**APPENDIX**

**ImageGenerator.py**

import camera as camera  
import cv2  
import imutils  
import numpy as np  
  
bg = None  
  
def run\_avg(image, aWeight):  
 global bg  
 if bg is None:  
 bg = image.copy().astype("float")  
 return  
 cv2.accumulateWeighted(image, bg, aWeight)  
  
def segment(image, threshold=25):  
 global bg  
 diff = cv2.absdiff(bg.astype("uint8"), image)  
 thresholded = cv2.threshold(diff, threshold, 255, cv2.THRESH\_BINARY)[1]  
 cnts, hierarchy = cv2.findContours(thresholded.copy(), cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 if len(cnts) == 0:  
 return  
 else:  
 segmented = max(cnts, key=cv2.contourArea)  
 return (thresholded, segmented)  
  
def main():  
 aWeight = 0.5  
 camera = cv2.VideoCapture(0)  
 top, right, bottom, left = 100, 400, 300, 600  
 num\_frames = 0  
 image\_num = 0  
 start\_recording = False  
  
 while(True):  
 (grabbed, frame) = camera.read()  
 if (grabbed == True):  
 frame = imutils.resize(frame, width=700)  
 frame = cv2.flip(frame, 1)  
 clone = frame.copy()  
 (height, width) = frame.shape[:2]  
 roi = frame[top:bottom, right:left]  
 gray = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY)  
 gray = cv2.GaussianBlur(gray, (7, 7), 0)  
  
 if num\_frames < 30:  
 run\_avg(gray, aWeight)  
 print(num\_frames)  
 else:  
 hand = segment(gray)  
 if hand is not None:  
 (thresholded, segmented) = hand  
 cv2.drawContours(clone, [segmented + (right, top)], -1, (0, 0, 255))  
 if start\_recording:  
 cv2.imwrite("Dataset/YoImages/yo\_" + str(image\_num) + '.png', thresholded)  
 image\_num += 1  
 cv2.imshow("Thesholded", thresholded)  
 cv2.rectangle(clone, (left, top), (right, bottom), (0, 255, 0), 2)  
 num\_frames += 1  
 cv2.imshow("Video Feed", clone)  
 keypress = cv2.waitKey(1) & 0xFF  
 print(image\_num)  
  
 if keypress == ord("q") or image\_num > 999:  
 break  
 if keypress == ord("s"):  
 start\_recording = True  
  
 else:  
 print("Error, Check Camera")  
 break  
  
main()  
#camera.release()  
cv2.destroyAllWindows()

**Resize.py**

from PIL import Image  
  
def resizeImage(imageName):  
 basewidth = 100  
 img = Image.open(imageName)  
 wpercent = (basewidth/float(img.size[0]))  
 hsize = int((float(img.size[1])\*float(wpercent)))  
 img = img.resize((basewidth,89), Image.ANTIALIAS)  
 img.save(imageName)  
  
for i in range(0, 100):  
 resizeImage("C:/Users/ayroy/Downloads/Gesture Controlled Mouse Pointer/Dataset/YoTest/yo\_" + str(i) + '.png')

**Trainer.py**

import tensorflow as tf  
import tflearn  
from tflearn.layers.conv import conv\_2d, max\_pool\_2d  
from tflearn.layers.core import input\_data, dropout, fully\_connected  
from tflearn.layers.estimator import regression  
import numpy as np  
import cv2  
from sklearn.utils import shuffle  
  
# Load Images from Swing  
loadedImages = []  
for i in range(0, 1000):  
 image = cv2.imread('Dataset/SwingImages/swing\_' + str(i) + '.png')  
 gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 loadedImages.append(gray\_image.reshape(89, 100, 1))  
  
# Load Images From Palm  
for i in range(0, 1000):  
 image = cv2.imread('Dataset/PalmImages/palm\_' + str(i) + '.png')  
 gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 loadedImages.append(gray\_image.reshape(89, 100, 1))  
  
# Load Images From Fist  
for i in range(0, 1000):  
 image = cv2.imread('Dataset/FistImages/fist\_' + str(i) + '.png')  
 gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 loadedImages.append(gray\_image.reshape(89, 100, 1))  
  
#Load Images From Peace  
for i in range(0, 1000):  
 image = cv2.imread('Dataset/PeaceImages/peace\_' + str(i) + '.png')  
 gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 loadedImages.append(gray\_image.reshape(89, 100, 1))  
  
#Load Images From ThumbsUp  
for i in range(0, 1000):  
 image = cv2.imread('Dataset/TriImages/tri\_' + str(i) + '.png')  
 gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 loadedImages.append(gray\_image.reshape(89, 100, 1))  
  
#Load Images From ThumbsDown  
for i in range(0, 1000):  
 image = cv2.imread('Dataset/YoImages/yo\_' + str(i) + '.png')  
 gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 loadedImages.append(gray\_image.reshape(89, 100, 1))  
  
# Create OutputVector  
outputVectors = []  
for i in range(0, 1000):  
 outputVectors.append([1, 0, 0, 0, 0, 0])  
  
for i in range(0, 1000):  
 outputVectors.append([0, 1, 0, 0, 0, 0])  
  
for i in range(0, 1000):  
 outputVectors.append([0, 0, 1, 0, 0, 0])  
  
for i in range(0, 1000):  
 outputVectors.append([0, 0, 0, 1, 0, 0])  
  
for i in range(0, 1000):  
 outputVectors.append([0, 0, 0, 0, 1, 0])  
  
for i in range(0, 1000):  
 outputVectors.append([0, 0, 0, 0, 0, 1])  
  
testImages = []  
  
for i in range(0, 100):  
 image = cv2.imread('Dataset/SwingTest/swing\_' + str(i) + '.png')  
 gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 testImages.append(gray\_image.reshape(89, 100, 1))  
  
for i in range(0, 100):  
 image = cv2.imread('Dataset/PalmTest/palm\_' + str(i) + '.png')  
 gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 testImages.append(gray\_image.reshape(89, 100, 1))  
  
for i in range(0, 100):  
 image = cv2.imread('Dataset/FistTest/fist\_' + str(i) + '.png')  
 gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 testImages.append(gray\_image.reshape(89, 100, 1))  
  
for i in range(0, 100):  
 image = cv2.imread('Dataset/PeaceTest/peace\_' + str(i) + '.png')  
 gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 testImages.append(gray\_image.reshape(89, 100, 1))  
  
for i in range(0, 100):  
 image = cv2.imread('Dataset/TriTest/tri\_' + str(i) + '.png')  
 gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 testImages.append(gray\_image.reshape(89, 100, 1))  
  
for i in range(0, 100):  
 image = cv2.imread('Dataset/YoTest/yo\_' + str(i) + '.png')  
 gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 testImages.append(gray\_image.reshape(89, 100, 1))  
  
testLabels = []  
  
for i in range(0, 100):  
 testLabels.append([1, 0, 0, 0, 0, 0])  
  
for i in range(0, 100):  
 testLabels.append([0, 1, 0, 0, 0, 0])  
  
for i in range(0, 100):  
 testLabels.append([0, 0, 1, 0, 0, 0])  
  
for i in range(0, 100):  
 testLabels.append([0, 0, 0, 1, 0, 0])  
  
for i in range(0, 100):  
 testLabels.append([0, 0, 0, 0, 1, 0])  
  
for i in range(0, 100):  
 testLabels.append([0, 0, 0, 0, 1, 0])  
  
tf.reset\_default\_graph()  
  
convnet = input\_data(shape=[None, 89, 100, 1], name='input')  
convnet = conv\_2d(convnet, 32, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
convnet = conv\_2d(convnet, 64, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
convnet = conv\_2d(convnet, 128, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
convnet = conv\_2d(convnet, 256, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
convnet = conv\_2d(convnet, 256, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
convnet = conv\_2d(convnet, 128, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
convnet = conv\_2d(convnet, 64, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
convnet = fully\_connected(convnet, 1000, activation='relu')  
convnet = dropout(convnet, 0.75)  
convnet = fully\_connected(convnet, 6, activation='softmax')  
convnet = regression(convnet, optimizer='adam', learning\_rate=0.001, loss='categorical\_crossentropy', name='regression')  
model = tflearn.DNN(convnet, tensorboard\_verbose=0)  
loadedImages, outputVectors = shuffle(loadedImages, outputVectors, random\_state=0)  
model.fit(loadedImages, outputVectors, n\_epoch=50, validation\_set=(testImages, testLabels),snapshot\_step=100, show\_metric=True, run\_id='convnet\_coursera')  
model.save("TrainedNewModel/GestureRecogModel.tfl")

**Main.py**

import tensorflow as tf  
import tflearn  
from tflearn.layers.conv import conv\_2d, max\_pool\_2d  
from tflearn.layers.core import input\_data, dropout, fully\_connected  
from tflearn.layers.estimator import regression  
import numpy as np  
from PIL import Image  
import cv2  
import imutils  
import pyautogui  
  
  
bg = None  
n=0  
cX=0  
cY=0  
nX=0  
nY=0  
i=0  
  
def resizeImage(imageName):  
 basewidth = 100  
 img = Image.open(imageName)  
 wpercent = (basewidth / float(img.size[0]))  
 hsize = int((float(img.size[1]) \* float(wpercent)))  
 img = img.resize((basewidth, hsize), Image.ANTIALIAS)  
 img.save(imageName)  
  
  
def run\_avg(image, aWeight):  
 global bg  
 if bg is None:  
 bg = image.copy().astype("float")  
 return  
 cv2.accumulateWeighted(image, bg, aWeight)  
  
def segment(image, threshold=25):  
 global bg  
 diff = cv2.absdiff(bg.astype("uint8"), image)  
 thresholded = cv2.threshold(diff, threshold, 255, cv2.THRESH\_BINARY)[1]  
 (cnts, \_) = cv2.findContours(thresholded.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 if len(cnts) == 0:  
 return  
 else:  
 segmented = max(cnts, key=cv2.contourArea)  
 return (thresholded, segmented)  
  
def main():  
 global cX,cY,nX,nY  
 aWeight = 0.5  
 camera = cv2.VideoCapture(0)  
 top, right, bottom, left = 110, 350, 325, 590  
 num\_frames = 0  
 start\_recording = False  
 n=0  
 while (True):  
 (grabbed, frame) = camera.read()  
 frame = imutils.resize(frame, width=700)  
 frame = cv2.flip(frame, 1)  
 clone = frame.copy()  
 (height, width) = frame.shape[:2]  
  
 roi = frame[top:bottom, right:left]  
 gray = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY)  
  
 gray = cv2.GaussianBlur(gray, (7, 7), 0)  
  
 if num\_frames < 30:  
 run\_avg(gray, aWeight)  
 else:  
 hand = segment(gray)  
  
 if hand is not None:  
 (thresholded, segmented) = hand  
 cv2.drawContours(clone, [segmented + (right, top)], -1, (0, 0, 255))  
 try:  
 M = cv2.moments(segmented + (right, top))  
 cX = int(M["m10"] / M["m00"])  
 cY = int(M["m01"] / M["m00"])  
 if nX == 0 and nY == 0 :  
 nX=cX  
 nY=cY  
 cv2.circle(clone, (cX, cY), 3, (255, 255, 255), -1)  
 #print(str(cX) + " " + str(cY))  
  
 except:  
 print("Empty")  
  
 if start\_recording:  
 cv2.imwrite('Temp.png', thresholded)  
 resizeImage('Temp.png')  
 predictedClass, confidence = getPredictedClass()  
 showStatistics(predictedClass, confidence)  
 cv2.imshow("Thesholded", thresholded)  
  
 cv2.rectangle(clone, (left, top), (right, bottom), (0, 255, 0), 2)  
 cv2.rectangle(clone, (375,215), (565,305), (255,0,0), 1)  
 num\_frames += 1  
 cv2.imshow("Video Feed", clone)  
 keypress = cv2.waitKey(1) & 0xFF  
  
 if keypress == ord("q"):  
 break  
 if keypress == ord("s"):  
 start\_recording = True  
  
def getPredictedClass():  
 image = cv2.imread('Temp.png')  
 gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 prediction = model.predict([gray\_image.reshape(89, 100, 1)])  
 return np.argmax(prediction), (np.amax(prediction) / (prediction[0][0] + prediction[0][1] + prediction[0][2] + prediction[0][3] + prediction[0][4] + prediction[0][5]))  
  
  
def showStatistics(predictedClass, confidence):  
 global n,cX,cY,nX,nY,i  
 pyautogui.FAILSAFE = False  
 textImage = np.zeros((300, 512, 3), np.uint8)  
 className = ""  
  
 if predictedClass == 0:  
 className = "Scroll Down - Swing"  
 if i==8:  
 i = 0  
 else:  
 pyautogui.scroll(-10)  
 n = 1  
 i=0  
  
 elif predictedClass == 1:  
 className = "Right Click - Palm"  
 if n != 2:  
 pyautogui.click(button='right')  
 n = 2  
 i = 0  
  
 elif predictedClass == 2:  
 className = "Mouse Movement - Fist"  
  
 if i<8:  
 pyautogui.move((cX-nX)\*10,(cY-nY)\*10)  
 if (nX < 375):  
 pyautogui.move(-30, 0)  
 if (nX > 565):  
 pyautogui.move(30, 0)  
 if (nY < 215):  
 pyautogui.move(0, -30)  
 if (nY > 305):  
 pyautogui.move(0, 30)  
  
 if abs(cX-nX)<2 and abs(cY-nY)<5:  
 i=i+1  
 else:  
 i=0  
 n = 3  
  
 elif predictedClass == 3:  
 className = "Left Click - Peace"  
 if n != 4:  
 pyautogui.click()  
 n = 4  
 i = 0  
  
 elif predictedClass == 4:  
 className = "Double Click - Three Finger"  
 if n!=5:  
 pyautogui.doubleClick()  
 n = 5  
 i = 0  
  
 elif predictedClass == 5:  
 className = "Scroll Up - Yo"  
 pyautogui.scroll(10)  
 n = 6  
 i = 0  
 nX=cX  
 nY=cY  
 print(className)  
  
 cv2.putText(textImage, "Gesture : " + className, (30, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2)  
 cv2.putText(textImage, "Precision : " + str(confidence \* 100) + '%', (30, 100), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2)  
 cv2.imshow("Statistics", textImage)  
  
tf.reset\_default\_graph()  
  
convnet = input\_data(shape=[None, 89, 100, 1], name='input')  
convnet = conv\_2d(convnet, 32, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
convnet = conv\_2d(convnet, 64, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
convnet = conv\_2d(convnet, 128, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
convnet = conv\_2d(convnet, 256, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
convnet = conv\_2d(convnet, 256, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
convnet = conv\_2d(convnet, 128, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
convnet = conv\_2d(convnet, 64, 2, activation='relu')  
convnet = max\_pool\_2d(convnet, 2)  
convnet = fully\_connected(convnet, 1000, activation='relu')  
convnet = dropout(convnet, 0.75)  
convnet = fully\_connected(convnet, 6, activation='softmax')  
convnet = regression(convnet, optimizer='adam', learning\_rate=0.001, loss='categorical\_crossentropy', name='regression')  
model = tflearn.DNN(convnet, tensorboard\_verbose=0)  
model.load("TrainedNewModel/GestureRecogModel.tfl")  
  
main()