Neural Network Gesture Recognition
Final Report
David Qin

Topic

In this project, I will build a neural network and train it using videos of people performing three closely related actions:

1. Finger drumming. (Strumming one's fingers on a table or a leg passes the time and, like other repetitive behavior, soothes. In professional settings you see this as people wait for someone to show up or finish talking. It is a way of saying, "Come on, let's get things moving here.")
2. Hands in pocket. (Many people are comforted by placing one or both hands in their pockets while talking to others. But sometimes this is seen as too informal and in some cultures is considered rude.)
3. Erratic arm and hand motions. (Sometimes we are confronted by an individual making erratic motions with the arms and hands. The arms and hands might be out of synchrony with the rest of the body and with the person's surroundings. In these instances, the best we can do is recognize that there may be a mental condition or disorder at play. Recognition and understanding are key to lending assistance if necessary.)

Not only will my model be able to detect between these three actions, they must also be able to work within a “Smart home” environment. This means the model must also be able to differentiate between all sorts of possible conditions that may occur with a household. There are going to be many common cases, such has having to deal with recognizing an action with only the persons backside. This could also include lighting which will vary throughout the day. However, there are harder cases that we need to deal with, such as possible obstruction due to furniture.

This project is done on Google Colab as it was easy to develop on as well as boasting some great performance when training. My final project will use a python script that will take command line arguments in order to run and guess. It will output a graph with timeline predictions as well as a json that contains the prediction value over time for each action.

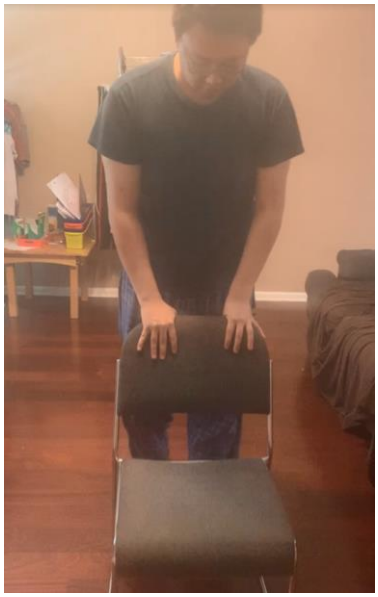
Dataset

For the three actions, I haven't had much luck finding good datasets that already had these actions, these were a bit unique. As a result, I focused on making my own personal dataset. At the start, my focus was to just cover the simplest case: mostly from the front with the entire body in the frame. I had about 15-20 videos of each action, totaling to about 55 short videos total. Occasionally, I'd change the angle my body was relative to the camera. Here are some example frames of the three actions:

Since the first and second lab, I now have others participating in performing these actions in order to make the data look more diverse since everyone performed these actions slightly differently. From this point on, I focused on how to make these more applicable to a smart home. For example, I would obstruct my body with furniture since it's common to have objects obstruct a camera's views in a house. Another thing I'd do was to perform them in a dark closet since lighting could be different. The actions have some variation too. For the erratic arm movements, these are usually related to "mental issues" and so in one example I threw around objects since the action is in a very similar category. Finger drumming was also done on different surfaces, whit the users in different positions too. Examples:

I have also neglected to include a negative dataset in the previous iterations. For this one, I have created more videos of myself performing some other common household actions. This could include walking, stretching, and opening/closing doors. This set is smaller than my other three actions however, and accounts for about 15 videos.



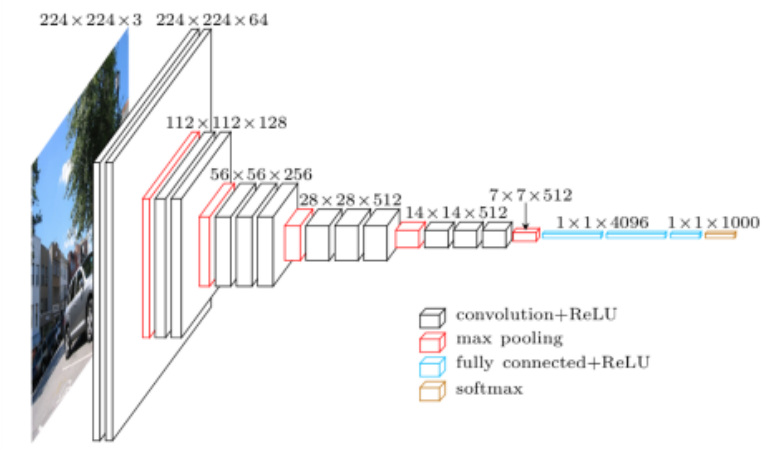


For testing and training, I aimed to partition the videos in about a 1:2 to 1:3 ratio for each action, with more videos being put into the training dataset. My model essentially learns frame by frame, which may not be an ideal method of learning since interaction from previous images should've also been considered, especially since most of these begin with myself standing still. Regardless, I thought this was okay as long as the frames with distinct actions were classified correctly.

DNN Model

Architecture

The model uses a sequential model with dense layers featuring dropout as well. In addition to, this will also use the VGG16 model. It's essentially a pre-trained CNN model performed on 1000 different classes. The 16 comes from the number of weights applied onto an image.



We first split each video by frames, about every 1 second. The VGG16 will help us extract certain image features first, then we will use those values and apply them onto our model.

```
base_model = VGG16(weights='imagenet', include_top=False)
```

```
X_train = base_model.predict(X_train) # extracting features
X_test = base_model.predict(X_test) # extracting features fo
```

```
# sizes must also change
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(25088,)))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(4, activation='softmax'))
```

My accuracy still holds around 93% accuracy. This is just due to the limited size of my custom dataset. The testing pool during this time around featured 31 different videos, which it successfully guessed 29 of them.

Input Shape of Tensor

Our train and test shape are shown below. 357 represents the number of frames split for the training dataset while 90 represents the number of frames for the testing dataset.

```
# Reshaping the training as well as validation frames
X_train = X_train.reshape(357, 7*7*512)
X_test = X_test.reshape(90, 7*7*512)
```

Output Shape of Tensor

- Y_train: (357,4)
- Y_test: (90,4)

Output Shape for the Layers

```
# Sizes must also change
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(25088,)))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(4, activation='softmax'))
```

- As seen above, our first layer is shaped: (357,512)
- Then the second goes to: (357,256)
- Third goes to this shape: (357,128)
- Fourth goes to this shape: (357,64)
- Finally, to represent the three different classes: (357,4)

Hyperparameters

List of hyperparameters

I use three simple hyperparameters here, batch size, epochs and dropout.

Range of hyperparameters tried

- Dropout varied between 0.3 and 0.6, I used less dropout near the later layers
- Batches varied between 20 to 60, I believe the best I have is at 32.
- Epochs varied between 25 to 100. Our dataset didn't need that much as I did before, so I tuned it down to 40

Personal best hyperparameters with my dataset

- Dropout at 0.5, and dropped out less per layer
- Batches are 32
- Epochs at 40

As of now, this is the best I can get for my dataset. If it was larger, these would change significantly.

Code Snippets

Training portion

The shapes are a bit different since I added more videos, but the structure of the code still stands. The most important thing to recognize here is how the VGG16 model is used to extract features before using it on our model.

```
# Creating validation set
# separating the target
y = train['class']

# creating the training and validation set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.2, stratify = y)

# creating dummies of target variable for train and validation set
y_train = pd.get_dummies(y_train)
y_test = pd.get_dummies(y_test)
```

```
[23] # Creating the base model of pre-trained VGG16 model
     # This is one form of CNN that can be used on each frame individually
     base_model = VGG16(weights='imagenet', include_top=False)
```

```
[24] X_train = base_model.predict(X_train) # extracting features for training frames
     X_test = base_model.predict(X_test) # extracting features for testing frames
```

```
[27] X_train.shape
```

```
(334, 7, 7, 512)
```

```
[28] X_test.shape
```

```
(84, 7, 7, 512)
```

```
[34] # Reshaping the training as well as validation frames
     X_train = X_train.reshape(334, 7*7*512)
     X_test = X_test.reshape(84, 7*7*512)

     # Normalizing the pixel values
     max = X_train.max()
     X_train = X_train/max
     X_test = X_test/max
```

```
[36] # Compiling the model
     model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])
```

```
[38] # training the model
     model.fit(X_train, y_train, epochs=100, validation_data=(X_test, y_test),  batch_size=30)
```

Testing

```
# creating two lists to store predicted and actual tags
predict = []
actual = []

# for loop to extract frames from each test video
for i in tqdm(range(test_videos.shape[0])):
    count = 0
    videoFile = test_videos[i]
    cap = cv2.VideoCapture(videoFile.split(' ')[0])   # capturing the video from the given path
    frameRate = cap.get(5) #frame rate
    # removing all other files from the temp folder
    # !rm -rf temp
    while(cap.isOpened()):
        frameId = cap.get(1) #current frame number
        ret, frame = cap.read()
        if (ret != True):
            break
        if (frameId % math.floor(frameRate) == 0):
            # storing the frames of this particular video in temp folder
            filename ='temp/' + "_frame%d.jpg" % count;count+=1
            cv2.imwrite(filename, frame)
    cap.release()

    # reading all the frames from temp folder
    images = glob("temp/*.jpg")
    prediction_images = []
    for i in range(len(images)):
        img = image.load_img(images[i], target_size=(224,224,3))
        img = image.img_to_array(img)
        img = img/255
        prediction_images.append(img)
    prediction_images = np.array(prediction_images)
    prediction_images = base_model.predict(prediction_images)
    prediction_images = prediction_images.reshape(prediction_images.shape[0], 7*7*512)
    prediction = model.predict_classes(prediction_images)
    # appending the mode of predictions in predict list to assign the tag to the video
    predict.append(y.columns.values[s.mode(prediction)[0][0]])

    # appending the actual tag of the video
    vidclass = (videoFile.split('_')[0])
    actual.append(vidclass.lower())
```

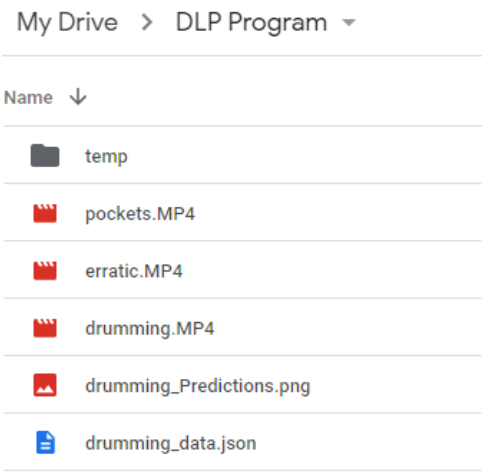
Performance from Training and Testing

Epochs have significantly been reduced since.

```
Epoch 19/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0318 - accuracy: 0.9888 - val_loss: 0.0666 - val_accuracy: 0.9889
Epoch 20/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0157 - accuracy: 0.9972 - val_loss: 0.0820 - val_accuracy: 0.9889
Epoch 21/40
357/357 [==============================] - 2s 7ms/step - loss: 0.0103 - accuracy: 1.0000 - val_loss: 0.0795 - val_accuracy: 0.9889
Epoch 22/40
357/357 [==============================] - 3s 8ms/step - loss: 0.0099 - accuracy: 0.9944 - val_loss: 0.0425 - val_accuracy: 0.9889
Epoch 23/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0188 - accuracy: 0.9944 - val_loss: 0.0501 - val_accuracy: 0.9889
Epoch 24/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0136 - accuracy: 0.9972 - val_loss: 0.0875 - val_accuracy: 0.9889
Epoch 25/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0231 - accuracy: 0.9972 - val_loss: 0.0917 - val_accuracy: 0.9889
Epoch 26/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0208 - accuracy: 0.9944 - val_loss: 0.0509 - val_accuracy: 0.9889
Epoch 27/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0181 - accuracy: 0.9944 - val_loss: 0.0405 - val_accuracy: 0.9889
Epoch 28/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0161 - accuracy: 0.9972 - val_loss: 0.0403 - val_accuracy: 0.9889
Epoch 29/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0158 - accuracy: 0.9944 - val_loss: 0.0724 - val_accuracy: 0.9889
Epoch 30/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0066 - accuracy: 0.9972 - val_loss: 0.0644 - val_accuracy: 0.9889
Epoch 31/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0120 - accuracy: 0.9972 - val_loss: 0.0724 - val_accuracy: 0.9889
Epoch 32/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0050 - accuracy: 1.0000 - val_loss: 0.0613 - val_accuracy: 0.9889
Epoch 33/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0034 - accuracy: 1.0000 - val_loss: 0.0494 - val_accuracy: 0.9889
Epoch 34/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 0.0517 - val_accuracy: 0.9889
Epoch 35/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.0543 - val_accuracy: 0.9889
Epoch 36/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0027 - accuracy: 1.0000 - val_loss: 0.0687 - val_accuracy: 0.9889
Epoch 37/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0024 - accuracy: 1.0000 - val_loss: 0.1003 - val_accuracy: 0.9889
Epoch 38/40
357/357 [==============================] - 2s 6ms/step - loss: 9.2599e-04 - accuracy: 1.0000 - val_loss: 0.1060 - val_accuracy: 0.9889
Epoch 39/40
357/357 [==============================] - 2s 6ms/step - loss: 0.0039 - accuracy: 1.0000 - val_loss: 0.0961 - val_accuracy: 0.9889
Epoch 40/40
357/357 [==============================] - 2s 6ms/step - loss: 5.5667e-04 - accuracy: 1.0000 - val_loss: 0.0931 - val_accuracy: 0.9889
<keras.callbacks.callbacks.History at 0x7f0b65cf6eb8>
```

Instructions on how to use the code

Both portions of the training and classification have been done on colab which will manage all dependencies for you. However, the training code relies on using your google drive directory, meaning that you will have to create a file hierarchy similar to what I have. Video demonstrations and this report will be provided on the github. This is what the directory looks like for the training portion:



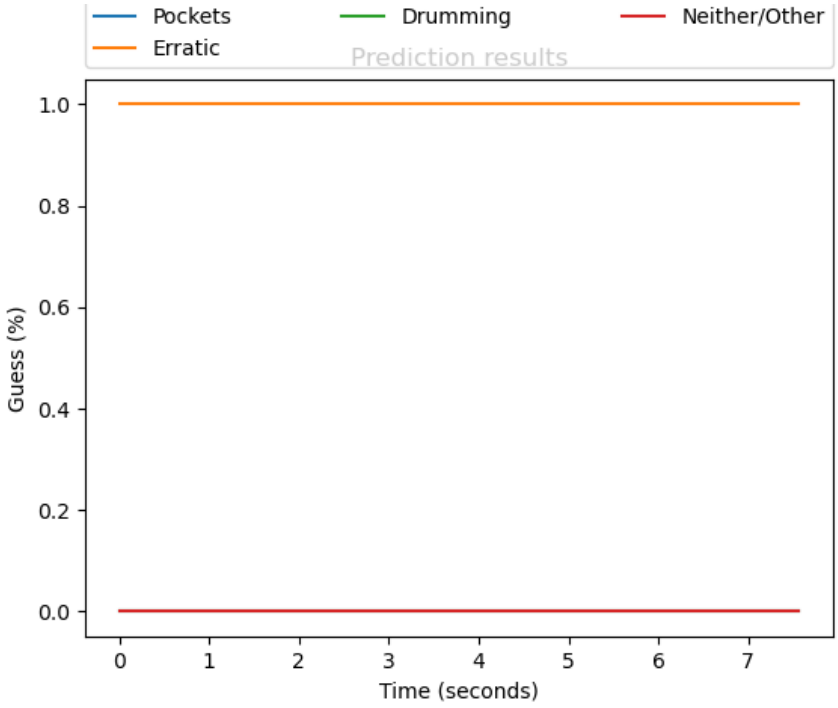
The classification can be downloaded from the github. There are a good number of dependencies involved. These include:

- OpenCV2
- Pandas
- Matplotlib
- Keras
- Tensorflow
- Glob
- tqdm

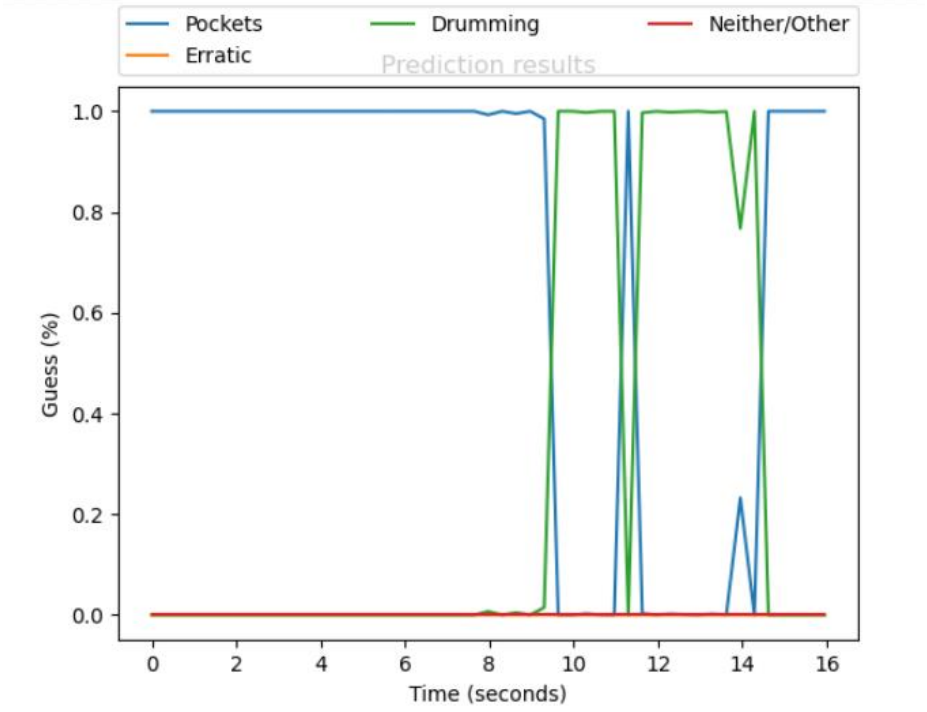
Be warned that the program saves each frame into a temporary folder before deleting it at the end. I'm unsure if this will work on all computers. Regardless, the program can be called through command line and requires an argument: the video filename. This video needs to be placed into the same directory as the file. It will then export the graph and json in the same directory.

Conclusions

I believe that my model is suffering from too much overfitting. This is an common example graph of how the predictions look for some of the videos that wasn't used in either the testing or training portions.



When using a mixed video, the graph becomes nonsensical. For example here, I am performing the hands in pockets, then erratic movements, then drumming.



I feel that my actions are not being detected properly, especially the erratic movements. I believe there are two reasons for this. The main reason is just how I approached training my model. I mentioned how every frame is classified on its own, but I didn't classify it well enough during the training. For example, putting my hands in my pockets will always have me standing still, which the model may have classified as pockets, when it should still be considered other until it hits the frame where hands are entering the pocket. Overall, it's possible that the three actions share many common features per frame, and the model isn't classifying it properly.

Another issue I feel is that my dataset isn't expansive enough. Putting my hands in pockets are simple enough, and some other cases may just involve other articles of clothing including coats. Erratic movements however is very hard to detect, and I feel like my dataset isn't expansive enough considering just how diverse the action can be, every frame would have the arms in super unique positions compared to the others, and I think the model struggles to figure out all the types of arm movements. Drumming also isn't expansive enough. All I have is myself taping a flat surface as well as others tapping on tables/couches. Still, this action can be done on a variety of surfaces at a variety of angles which I didn't cover enough.