CHAPTER 1

INTRODUCTION

Controlling appliances and electronics through hand gestures is an important and still developing technology. Being able to completely detach from electronics to control appliances is very difficult to achieve as there are many ways an appliance may receive wrong commands. It is important that the system of control is robust enough to not let the appliance accept wrong/ false commands and still achieve the goal of communication. This project is the first step to detach the need of touch to control an appliance. There are many ways of achieving this, the most common is the use of a smaller appliance attached to the user that can detect the movement and communicate the information further as per requirement. Some other ways are detection of gestures through a camera, detection of the motion through a camera. This detection involves subdomains of computer vision such as object tracking and recognition of text.

Object tracking is one of the important applications in computer vision. Object tracking helps in understanding the environment and helps the computers or machines respond to the stimulus of movement appropriately. Writing text on air to control the applications in the machines involves object tracking, then it further involves the prediction of the instruction i.e. the recognition of the text.

Recognition of the text written is also an important application in computer vision. Recognition of the text in natural images with the help of sliding windows and connected component analysis are mainstream in this domain. Recognition of text is particularly difficult as it involves the high variation in font, size, orientation and the complicated backgrounds.

Our project involves the recognition of one single digit that is written on the air[1]. To achieve this we have adopted a deep learning based approach. The model adopted is convolutional neural network (CNN) which is most commonly used in the analyses of visual imagery. They are robust to shift and space variance based on their shared weights architecture and translation invariance. CNNs structure helps in avoiding overfitting in the model. The model needs to be trained on a dataset so that the trained model can be used in the prediction or recognition of the written text. The dataset used for this purpose is MNIST dataset. The Dataset consists of 60,000 training images of handwritten digits from 0-9. Recognition/ prediction can be achieved through this pre trained network.

[1] "Text Writing in Air - arXiv." https://arxiv.org/pdf/1604.08245. Accessed 19 Jul. 2020.

OBJECTIVE OF THE PROJECT

To track an object with the help of the webcam of a laptop and store the data written with that object. The object used can be fingertip or of any specific color, we opted to track objects with a specific color. Once we get the data from tracking the object we then predict the data written on thin air with the help of trained CNN. This has many applications and some of them are listed below.

APPLICATIONS

Application involves making the electronics smarter and making the accessing of the electronics easier. Every industry is dependent on machines/ electronics and hence the application lies in every industry. Some specific applications are following.

HOME AUTOMATION

Home automation is where the appliances in the specific house have easy accessibility, which is generally achieved through the internet of things (IoT). Through linking all the devices by internet the accessibility of the devices is made easier. The application is giving the command to that device can be done through hand gestures, making the home even smarter.

EDUCATIONAL SECTOR

One of the applications lies in the educational sector by replacing the traditional teaching technique of writing on the board with chalk or marker. This makes the teaching easier and it can also be made interactive.

AUTOMOBILE INDUSTRY

Driver distractions are a huge problem for traffic safety. Automobile manufacturers are coming up with more natural ways to control the infotainment system helping the driver’s to keep eye on the road. Easy way to change the volume, to accept calls as they drive etc. BMW’s camera based gesture control system is one example.

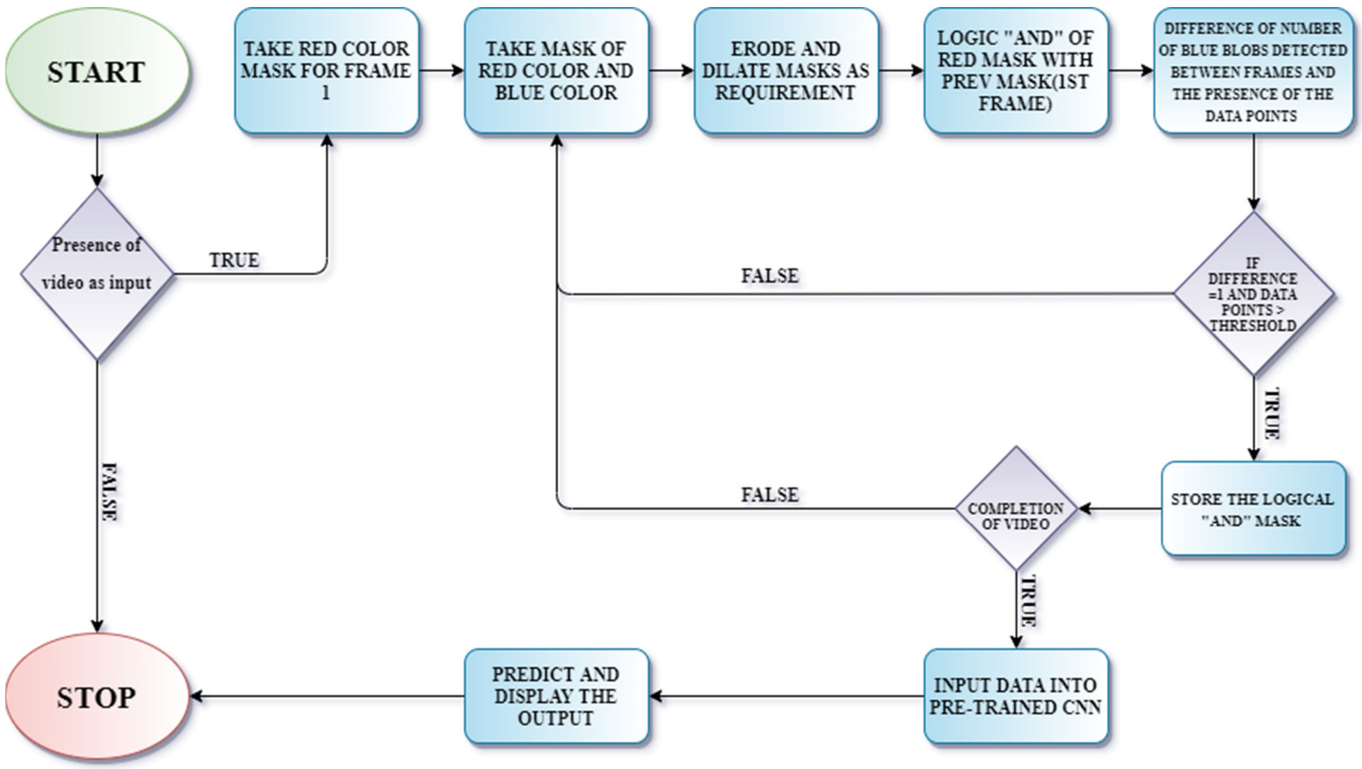
ROBOT/DRONE CONTROL/GAMING

Controlling robots and drones to do tasks through gestures is also one of the applications. Summoning the drone back or commanding it to capture an image, such commands are easier done with gestures. Similarly virtual reality gaming is a growing industry which gives the experience of playing games through gestures.

CHAPTER 2

IMPLEMENTATION

BLOCK DIAGRAM


START
Presence of video as input
TRUE
FALSE
TAKE RED COLOR MASK FOR FRAME 1
TAKE MASK OF RED COLOR AND BLUE COLOR
ERODE AND DILATE MASKS AS REQUIREMENT
LOGIC "AND" OF RED MASK WITH PREV MASK(1ST FRAME)
DIFFERENCE OF NUMBER OF BLUE BLOBS DETECTED BETWEEN FRAMES AND THE PRESENCE OF THE DATA POINTS
IF DIFFERENCE =1 AND DATA POINTS > THRESHOLD
FALSE
TRUE
STORE THE LOGICAL "AND" MASK
COMPLETION OF VIDEO
FALSE
TRUE
INPUT DATA INTO PRE-TRAINED CNN
PREDICT AND DISPLAY THE OUTPUT
STOP

SOFTWARE USED

ANACONDA: It is a python prepackaged distribution of python which contains python modules and packages, including JUPYTER.

JUPYTER NOTEBOOK: It is an Integrated Development Environment (IDE) and used as a coding tool, to write, test, and debug written code easily.

TENSORFLOW and KERAS[2]: Tensorflow and keras are the most popular frameworks in Deep Learning. Tensorflow is an open source platform for machine learning. It has a flexible ecosystem of tools, libraries and other resources that provide workflow with a high level API (application programming interface). Keras, on the other hand, is a high level neural networks library which is running on the top of Tensorflow, i.e. they are integrated with each other.

[2] "Keras Conv2D and Convolutional Layers - PyImageSearch." 31 Dec. 2018, https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/. Accessed 19 Jul. 2020.

OBJECT TRACKING

The major part of the project involves tracking of the object. We surveyed a few methods which would help in the tracking. One of them is taking the histogram of the object that is to be tracked and then subtracting the background environment in real time.

HISTOGRAM MATCHING

Histogram is the representation of the frequency of occurrence of pixels in certain intensity. Tracking an object by this method has its own advantages and disadvantages. Advantages being that this will not give false detection that might occur due to illumination changes which is generally a major problem in object tracking. Disadvantages are that the detection can be false if the background has similar histogram, so the background being of different colour than that of the object that is to be detected becomes a necessity.

In our project we need to detect the finger tip for the tracking. Using this method causes few problems as we wouldn't be able to have anything in the frame that matches the skin color such as the face of the person or the background needs to be of a contrasting colour to get proper detection. Adequate results of detection were not achieved with this method in our project hence we didn’t opt for it.

DETECTION THROUGH COLOR

This method involves the detection of the certain color in the frame and tracking the colored object. A image is a combination of red, green, blue intensities and tracking a colored object can be difficult if the image/frame is being dealt in RGB, so to make it easier we deal with frames in HSV i.e. hue saturation value. The HSV model describes colors similarly to how the human eye tends to perceive color.

Hue represents the color, saturation represents the amount to which that respective color is mixed with white and value represents the amount to which that respective color is mixed with black (gray level). This model helps in separating the image luminance from color information.

This method involves masking of the detected object and then proceeding to track. Once we mask the image we will be able to detect the centroid of the detected blob. Centroid tracking is an efficient way to track an object but in our project we didn’t just want the tracking information we rather wanted to retain the entire information of the blob as that was easier for us. We opted to perform bitwise AND logical operation in each frame to track and store the information of movement of the object.

We opted with this method of tracking because there was no necessity of not having the similar histogram as to skin if we were to track the tip of the finger and histogram method to track a specific color was redundant.

Method followed in the project

The method in the project involves the detection of red and blue color in the frame and masking frame. Once we attain the masked images we proceed to track the red blob by performing bitwise AND with the previous frame red color masked image. Since the mask of the detected blob is black, bitwise AND would store the previous frames information and add the information of the present frame.

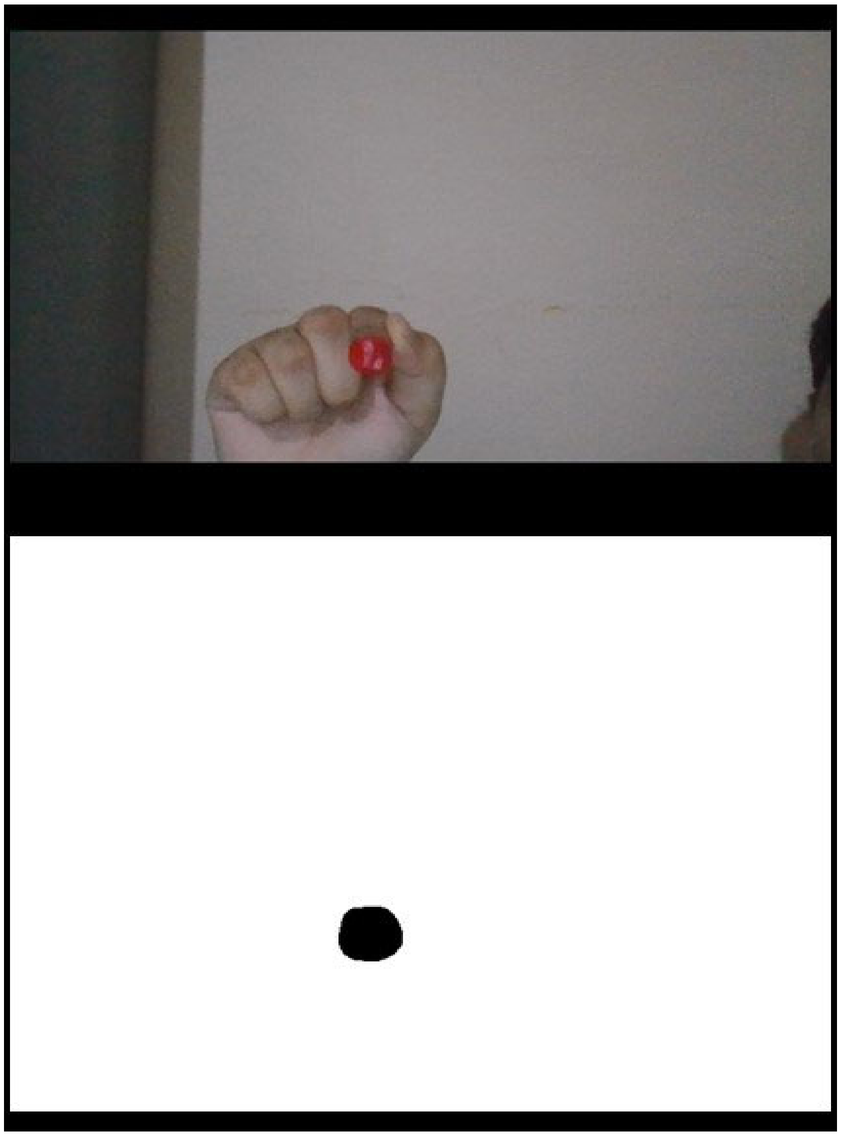


Fig: THIS IMAGE IS THE MASK OF RED COLOR IN THE FRAME.

This is frame to frame detection of red blob and writing information. The information written is the digit 1.

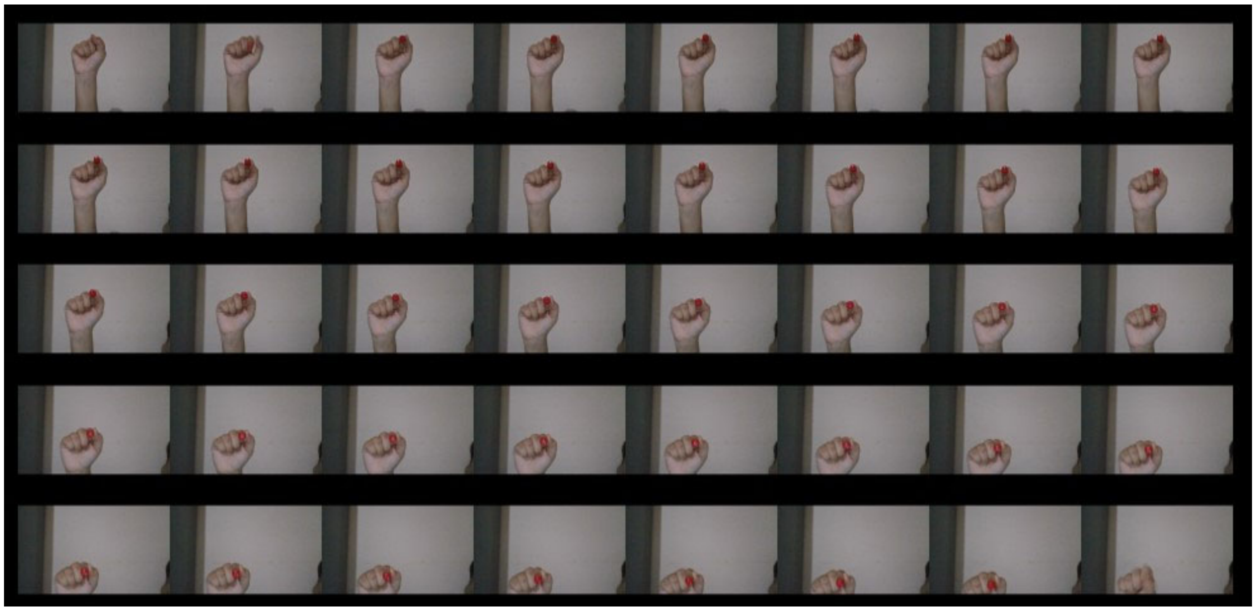


Fig: FRAMES OF WRITING THE DIGIT 1.

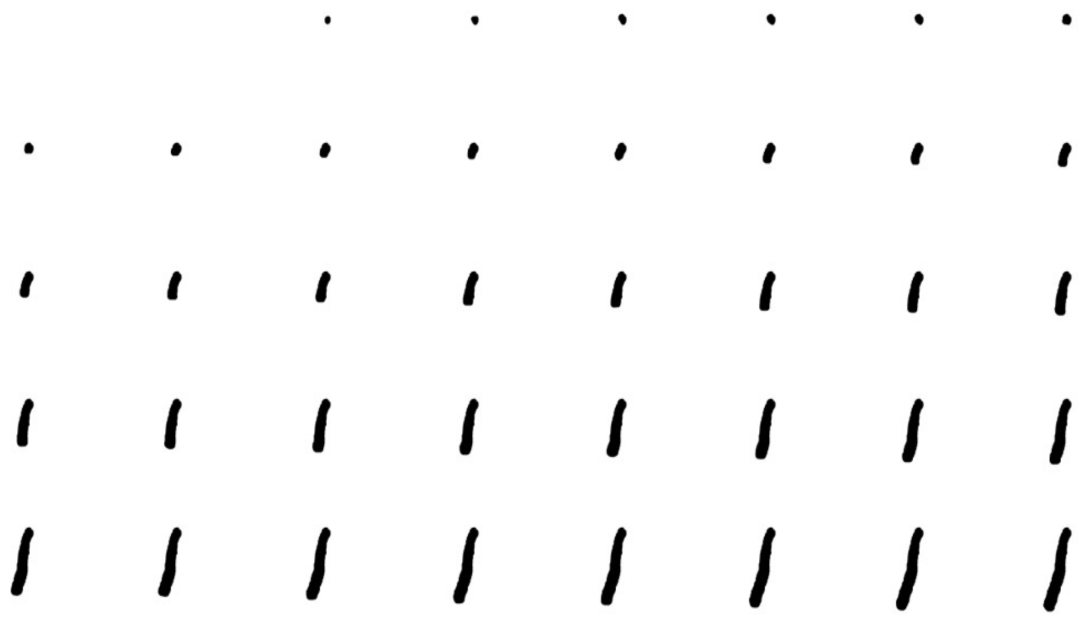


Fig: FRAMES OF MASKED RED COLOR WRITING DIGIT 1(retaining the information of previous frame).

This is frame to frame tracking of red blob and the information being written is digit 2.

Fig: FRAMES OF WRITING THE DIGIT 2.

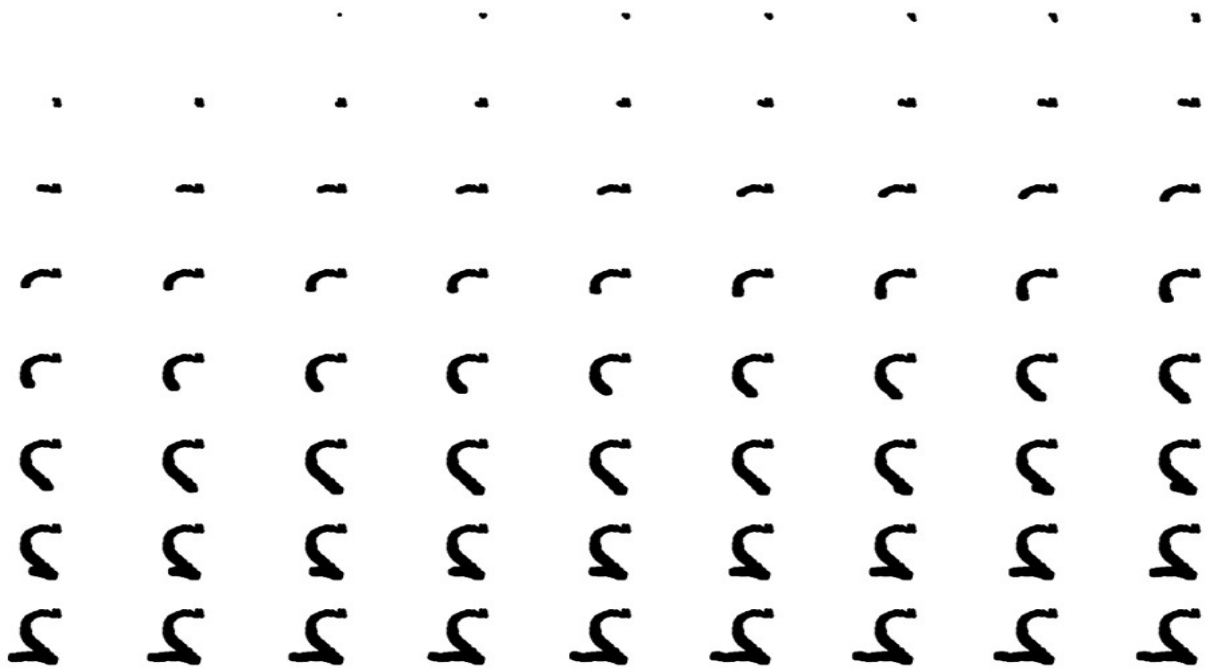


Fig: FRAMES OF MASKED RED COLOR WRITING DIGIT 2(retaining the information of previous frame).

This is frame to frame tracking of red blob and the information being written is digit 3.

Fig: FRAMES OF WRITING THE DIGIT 3.

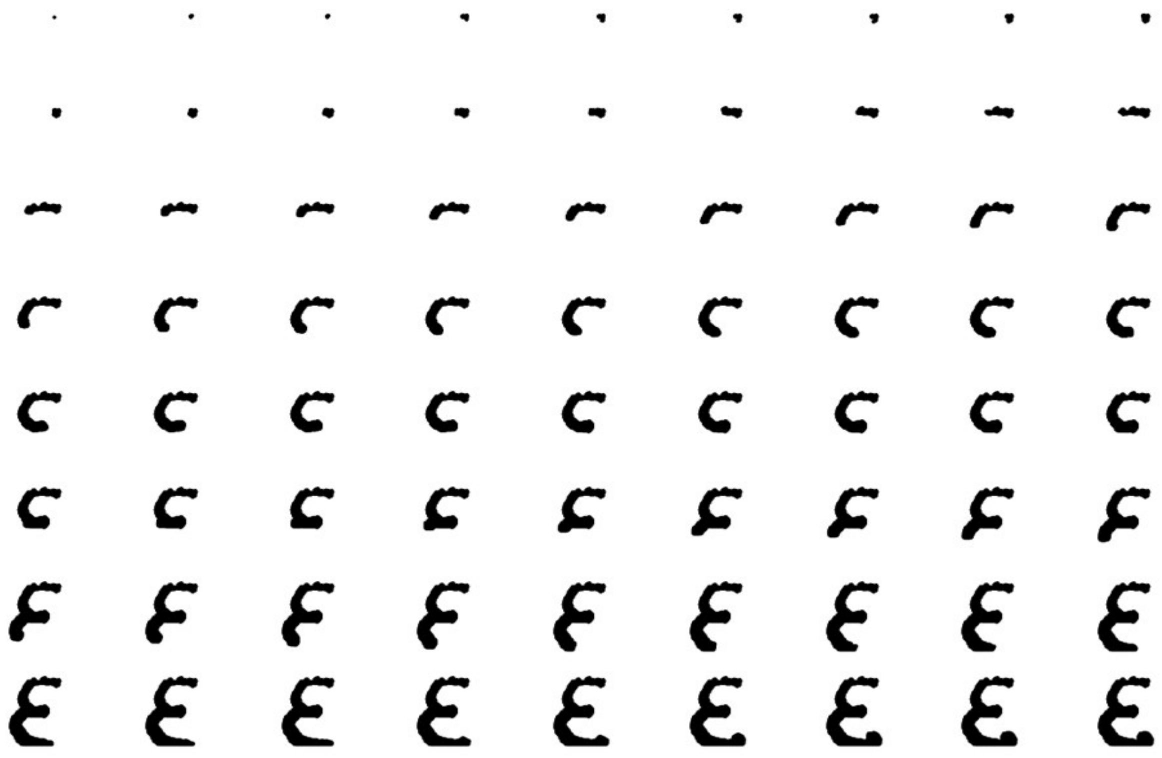


Fig: FRAMES OF MASKED RED COLOR WRITING DIGIT 3(retaining the information of previous frame).

This is frame to frame tracking of red blob and the information being written is digit 6.

Fig: FRAMES OF WRITING THE DIGIT 6.

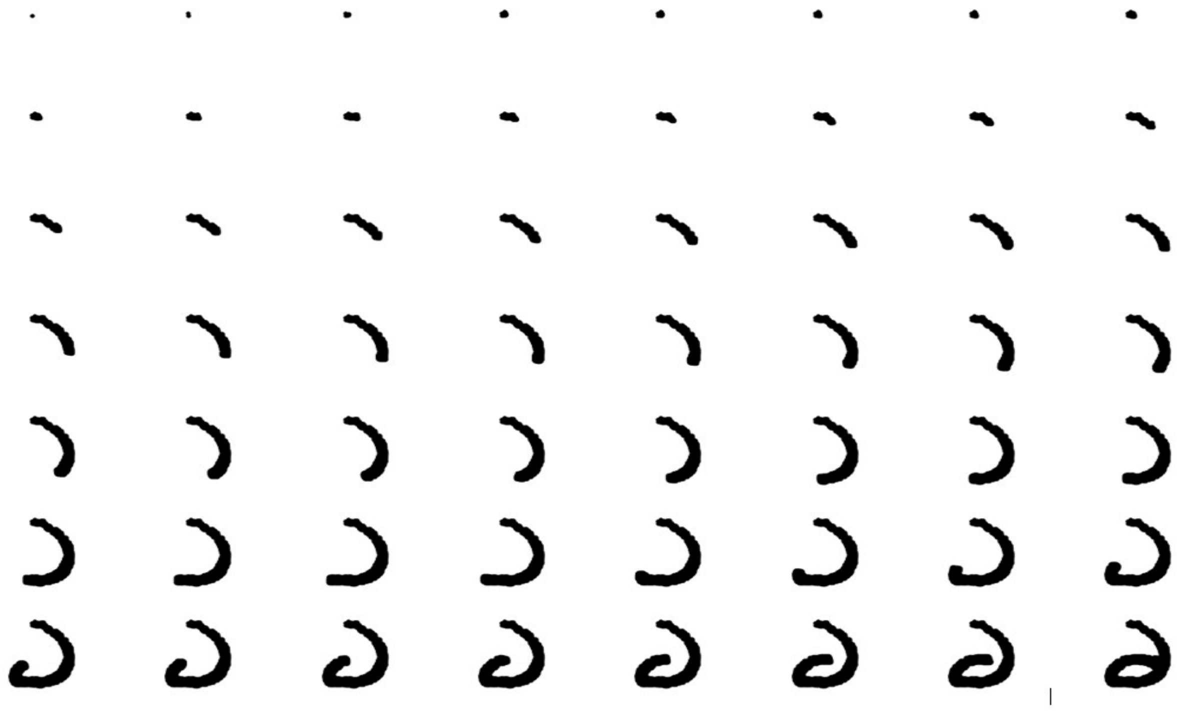


Fig: FRAMES OF MASKED RED COLOR WRITING DIGIT 6(retaining the information of previous frame).

Erasing of the mask to write more information or digits is important. We achieved that with the use of detection of the color blue.We calculate the number of blue blobs detected in each frame and once we detect the blue blob for the first time in the frame, we check if enough information has been written on the masked image, if so then we clear the mask so that we can write more information. Blue color detection is used as an all clear for the screen.

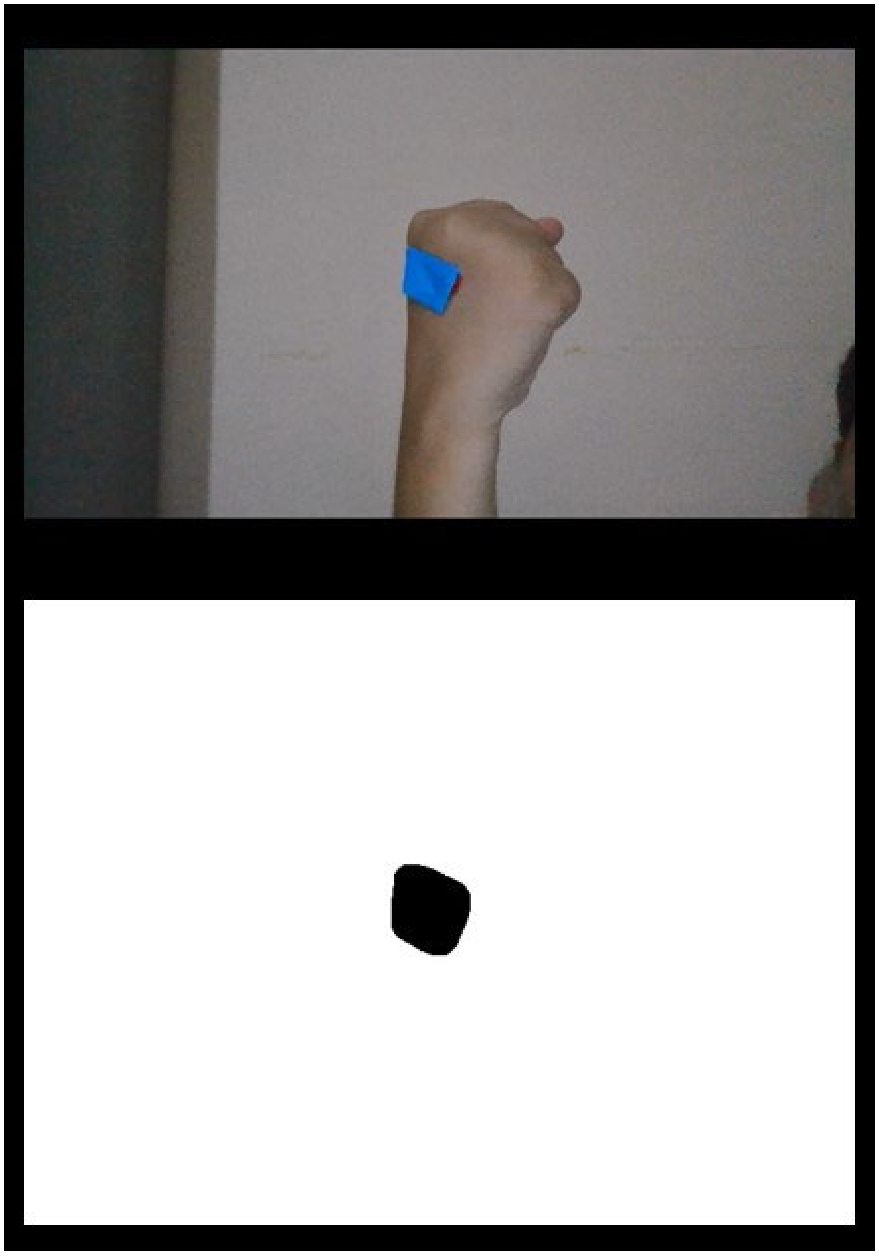


Fig: THIS IMAGE IS THE MASK OF BLUE COLOR IN FRAME

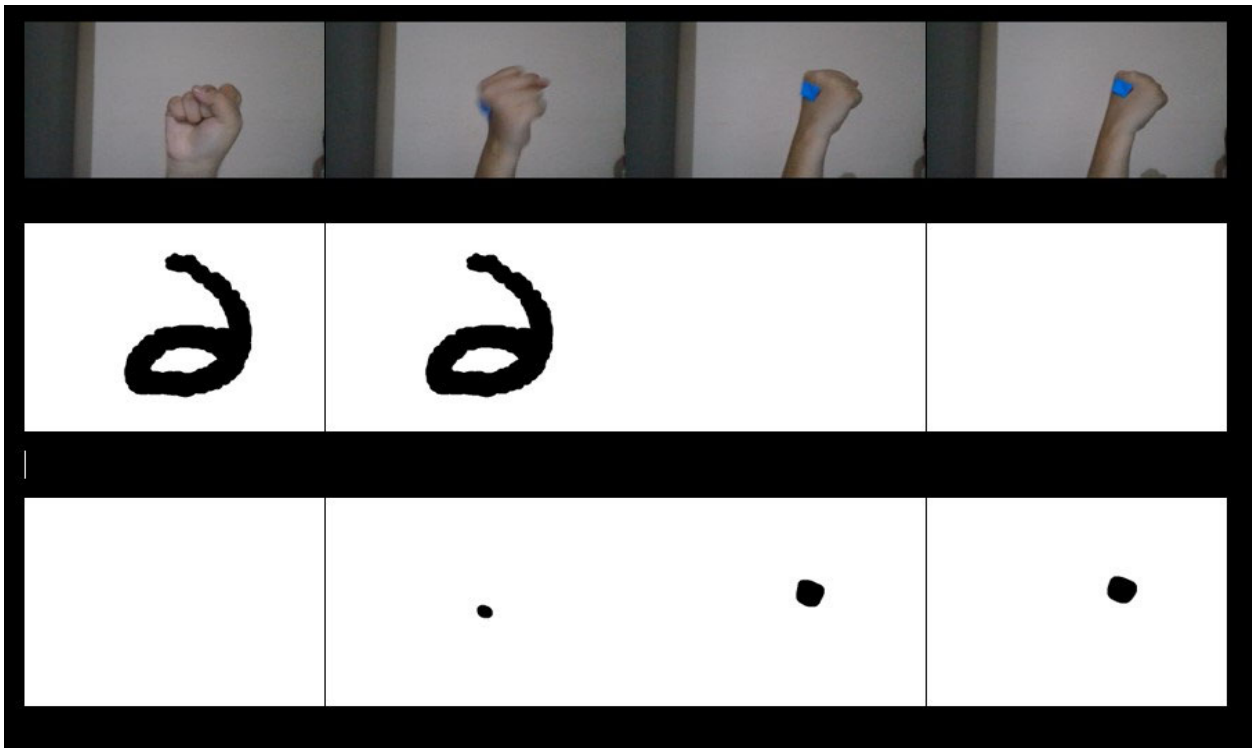


Fig: SHOWS THE ERASING OF THE RED MASK UPON DETECTION OF BLUE BLOB.

From the pictures we can see that the data obtained is flipped. That flipping occurs because the data is taken with the help of the web camera of the laptop and hence causes the mirrored image. Flipping of the image to get normal data is quite easy with the help of openCV.
Now we flip the image and resize the image so that the image can be fed into the trained CNN for classification or prediction. The frame size is 480X640, hence the masked image has the same size. The CNN is trained with images with a size of 28X28 images. So, we resize the masked image to 28X28.

Fig: DIGIT 1, FLIPPED DIGIT 1.

Fig: DIGIT 6, FLIPPED DIGIT 6.

The above images have a resolution of 480X640, we resize it using appropriate interpolation technique that doesn’t lose much of the information. Among the interpolation techniques we chose INTER_AREA[3][4] as it resamples using pixel area relation. This is the method that preserves quality of image after downsampling.

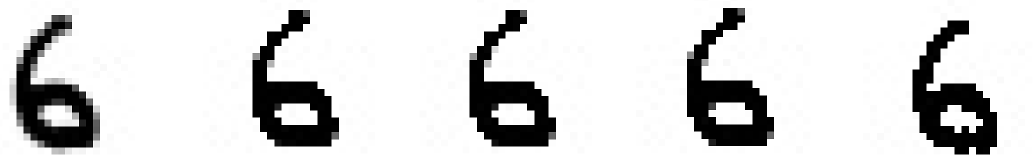
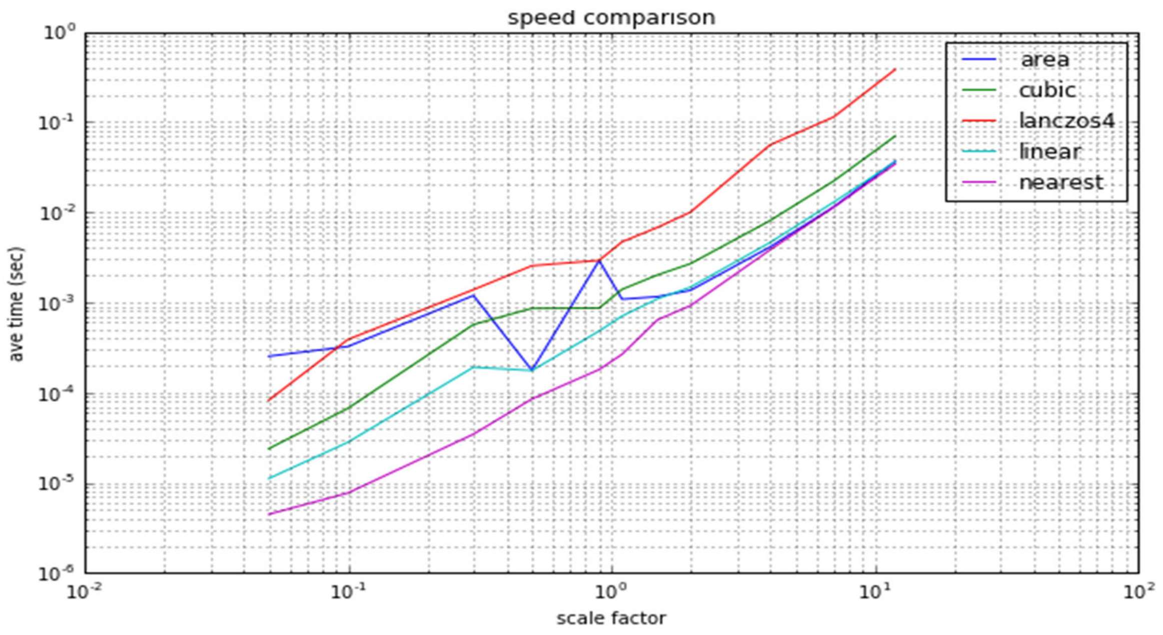


Fig: inter_area, inter_cubic, inter_lanczo4, inter_linear, inter_nearest respectively (size 28x28).



Now this image can be fed into CNN.

RECOGNITION OF DATA

Recognition of the data is very important and once the data is recognised we can then proceed to control the electronics. We have used Convolutional neural network for this step as CNNs are robust and the models can give predictions with high accuracy.

CONVOLUTIONAL NEURAL NETWORK[5]

In neural networks, convolutional neural networks is one of the main categories to do image recognition, image classification. Object detections, recognition of faces are some of the areas

[3] "cv2 resize interpolation methods - Chadrick's Blog." 14 Nov. 2018, https://chadrick-kwag.net/cv2-resize-interpolation-methods/. Accessed 19 Jul. 2020.
[4] "How do I choose an image interpolation method? (Emgu ...." 25 Jun. 2010, https://stackoverflow.com/questions/3112364/how-do-i-choose-an-image-interpolation-method-emgu-opencv. Accessed 19 Jul. 2020.
[5] "Understanding of Convolutional Neural Network (CNN) — Deep." 4 Mar. 2018, https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148. Accessed 19 Jul. 2020.

where CNNs are widely used. A CNN is a deep learning algorithm which can take in an image as input, assign importance to various aspects/ objects in the image and categories the images.

We prefer CNNs over feed forward neural networks because for image analysis feed forward neural networks can be overfitting and hence result in wrong outputs. Whereas CNNs can successfully capture the spatial and temporal dependencies in an image through the application of relevant filters or kernels.

Layers of CNN

1. Convolution layer
2. Pooling layer
3. Fully connected layer

CONVOLUTION LAYER

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel. Convolutional of an image gives a feature map as an output that reduces the size of the image and hence reducing the processing time of the algorithm without compromising the accuracy of the model. Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters.

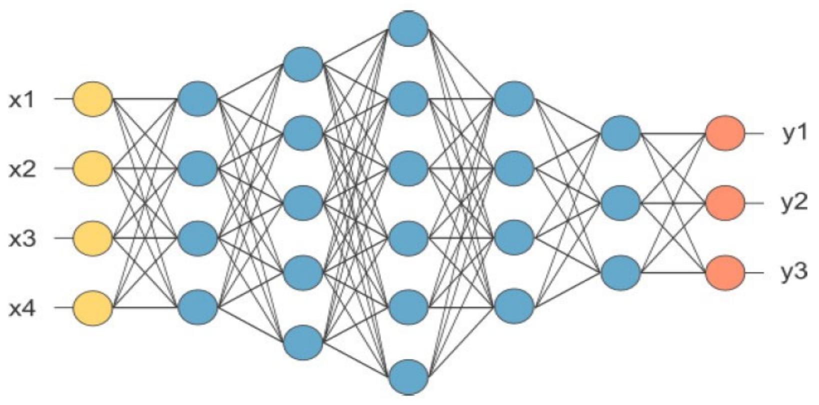
Convolution of image can give negative values and that is avoided with the help of rectified linear unit for a non linearity operation (ReLU). The output of this operation is max(0,x), where x represents the pixel value in consideration. The performance of this function is better compared to its counterparts, tanh or sigmoid.

POOLING LAYER

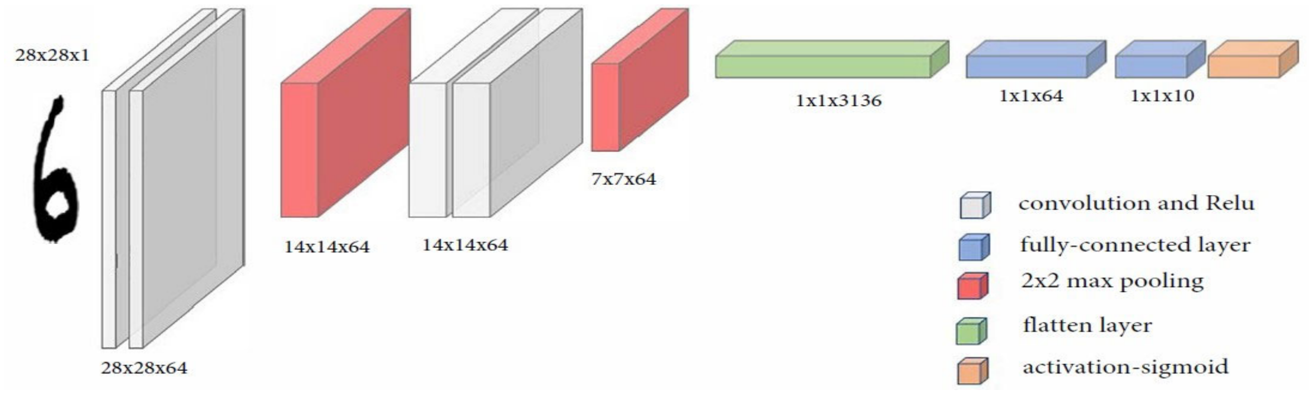
Pooling layer helps in the reduction of the number of parameters when the image is too large. There are three types of pooling. *Max pooling,* we take the largest element from the rectified feature map. *Average pooling,* we take the average of the elements of the specified size of the matrix. *Sum pooling,* we take the sum of the elements of the said matrix.

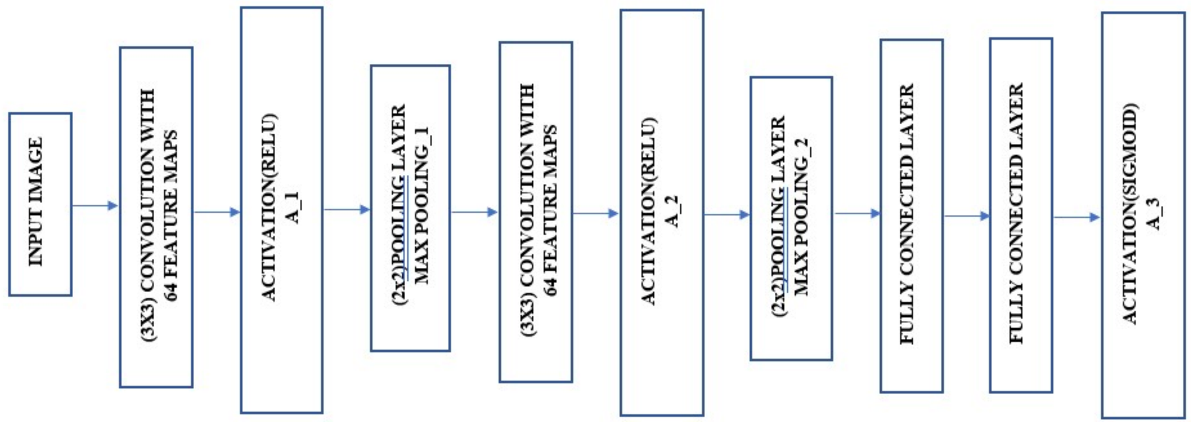
FULLY CONNECTED LAYER

We flatten our matrix into a vector and feed it into a fully connected layer like a neural network. With fully connected layers, we combine features to create a model. We finally have an activation function such as softmax or sigmoid to classify the outputs into the respective categories.



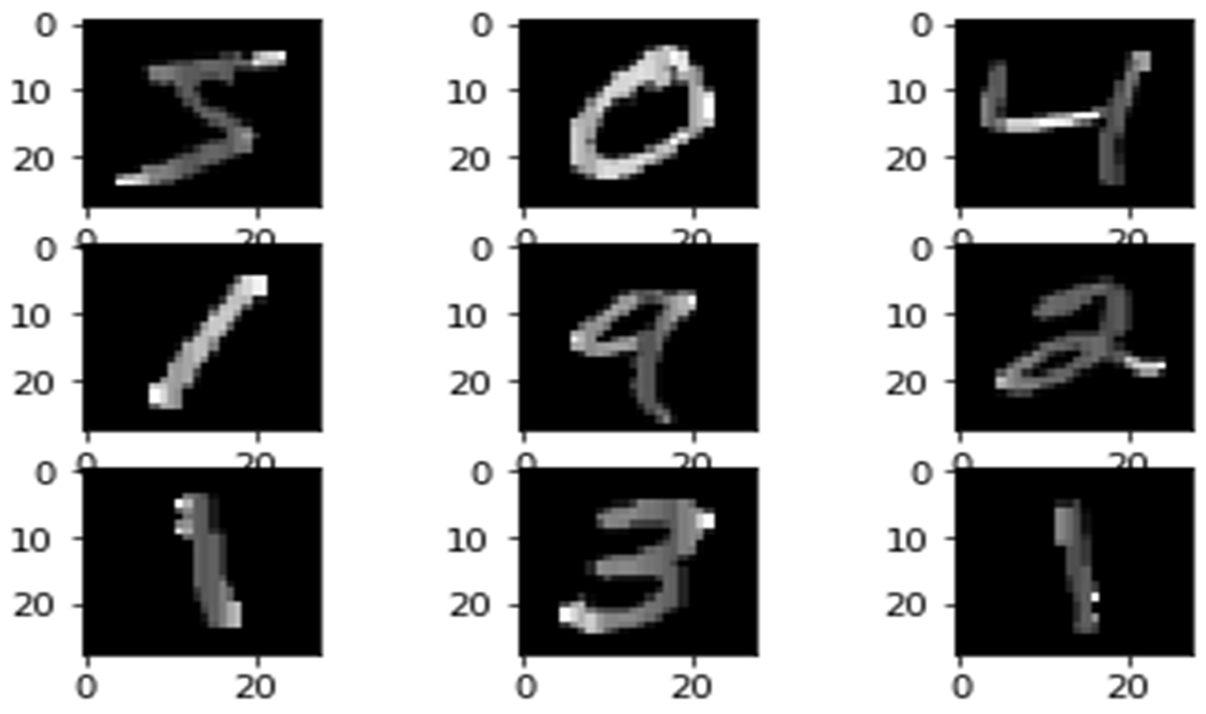
THE PROPOSED CNN ARCHITECTURE




INPUT IMAGE
(3X3) CONVOLUTION WITH 64 FEATURE MAPS
ACTIVATION(RELU) A_1
(2x2)POOLING LAYER MAX POOLING_1
(3X3) CONVOLUTION WITH 64 FEATURE MAPS
ACTIVATION(RELU) A_2
(2x2)POOLING LAYER MAX POOLING_2
FULLY CONNECTED LAYER
FULLY CONNECTED LAYER
ACTIVATION(SIGMOID) A_3

DATASET

It is very important to have a dataset for the proper training of the CNN model and for the prediction of the output for input from outside the training dataset. The dataset must include images of the categories that are different and yet classifiable. We chose MNIST dataset with 60,000 images training images of the digits from 0-9 in grayscale of size 28x28. Despite the dataset being with a small resolution the accuracy of 98% is achieved with CNN model on the training dataset.



CODE

We import all the required libraries, tensorflow is used to build CNN models and train it with the MNIST dataset[6]. The required layers are also imported. Matplotlib helps in the visualization of the data and in our project to see the transformations on the image. Numpy is a general purpose array processing package. OpenCV is an open source computer vision and machine learning software library.

```
##import all the required libraries
import tensorflow as tf
import tensorflow.keras.callbacks as TensorBoard
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation,
Flatten, Conv2D, MaxPooling2D
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

We load the dataset into mnist and load the data into appropriate variables.

```
##loading the data for the training
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test,y_test) = mnist.load_data()
```

[6] "MNIST Data: A real-life test - Anmol Chawla - Medium." 17 Nov. 2018, https://medium.com/@anmol.chawlatrojan/mnsit-data-a-real-life-test-83f220e5422a. Accessed 19 Jul. 2020.

```
##printing the dimensions of the array containing the data for the
training
print("training data shape is {} ".format(x_train.shape))
print("train label data shape is {} ".format(y_train.shape))
print("test data shape is {} ".format(x_test.shape))
print("test label data shape is {} ".format(y_test.shape))
```

```
training data shape is (60000, 28, 28)
train label data shape is (60000,)
test data shape is (10000, 28, 28)
test label data shape is (10000,)
```

We need to take care that the data being used to train the CNN is made appropriate for the model. From the output we can say that they are 60000 images in the dataset and the resolution of the image is 28X28.

```
##preparing the data for the training
x_train = tf.keras.utils.normalize(x_train,axis=1)
#values are scaled between 0 and 1
x_test = tf.keras.utils.normalize(x_test,axis=1)
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
#reshaping the data to fit the CNN model
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
print("training data shape is {} ".format(x_train.shape))
```

```
training data shape is (60000, 28, 28, 1)
```

This the model of CNN used in the project. The layers are kept one after the other with the help of sequential. The first convolution layer has 64 filters with a size of 3X3 that are trained with the input image. We use the activation of relu because that is known to give optimum results. We do maxpooling of size 2X2 to reduce the spatial dimensions. The second convolution layer is similar to the first and has the same parameters. After that we flatten the obtained images and feed it into a 64 nodes fully connected layer that proceeds to connect to 10 nodes since the number of classes is 10 (0-9 digits in the dataset). Fully connected layer has sigmoid as the activation function. *Adam optimizer* is one of the most commonly used optimizers in CNN, it is a stochastic gradient descent method that is based on adaptive estimation of first order and second order moments. We are using *sparse categorical cross entropy,* which implies that we are taking loss considering the categories of classification and sparse refers to using a single integer from zero to the number of classes. Then we fit the dataset to the model with a size of batch equal to 120 and with 3 epochs we get the accuracy around 98%.

```
##the model
input_shape = (28, 28, 1)
model = Sequential()
```

```
model.add(Conv2D(64,(3,3),input_shape = input_shape))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Dense(10))
model.add(Activation('sigmoid'))

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

history=model.fit(x_train,y_train,validation_split=0.1,batch_size=120
,epochs=10)
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 45s 841us/sample - loss: 0.2826 - accuracy: 0.9183
- val_loss: 0.0897 - val_accuracy: 0.9727
Epoch 2/10
54000/54000 [==============================] - 47s 871us/sample - loss: 0.0837 - accuracy: 0.9742
- val_loss: 0.0581 - val_accuracy: 0.9837
Epoch 3/10
54000/54000 [==============================] - 46s 848us/sample - loss: 0.0610 - accuracy: 0.9813
- val_loss: 0.0585 - val_accuracy: 0.9838
Epoch 4/10
54000/54000 [==============================] - 45s 827us/sample - loss: 0.0502 - accuracy: 0.9845
- val_loss: 0.0544 - val_accuracy: 0.9850
Epoch 5/10
54000/54000 [==============================] - 48s 886us/sample - loss: 0.0402 - accuracy: 0.9876
- val_loss: 0.0552 - val_accuracy: 0.9845
Epoch 6/10
54000/54000 [==============================] - 47s 863us/sample - loss: 0.0333 - accuracy: 0.9898
- val_loss: 0.0459 - val_accuracy: 0.9858
Epoch 7/10
54000/54000 [==============================] - 45s 828us/sample - loss: 0.0289 - accuracy: 0.9906
- val_loss: 0.0429 - val_accuracy: 0.9867
Epoch 8/10
54000/54000 [==============================] - 45s 836us/sample - loss: 0.0233 - accuracy: 0.9924
- val_loss: 0.0394 - val_accuracy: 0.9892
Epoch 9/10
54000/54000 [==============================] - 46s 855us/sample - loss: 0.0203 - accuracy: 0.9932
- val_loss: 0.0469 - val_accuracy: 0.9882
```

```
Epoch 10/10
54000/54000 [==============================] - 47s 862us/sample - loss: 0.0182 - accuracy: 0.9944
- val_loss: 0.0412 - val_accuracy: 0.9900
```

We are defining the upper and lower limits of the colors to be detected in HSV as it is easier to express in this format than in RGB values. Using opencv we access the video file.

```
##the detection
lower_red = np.array([150,150,120])
upper_red = np.array([180,255,255])

lower_blue = np.array([90,150,120])
upper_blue = np.array([110,255,255])

cap = cv2.VideoCapture('test4.avi')
ret1, frame = cap.read()
```

We are taking the HSV of the first frame and using the mask of the frame to store the data.

```
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
mask1 = cv2.inRange(hsv, lower_red, upper_red)
ret2, mask1 = cv2.threshold(mask1, 120, 255, cv2.THRESH_BINARY_INV)
r,c = np.shape(mask1)
print(r,c)
prevblobs = 0
i = 0
masks = np.zeros((10,r,c))
```

```
480 640
```

This involves the detection of the red and blue color and the tracking of the red color for data points and blue color to erase the mask. The number of iterations of erosion and dilations done was chosen through trial and error. Medianblur was used because it gave appropriate results and with minimum requirement of the area size we detect the contours in the masked images. Once we detect the contours we can detect the number of blobs in the masked images. Once we get the keypoints in the detected contour we then can indicate the detected contour in the frame with the help of cv2.drawKeypoints. We do logic bitwise and to track the red color and store the data into a new variable upon the detection of blue color and erase the mask.

```
##the tracking of red color on the finger
while True:
    ret1, frame1 = cap.read()

    if ret1:
        hsv = cv2.cvtColor(frame1, cv2.COLOR_BGR2HSV)
        mask2 = cv2.inRange(hsv, lower_red, upper_red)
        mask3 = cv2.inRange(hsv, lower_blue, upper_blue)

        ret2, mask2 = cv2.threshold(mask2, 120, 255,
cv2.THRESH_BINARY_INV)
        ret2, mask3 = cv2.threshold(mask3, 120, 255,
cv2.THRESH_BINARY_INV)

        mask2 = cv2.erode(mask2, None, iterations = 12)
        mask2 = cv2.dilate(mask2, None, iterations = 3)
        mask2 = cv2.medianBlur(mask2,15)

        mask3 = cv2.erode(mask3, None, iterations = 12)
        mask3 = cv2.dilate(mask3, None, iterations = 3)
        mask3 = cv2.medianBlur(mask3,15)

        params = cv2.SimpleBlobDetector_Params()
        params.minThreshold = 100
        params.maxThreshold = 150
        params.filterByArea = True
        params.minArea = 700

   detector = cv2.SimpleBlobDetector_create(params)

        keypoints = detector.detect(mask2)
        keypoints1 = detector.detect(mask3)
        blobs = len(keypoints1)
        #print(blobs)

        imgKeyPoints = cv2.drawKeypoints(frame1, keypoints,
np.array([]),
(0,0,255),cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

        mask1 = cv2.bitwise_and(mask1,mask2);

        cv2.imshow('mask2', mask2)
        cv2.imshow('mask1',mask1)
        cv2.imshow('frame', imgKeyPoints)
        #print("blobs",blobs)
        #print("prevblobs",prevblobs)
        z = np.sum(mask1==0)
        if blobs-prevblobs >= 1 and z>5000:
            print("done")
            masks[i][:][:] = mask1
            mask1 = mask1*0 + 255
            i=i+1
        prevblobs = blobs
        k = cv2.waitKey(5) & 0xFF
        if k == 27:
            break

    else:
        break
cv2.destroyAllWindows()
```

```
cap.release()
```

```
done
done
done
done
```

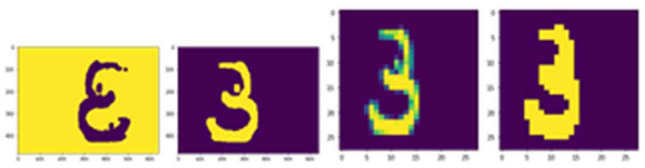
Setting the number of iterations required to make proper predictions.

```
a = np.zeros(i)
i=i-1
j=i
k=i
print(i)
```

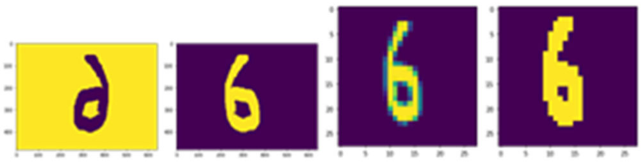
```
3
```

We resize and make the image compatible with the input size of the model, normalize the dataset and get a prediction.

```
while i>=0:
    img = masks[i][:][:]
    plt.imshow(img)
    plt.show()
    ret2, img1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
    img1 = cv2.flip(img1,1)
    img1 = cv2.erode(img1, None, iterations = 10)
    img1 = cv2.dilate(img1, None, iterations = 10)
    img2 = cv2.resize(img1,(28,28),interpolation = cv2.INTER_AREA)
    img2 = img2>0
    print(np.shape(img2))
    plt.imshow(img2)
    plt.show()
    img2 = img2>0
    plt.imshow(img2)
    plt.show()
    test1 = tf.keras.utils.normalize(img2,axis=1) #making the image
fit the CNN
    test1 = test1.reshape(1,28,28,1)
    predictions = model.predict(test1)
    print("Prediction: "),
    print(np.argmax(predictions))
    a[i] = np.argmax(predictions)
    i = i-1
```

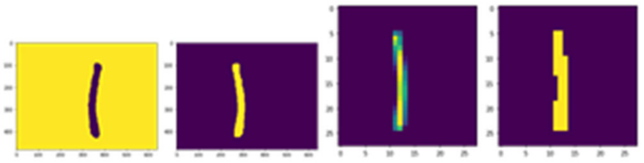


Prediction: 3



Prediction: 6

Prediction: 2



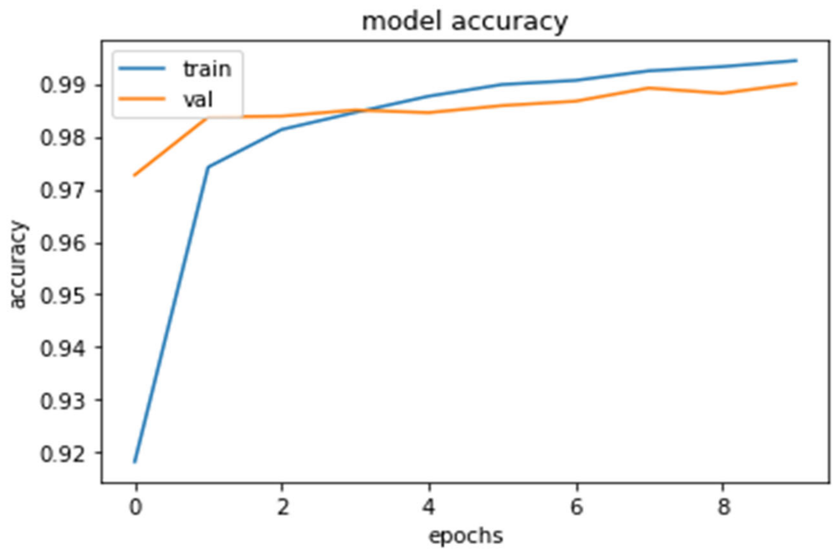
Prediction: 1

CHAPTER 3

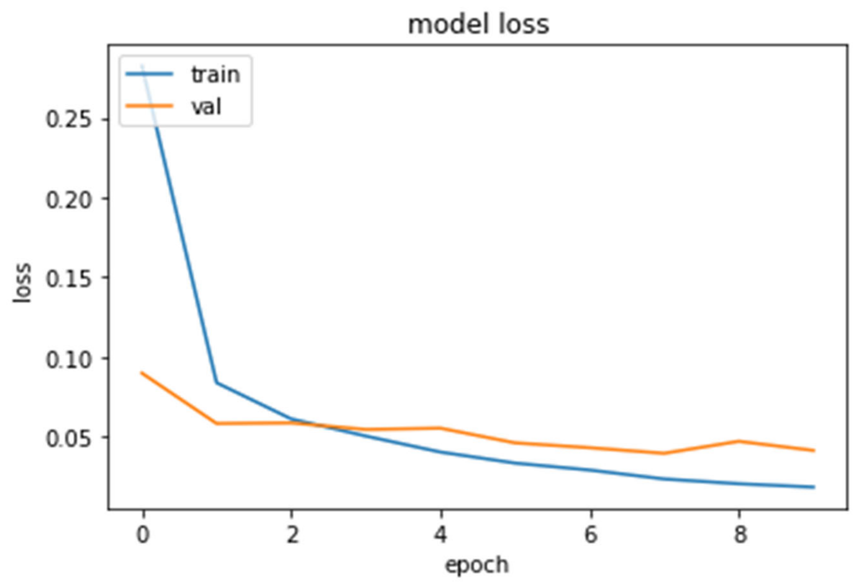
RESULT/ FUTURE SCOPE AND REFERENCES

RESULT

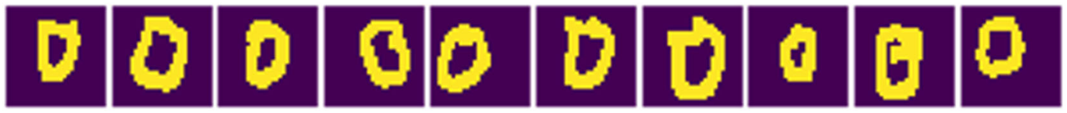
The plot of training dataset accuracy and validation accuracy with each epoch.



The plot of training dataset loss and validation loss with each epoch.



DIGIT 0



Prediction value:[0,0,0,0,0,0,0,0,8,9]

For the given inputs, accuracy of prediction = 8/10 = 0.8

DIGIT 1

Prediction values:[1,1,1,1,1,1,1,1,1]

For the given inputs, accuracy of prediction = 10/10 = 1

DIGIT 2

Prediction values:[2,2,2,2,2,2,2,2,2,2]

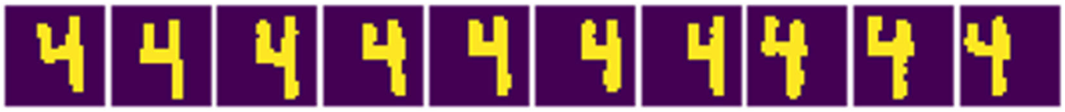
For the given inputs, accuracy of prediction = 10/10 = 1

DIGIT 3

Prediction values:[3,3,3,3,3,3,3,3,3,3]

For the given inputs, accuracy of prediction = 10/10 = 1

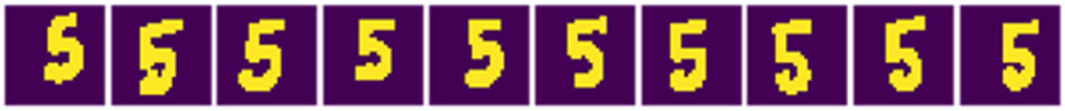
DIGIT 4



Prediction values:[4,4,4,4,4,4,4,8,8,6]

For the given inputs, accuracy of prediction = 7/10 = 0.7

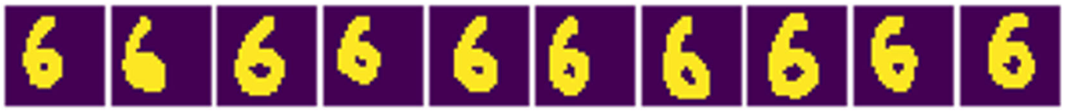
DIGIT 5



Prediction values:[5,5,5,5,5,5,5,5,5,5]

For the given inputs, accuracy of prediction = 10/10 = 1

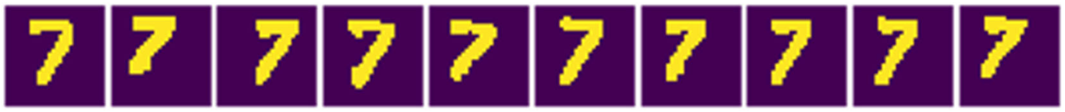
DIGIT 6



Prediction values:[6,6,6,6,6,6,6,6,6,6]

For the given inputs, accuracy of prediction = 10/10 = 1

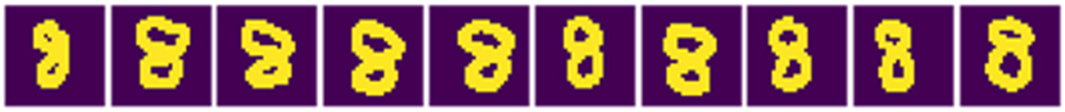
DIGIT 7



Prediction values:[7,7,7,7,7,7,7,7,7,7]

For the given inputs, accuracy of prediction = 10/10 = 1

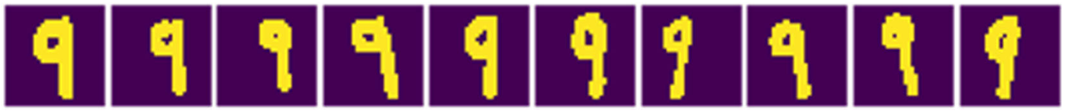
DIGIT 8



Prediction values:[8,8,8,8,8,8,8,8,8,0]

For the given inputs, accuracy of prediction = 9/10 = 0.9

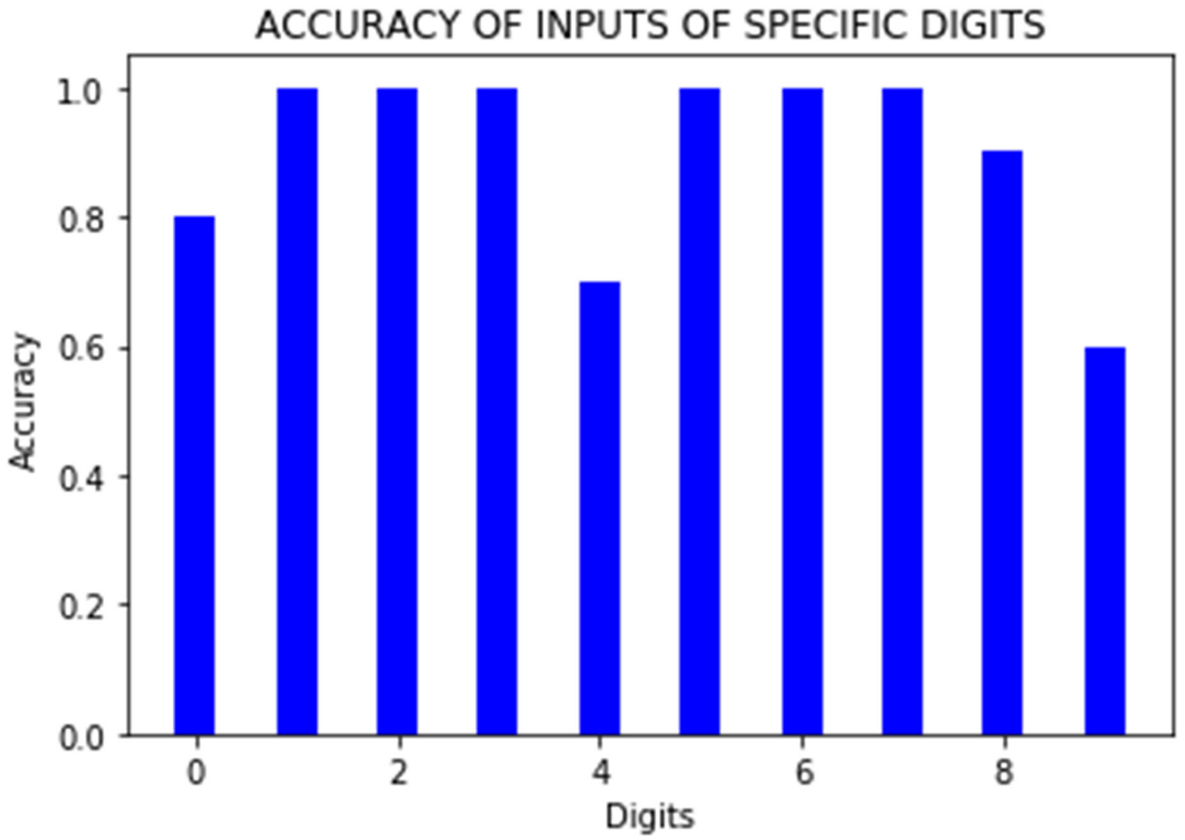
DIGIT 9



Prediction values:[9,9,9,9,9,9,4,8,1,8]

For the given inputs, accuracy of prediction = 6/10 = 0.6

Accuracy plot of above dataset.



FUTURE SCOPE

1. The project can be further improved by tracking the finger tip without any color specific detection.
2. The dataset used in CNN can be augmented data to increase the accuracy in the test cases.
3. The detection of multiple digits/alphabets can be achieved with the implementation of Optical Character Recognition (OCR).

REFERENCES

Chawla A. 2018. MNIST Data: A real-life test. Medium [Internet]. Available from: https://medium.com/@anmol.chawlatrojan/mnsit-data-a-real-life-test-83f220e5422a

Hussain A, Rosebrock A, Jaffar, et al. 2020. Keras Conv2D and Convolutional Layers. PyImageSearch [Internet]. Available from: https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/

Prabhu. 2019. Understanding of Convolutional Neural Network (CNN) - Deep Learning. Medium [Internet]. Available from: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

Squirrel A, Anonymous, Pedro, Federico, Tommy. 2018. cv2 resize interpolation methods. Chadrick's Blog [Internet]. Available from: https://chadrick-kwag.net/cv2-resize-interpolation-methods/

Tom WrightTom Wright 10.4k1212 gold badges6868 silver badges135135 bronze badges, MusiGenesisMusiGenesis 70.3k3636 gold badgets175175 silver badges318318 bronze badges, nbsrujannbsruja 98411 gold badge88 silver badges2525 bronze badges, Federico CristinaFederico Cristina 1, Martin BeckettMartin Beckett 87.7k2222 gold badges173173 silver badges247247 bronze badges. 1960. How do I choose an image interpolation method? (Emgu/OpenCV). Stack Overflow [Internet]. Available from: https://stackoverflow.com/questions/3112364/how-do-i-choose-an-image-interpolation-method-emgu-opencv

CHAPTER 4

PROGRAMME AND CODING OF PROJECT

```
##import all the required libraries
import tensorflow as tf
import tensorflow.keras.callbacks as TensorBoard
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D,
MaxPooling2D
import cv2
import numpy as np
import matplotlib.pyplot as plt

##loading the data for the training
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test,y_test) = mnist.load_data()

##printing the dimensions of the array containing the data for the training
print("training data shape is {} ".format(x_train.shape))
print("train label data shape is {} ".format(y_train.shape))
print("test data shape is {} ".format(x_test.shape))
print("test label data shape is {} ".format(y_test.shape))

##preparing the data for the training
x_train = tf.keras.utils.normalize(x_train,axis=1) #values are scaled between 0 and 1
x_test = tf.keras.utils.normalize(x_test,axis=1)
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1) #reshaping the data to fit the CNN model
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
print("training data shape is {} ".format(x_train.shape))

for i in range(0,9):
   plt.subplot(330+1+i)
   plt.imshow(x_train[i].reshape(28,28), cmap=plt.get_cmap('gray'))
plt.show()
##the model
input_shape = (28, 28, 1)
```

```
model = Sequential()
model.add(Conv2D(64,(3,3),input_shape = input_shape))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Dense(10))
model.add(Activation('sigmoid'))

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

history=model.fit(x_train,y_train,validation_split=0.1,batch_size=120,epochs=10)

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epochs')
plt.legend(['train','val'], loc='upper left')

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

##the detection
lower_red = np.array([150,150,120])
upper_red = np.array([180,255,255])

lower_blue = np.array([90,150,120])
```

```
upper_blue = np.array([110,255,255])

cap = cv2.VideoCapture('test4.avi')
ret1, frame = cap.read()

hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
mask1 = cv2.inRange(hsv, lower_red, upper_red)
ret2, mask1 = cv2.threshold(mask1, 120, 255, cv2.THRESH_BINARY_INV)
r,c = np.shape(mask1)
print(r,c)
prevblobs = 0
i = 0
masks = np.zeros((10,r,c))

##the tracking of red color on the finger
while True:
    ret1, frame1 = cap.read()

    if ret1:
        hsv = cv2.cvtColor(frame1, cv2.COLOR_BGR2HSV)
        mask2 = cv2.inRange(hsv, lower_red, upper_red)
        mask3 = cv2.inRange(hsv, lower_blue, upper_blue)

        ret2, mask2 = cv2.threshold(mask2, 120, 255, cv2.THRESH_BINARY_INV)
        ret2, mask3 = cv2.threshold(mask3, 120, 255, cv2.THRESH_BINARY_INV)

        mask2 = cv2.erode(mask2, None, iterations = 12)
        mask2 = cv2.dilate(mask2, None, iterations = 3)
        mask2 = cv2.medianBlur(mask2,15)

        mask3 = cv2.erode(mask3, None, iterations = 12)
        mask3 = cv2.dilate(mask3, None, iterations = 3)
        mask3 = cv2.medianBlur(mask3,15)

        params = cv2.SimpleBlobDetector_Params()
        params.minThreshold = 100
        params.maxThreshold = 150
        params.filterByArea = True
        params.minArea = 700
        detector = cv2.SimpleBlobDetector_create(params)

        keypoints = detector.detect(mask2)
        keypoints1 = detector.detect(mask3)
        blobs = len(keypoints1)
        #print(blobs)
        #contours, hierarchy =
cv2.findContours(mask2,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

        imgKeyPoints = cv2.drawKeypoints(frame1, keypoints, np.array([]),
(0,0,255),cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)


        mask1 = cv2.bitwise_and(mask1,mask2);

        cv2.imshow('mask2', mask2)
        cv2.imshow('mask1',mask1)
        cv2.imshow('frame', imgKeyPoints)
        #print("blobs",blobs)
        #print("prevblobs",prevblobs)
        z = np.sum(mask1==0)
        if blobs-prevblobs >= 1 and z>5000:
            print("done")
            masks[i][:][:] = mask1
            mask1 = mask1*0 + 255
            i=i+1
        prevblobs = blobs
        k = cv2.waitKey(5) & 0xFF
        if k == 27:
            break

    else:
        break
#cv2.imwrite('letter.png',mask1)
cv2.destroyAllWindows()
cap.release()

a = np.zeros(i)
i=i-1
j=i
```

```
k=i
print(i)

while i>=0:
    img = masks[i][:][:]
    plt.imshow(img)
    plt.show()
    ret2, img1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
    img1 = cv2.flip(img1,1)
    img1 = cv2.erode(img1, None, iterations = 10)
    #plt.imshow(img1)
    #plt.show()
    img1 = cv2.dilate(img1, None, iterations = 10)
    plt.imshow(img1)
    plt.show()
    #img2 = np.resize(img1,(28,28,1))
    img2 = cv2.resize(img1,(28,28),interpolation = cv2.INTER_AREA)
    #img2 = cv2.dilate(img2, None, iterations = 1)

    print(np.shape(img2))
    #fig.add_subplot(2,2,2)
    plt.imshow(img2)
    plt.show()
    img2 = img2>0
    plt.imshow(img2)
    plt.show()
    test1 = tf.keras.utils.normalize(img2,axis=1) #making the image fit the CNN
    test1 = test1.reshape(1,28,28,1)
    predictions = model.predict(test1)
    print("prediction: "),
    print(np.argmax(predictions))
    #print(np.argmax(predictions))
    a[i] = np.argmax(predictions)
    i = i-1
#calculating the accuracy and displaying the processed images and predicted value of 0-9 digits
j=0
while j<=9:
    img = cv2.cvtColor(cv2.imread('test1_'+str(j)+'.jpg'), cv2.COLOR_BGR2GRAY)
    ret2, img1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
    img1 = cv2.flip(img1,1)
    img1 = cv2.erode(img1, None, iterations = 10)
    img1 = cv2.dilate(img1, None, iterations = 10)
    img2 = cv2.resize(img1,(28,28),interpolation = cv2.INTER_AREA)
    img2 = img2>0
    plt.axis('off')
    plt.imshow(img2)
    plt.show()
    test1 = tf.keras.utils.normalize(img2,axis=1) #making the image fit the CNN
    test1 = test1.reshape(1,28,28,1)
    predictions = model.predict(test1)
    print("prediction: "),
    print(np.argmax(predictions))
    j=j+1
j=0
while j<=9:
    img = cv2.cvtColor(cv2.imread('test0_'+str(j)+'.jpg'), cv2.COLOR_BGR2GRAY)
    ret2, img1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
    img1 = cv2.flip(img1,1)
    img1 = cv2.erode(img1, None, iterations = 10)
    img1 = cv2.dilate(img1, None, iterations = 10)
    img2 = cv2.resize(img1,(28,28),interpolation = cv2.INTER_AREA)
    img2 = img2>0
    plt.axis('off')
    plt.imshow(img2)
    plt.show()
    test1 = tf.keras.utils.normalize(img2,axis=1) #making the image fit the CNN
    test1 = test1.reshape(1,28,28,1)
    predictions = model.predict(test1)
    print("prediction: "),
    print(np.argmax(predictions))
    j=j+1
j=0
while j<=9:
    img = cv2.cvtColor(cv2.imread('test2_'+str(j)+'.jpg'), cv2.COLOR_BGR2GRAY)
    ret2, img1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
    img1 = cv2.flip(img1,1)
    img1 = cv2.erode(img1, None, iterations = 10)
    img1 = cv2.dilate(img1, None, iterations = 10)
    img2 = cv2.resize(img1,(28,28),interpolation = cv2.INTER_AREA)
    img2 = img2>0
    plt.axis('off')
    plt.imshow(img2)
    plt.show()
    test1 = tf.keras.utils.normalize(img2,axis=1) #making the image fit the CNN
    test1 = test1.reshape(1,28,28,1)
    predictions = model.predict(test1)
    print("prediction: "),
    print(np.argmax(predictions))
    j=j+1
j=0
while j<=9:
    img = cv2.cvtColor(cv2.imread('test3_'+str(j)+'.jpg'), cv2.COLOR_BGR2GRAY)
    ret2, img1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
    img1 = cv2.flip(img1,1)
    img1 = cv2.erode(img1, None, iterations = 10)
    img1 = cv2.dilate(img1, None, iterations = 10)
    img2 = cv2.resize(img1,(28,28),interpolation = cv2.INTER_AREA)
    img2 = img2>0
    plt.axis('off')
    plt.imshow(img2)
    plt.show()
    test1 = tf.keras.utils.normalize(img2,axis=1) #making the image fit the CNN
    test1 = test1.reshape(1,28,28,1)
    predictions = model.predict(test1)
    print("prediction: "),
    print(np.argmax(predictions))
    j=j+1
j=0
while j<=9:
    img = cv2.cvtColor(cv2.imread('test4_'+str(j)+'.jpg'), cv2.COLOR_BGR2GRAY)
    ret2, img1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
    img1 = cv2.flip(img1,1)
    img1 = cv2.erode(img1, None, iterations = 10)
    img1 = cv2.dilate(img1, None, iterations = 10)
    img2 = cv2.resize(img1,(28,28),interpolation = cv2.INTER_AREA)
    img2 = img2>0
    plt.axis('off')
    plt.imshow(img2)
    plt.show()
    test1 = tf.keras.utils.normalize(img2,axis=1) #making the image fit the CNN
    test1 = test1.reshape(1,28,28,1)
    predictions = model.predict(test1)
    print("prediction: "),
    print(np.argmax(predictions))
    j=j+1
j=0
while j<=9:
    img = cv2.cvtColor(cv2.imread('test6_'+str(j)+'.jpg'), cv2.COLOR_BGR2GRAY)
    ret2, img1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
    img1 = cv2.flip(img1,1)
    img1 = cv2.erode(img1, None, iterations = 10)
    img1 = cv2.dilate(img1, None, iterations = 10)
    img2 = cv2.resize(img1,(28,28),interpolation = cv2.INTER_AREA)
    img2 = img2>0
    plt.axis('off')
    plt.imshow(img2)
    plt.show()
    test1 = tf.keras.utils.normalize(img2,axis=1) #making the image fit the CNN
    test1 = test1.reshape(1,28,28,1)
    predictions = model.predict(test1)
    print("prediction: "),
    print(np.argmax(predictions))
    j=j+1
j=0
while j<=9:
    img = cv2.cvtColor(cv2.imread('test5_'+str(j)+'.jpg'), cv2.COLOR_BGR2GRAY)
    ret2, img1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
    img1 = cv2.flip(img1,1)
    img1 = cv2.erode(img1, None, iterations = 10)
    img1 = cv2.dilate(img1, None, iterations = 10)
    img2 = cv2.resize(img1,(28,28),interpolation = cv2.INTER_AREA)
    img2 = img2>0
    plt.axis('off')
    plt.imshow(img2)
    plt.show()
    test1 = tf.keras.utils.normalize(img2,axis=1) #making the image fit the CNN
    test1 = test1.reshape(1,28,28,1)
    predictions = model.predict(test1)
    print("prediction: "),
    print(np.argmax(predictions))
    j=j+1
j=0
while j<=9:
    img = cv2.cvtColor(cv2.imread('test7_'+str(j)+'.jpg'), cv2.COLOR_BGR2GRAY)
    ret2, img1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
    img1 = cv2.flip(img1,1)
    img1 = cv2.erode(img1, None, iterations = 10)
    img1 = cv2.dilate(img1, None, iterations = 10)
    img2 = cv2.resize(img1,(28,28),interpolation = cv2.INTER_AREA)
    img2 = img2>0
    plt.axis('off')
    plt.imshow(img2)
    plt.show()
    test1 = tf.keras.utils.normalize(img2,axis=1) #making the image fit the CNN
    test1 = test1.reshape(1,28,28,1)
    predictions = model.predict(test1)
    print("prediction: "),
    print(np.argmax(predictions))
    j=j+1
j=0
while j<=9:
    img = cv2.cvtColor(cv2.imread('test9_'+str(j)+'.jpg'), cv2.COLOR_BGR2GRAY)
    ret2, img1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
    img1 = cv2.flip(img1,1)
    img1 = cv2.erode(img1, None, iterations = 10)
    img1 = cv2.dilate(img1, None, iterations = 10)
    img2 = cv2.resize(img1,(28,28),interpolation = cv2.INTER_AREA)
    img2 = img2>0
    plt.axis('off')
    plt.imshow(img2)
    plt.show()
    test1 = tf.keras.utils.normalize(img2,axis=1) #making the image fit the CNN
    test1 = test1.reshape(1,28,28,1)
    predictions = model.predict(test1)
    print("prediction: "),
    print(np.argmax(predictions))
    j=j+1
j=0
while j<=9:
    img = cv2.cvtColor(cv2.imread('test8_'+str(j)+'.jpg'), cv2.COLOR_BGR2GRAY)
    ret2, img1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
    img1 = cv2.flip(img1,1)
    img1 = cv2.erode(img1, None, iterations = 10)
    img1 = cv2.dilate(img1, None, iterations = 10)
    img2 = cv2.resize(img1,(28,28),interpolation = cv2.INTER_AREA)
    img2 = img2>0
    plt.axis('off')
    plt.imshow(img2)
    plt.show()
    test1 = tf.keras.utils.normalize(img2,axis=1) #making the image fit the CNN
    test1 = test1.reshape(1,28,28,1)
    predictions = model.predict(test1)
    print("prediction: "),
    print(np.argmax(predictions))
    j=j+1

i=0
accuracy_test = np.zeros(10,dtype=int)
while i<=9:
    j=0
    while j<=9:
        img = cv2.cvtColor(cv2.imread('test'+str(i)+'_'+str(j)+'.jpg'), cv2.COLOR_BGR2GRAY)
        ret2, img1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
        img1 = cv2.flip(img1,1)
        img1 = cv2.erode(img1, None, iterations = 10)
        img1 = cv2.dilate(img1, None, iterations = 10)
        img2 = cv2.resize(img1,(28,28),interpolation = cv2.INTER_AREA)
        img2 = img2>0
        test1 = tf.keras.utils.normalize(img2,axis=1) #making the image fit the CNN
        test1 = test1.reshape(1,28,28,1)
        predictions = model.predict(test1)
        pred = np.argmax(predictions)
        if pred == i:
            accuracy_test[i] = accuracy_test[i]+1
        j=j+1
    i=i+1

index = np.array([0,1,2,3,4,5,6,7,8,9])
accuracy_test = accuracy_test/10
plt.bar(index,accuracy_test,color='blue',width=0.4)
plt.xlabel('Digits')
plt.ylabel('Accuracy')
plt.title('ACCURACY OF INPUTS OF SPECIFIC DIGITS')
plt.show()
```