 2)))
    iii. **Flattening** layer : converts the data into a 1-D array for input to the next layer.
    Ex - model.add(Flatten())
    iv. **Dense** (fully connected) layers - perform classification on the features extracted by the convolutional layers and downsampled by the pooling layers.
    Activation Functions used were :
      a. Relu activation function in hidden layers
      b. Softmax activation function in output layer

    Ex - model.add(Dense(units=128, activation='relu'))
    model.add(Dense(units=6, activation='softmax'))

CNN EXAMPLE :

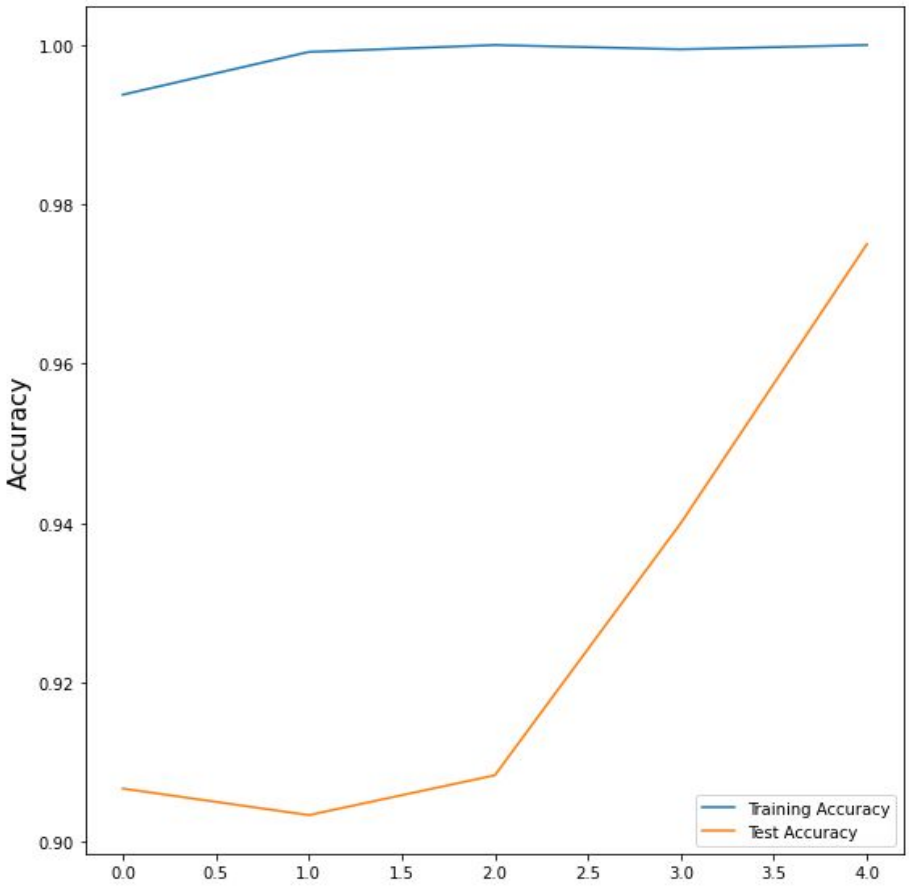


- Predicting real-time results :
    - Using OpenCV, real-time frames are taken, preprocessed, and are classified using the generated model.

Analysis

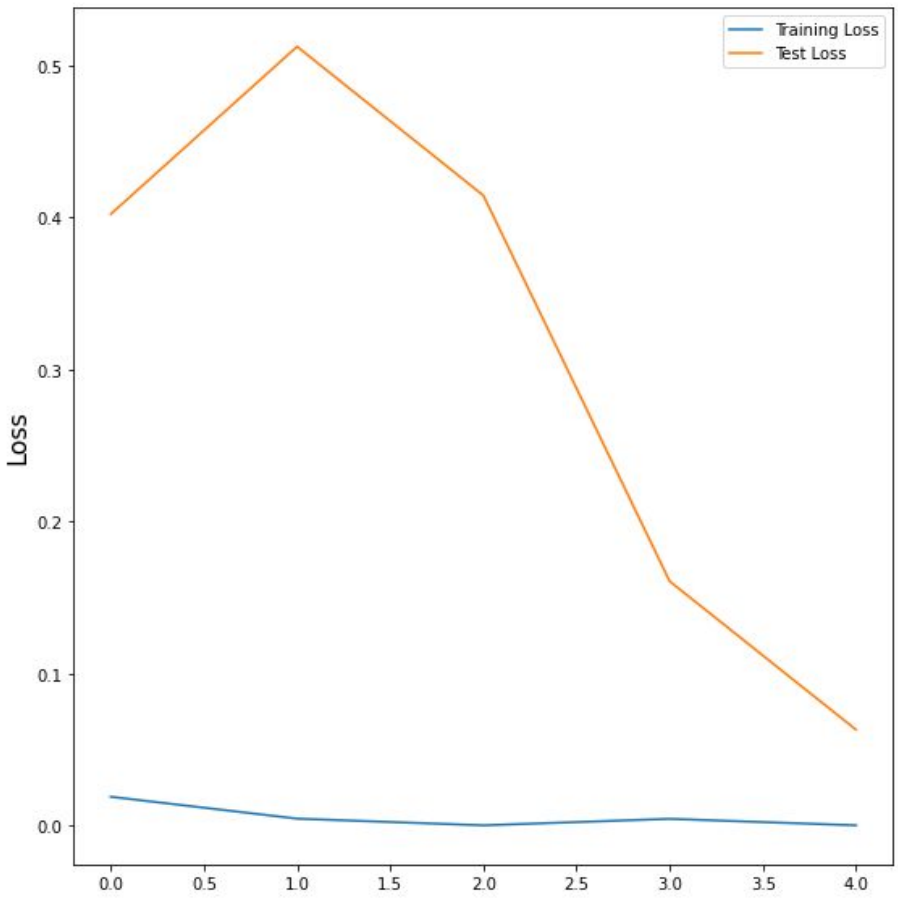
- Training accuracy, test accuracy, training loss, and test loss were the four parameters we used to judge our model
- Test accuracy and loss are important as they indicate how our model performs when it gets a new data
- After 5 epochs, our model gave decent results :

Test accuracy: 0.9750
Test loss: 0.0629



Plotting accuracy against the number of epochs :

The training and test accuracy increase with the increase in the number of epochs.



Plotting loss against the number of epochs :

The training and test loss decreases with an increase in the number of epochs.

CODE

Collect-data.py

```
# Create the directory structure
if not os.path.exists("data"):
    os.makedirs("data")
    os.makedirs("data/train")
    os.makedirs("data/test")
    os.makedirs("data/train/0")
    os.makedirs("data/train/1")
    os.makedirs("data/train/2")
    os.makedirs("data/train/3")
    os.makedirs("data/train/4")
    os.makedirs("data/train/5")
    os.makedirs("data/test/0")
```

```
    os.makedirs("data/test/1")
    os.makedirs("data/test/2")
    os.makedirs("data/test/3")
    os.makedirs("data/test/4")
    os.makedirs("data/test/5")


# Train or test
mode = 'train'
directory = 'data/'+mode+'/'

cap = cv2.VideoCapture(0)

while True:
    _, frame = cap.read()
    # Simulating mirror image
    frame = cv2.flip(frame, 1)

    # Getting count of existing images
    count = {'index': len(os.listdir(directory+"/0")),
             'palm': len(os.listdir(directory+"/1")),
             'c': len(os.listdir(directory+"/2")),
             'ok': len(os.listdir(directory+"/3")),
             'thumbsup': len(os.listdir(directory+"/4")),
             'fist': len(os.listdir(directory+"/5"))}

    # Printing the count in each set to the screen
    cv2.putText(frame, "MODE : "+mode, (10, 50), cv2.FONT_HERSHEY_PLAIN,
1, (0,120,255), 1)
    cv2.putText(frame, "IMAGE COUNT", (10, 100), cv2.FONT_HERSHEY_PLAIN,
1, (0,120,255), 1)
    cv2.putText(frame, "INDEX : "+str(count['index']), (10, 120),
cv2.FONT_HERSHEY_PLAIN, 1, (0,120,255), 1)
    cv2.putText(frame, "PALM : "+str(count['palm']), (10, 140),
cv2.FONT_HERSHEY_PLAIN, 1, (0,120,255), 1)
    cv2.putText(frame, "C : "+str(count['c']), (10, 160),
cv2.FONT_HERSHEY_PLAIN, 1, (0,120,255), 1)
    cv2.putText(frame, "OK : "+str(count['ok']), (10, 180),
cv2.FONT_HERSHEY_PLAIN, 1, (0,120,255), 1)
    cv2.putText(frame, "THUMBSUP : "+str(count['thumbsup']), (10, 200),
cv2.FONT_HERSHEY_PLAIN, 1, (0,120,255), 1)
    cv2.putText(frame, "FIST : "+str(count['fist']), (10, 220),
cv2.FONT_HERSHEY_PLAIN, 1, (0,120,255), 1)

    # Coordinates of the ROI
    x1 = int(0.5*frame.shape[1])
    y1 = 10
    x2 = frame.shape[1]-10
    y2 = int(0.5*frame.shape[1])
    # Drawing the ROI
    # The increment/decrement by 1 is to compensate for the bounding box
    cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (255,0,0) ,1)
    # Extracting the ROI
    roi = frame[y1:y2, x1:x2]
```

```
    roi = cv2.resize(roi, (64, 64))

    cv2.imshow("Frame", frame)

    #_, mask = cv2.threshold(mask, 200, 255, cv2.THRESH_BINARY)
    #kernel = np.ones((1, 1), np.uint8)
    #img = cv2.dilate(mask, kernel, iterations=1)
    #img = cv2.erode(mask, kernel, iterations=1)
    # do the processing after capturing the image!
    roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    _, roi = cv2.threshold(roi, 120, 255, cv2.THRESH_BINARY)
    cv2.imshow("ROI", roi)

    interrupt = cv2.waitKey(10)
    if interrupt & 0xFF == 27: # esc key
        break
    if interrupt & 0xFF == ord('0'):
        cv2.imwrite(directory+'0/'+str(count['index'])+'.jpg', roi)
    if interrupt & 0xFF == ord('1'):
        cv2.imwrite(directory+'1/'+str(count['palm'])+'.jpg', roi)
    if interrupt & 0xFF == ord('2'):
        cv2.imwrite(directory+'2/'+str(count['c'])+'.jpg', roi)
    if interrupt & 0xFF == ord('3'):
        cv2.imwrite(directory+'3/'+str(count['ok'])+'.jpg', roi)
    if interrupt & 0xFF == ord('4'):
        cv2.imwrite(directory+'4/'+str(count['thumbsup'])+'.jpg', roi)
    if interrupt & 0xFF == ord('5'):
        cv2.imwrite(directory+'5/'+str(count['fist'])+'.jpg', roi)

cap.release()
cv2.destroyAllWindows()
```

Train.py

```
#Preprocessing the data
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

training_set = train_datagen.flow_from_directory('data/train',
                                                 target_size=(64, 64),
```

```
                                                 batch_size=5,
                                                 color_mode='grayscale',
                                                 class_mode='categorical')

test_set = test_datagen.flow_from_directory('data/test',
                                            target_size=(64, 64),
                                            batch_size=5,
                                            color_mode='grayscale',
                                            class_mode='categorical')

# Building the CNN
model = Sequential()
model.add(Convolution2D(32, (5, 5), activation='relu', input_shape=(64,
64, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(6, activation='softmax'))

# Compiling the CNN
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy']) # categorical_crossentropy for more than 2

# Training the model

history = model.fit_generator(
        training_set,
        steps_per_epoch=6000, # No of images in training set
        epochs=5,
        validation_data=test_set,
        validation_steps=600)# No of images in test set


# Saving the model
 model_json = model.to_json()
 with open("model-bw.json", "w") as json_file:
 json_file.write(model_json)
 model.save_weights('model-bw.h5')
```

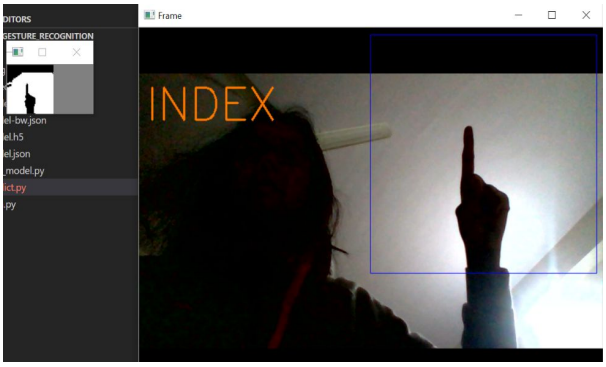
Live-prediction.py

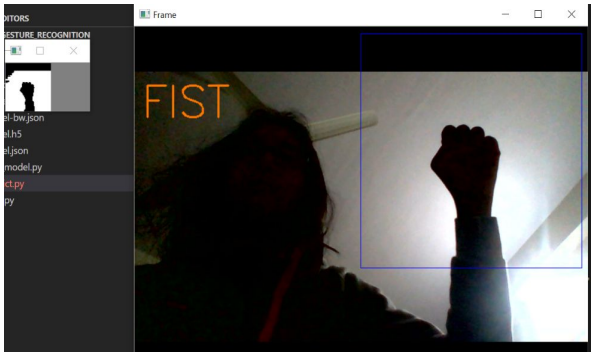
```
# Loading the model
json_file = open("model-bw.json", "r")
model_json = json_file.read()
json_file.close()
model = model_from_json(model_json)
# load weights into new model
model.load_weights("model-bw.h5")
```

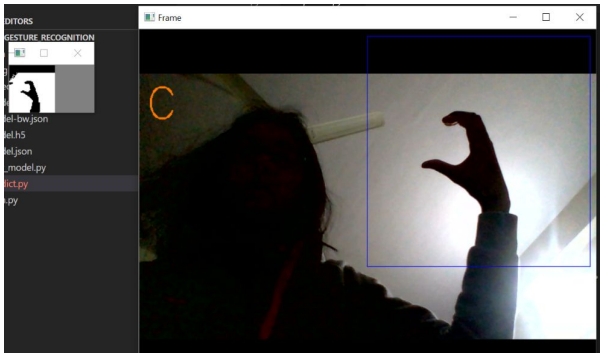
```
cap = cv2.VideoCapture(0)


# Class labels
classes = ['INDEX', 'PALM', 'C', 'OK', 'THUMBSUP', 'FIST']

while True:
    _, frame = cap.read()
    # Simulating mirror image
    frame = cv2.flip(frame, 1)

    # Got this from collect-data.py
    # Coordinates of the ROI
    x1 = int(0.5*frame.shape[1])
    y1 = 10
    x2 = frame.shape[1]-10
    y2 = int(0.5*frame.shape[1])
    # Drawing the ROI
    # The increment/decrement by 1 is to compensate for the bounding box
    cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (255,0,0) ,1)
    # Extracting the ROI
    roi = frame[y1:y2, x1:x2]

    # Resizing the ROI so it can be fed to the model for prediction
    roi = cv2.resize(roi, (64, 64))
    roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    _, test_image = cv2.threshold(roi, 120, 255, cv2.THRESH_BINARY)
    cv2.imshow("test", test_image)
    # Batch of 1
    result = model.predict(test_image.reshape(1, 64, 64, 1))

    # Displaying the predictions
    cv2.putText(frame, classes[np.argmax(result)], (10, 120),
cv2.FONT_HERSHEY_SIMPLEX, 2, (0,120,255), 2)
    cv2.imshow("Frame", frame)

    interrupt = cv2.waitKey(10)
    if interrupt & 0xFF == 27: # esc key
        break


cap.release()
cv2.destroyAllWindows()
```

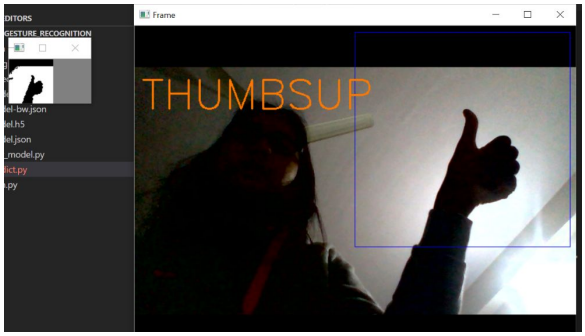
RESULT

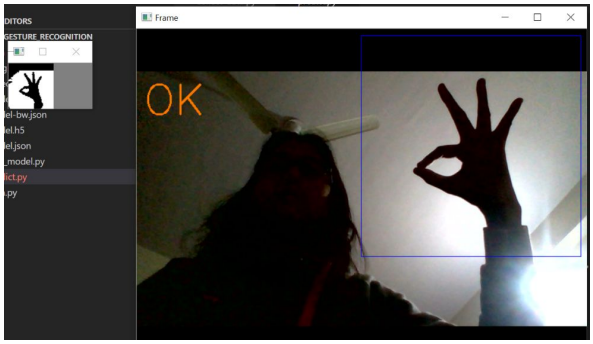

Frame
PALM


Frame
INDEX


Frame
FIST


Frame
C


Frame
THUMBSUP


Frame
OK

FUTURE SCOPE

- This model could be used for communicating with the deaf and dumb using these various gestures.
- Enhancing the model to work on a 3-D dataset will allow us to extend the idea to this goal.
- This will help to make the communication between the deaf/dumb and the people who don’t understand sign language a lot easier.
- It can also be used in applications involving gesture navigation.

CONCLUSION

- Based on the results presented in the previous section, we can conclude that our algorithm successfully classifies different hand gesture 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.

REFERENCES

- Using Deep Learning and CNNs to make a Hand Gesture recognition model
    - https://towardsdatascience.com/tutorial-using-deep-learning-and-cnns-to-make-a-hand-gesture-recognition-model-371770b63a51
- From raw images to real-time predictions with Deep Learning
    - https://towardsdatascience.com/from-raw-images-to-real-time-predictions-with-deep-learning-ddbbda1be0e4