Neural Network Gesture Recognition
Report 3
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.)

For my dataset, I'll be using recordings of myself performing these actions. This project is done on Google Colab as it was easy to develop on as well as boasting some great performance when training.

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:

DNN Model

Architecture

Our model will use the sequential model and will include three hidden layers. We will also use some dropout to prevent overfitting:

```
#defining the model architecture
model = Sequential()
model.add(Dense(1000, activation='relu', input_shape=(25088,)))
model.add(Dropout(0.5))
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))
```

This was similar to what I had before, but this time I got a 93% accuracy. I'd say it's due to how I partitioned this new dataset. I made sure to balance the testing and training datasets more carefully with my additions from the previous reports.

Input Shape of Tensor

Our train and test shape are shown below. 334 represents the number of frames split for the training dataset while 84 represents the number of frames for the testing dataset.

```
# Reshaping the training as well as vali
X_train = X_train.reshape(334, 7*7*512)
X_test = X_test.reshape(84, 7*7*512)
```

Output Shape of Tensor

- Y_train: (334,3)
- Y_test: (84,3)

Output Shape for the Layers

- As seen above, our first layer is shaped: (334,1000)
- Then the second goes to: (334,500)
- Third goes to this shape: (334,100)
- Finally, to represent the three different classes: (334,3)

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, but personally I found decent results sticking at around 0.5

- Batches varied between 20 to 60, 30-40 presented to be decent sizes.
- Epochs varied between 25 to 100. The best we had was at 40 so far

Personal best hyperparameters with my dataset

- Dropout at 0.5
- Batches are 35
- Epochs at 40

The main goal now is to experiment with these parameters and maybe try to add more to the dataset.

Code Snippets

Training portion

```
# 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

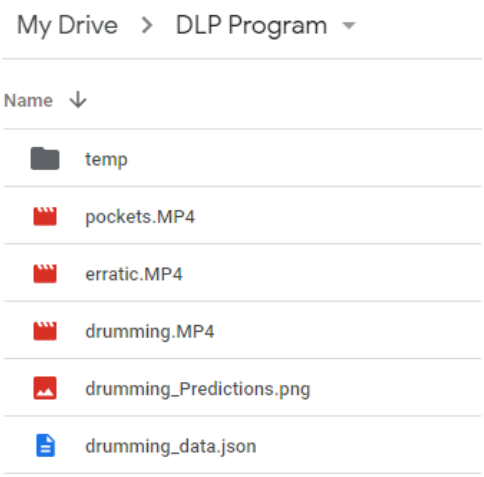
The accuracy and loss values do not look very good.

```
Train on 334 samples, validate on 84 samples
Epoch 1/100
334/334 [==============================] - 4s 12ms/step - loss: 0.9302 - accuracy: 0.5569 - val_loss: 0.4854
Epoch 2/100
334/334 [==============================] - 4s 12ms/step - loss: 0.6888 - accuracy: 0.6707 - val_loss: 0.3536
Epoch 3/100
334/334 [==============================] - 4s 12ms/step - loss: 0.5361 - accuracy: 0.7545 - val_loss: 0.3077
Epoch 4/100
334/334 [==============================] - 4s 12ms/step - loss: 0.3640 - accuracy: 0.8353 - val_loss: 0.1218
Epoch 5/100
334/334 [==============================] - 4s 13ms/step - loss: 0.1767 - accuracy: 0.9431 - val_loss: 0.0458
Epoch 6/100
334/334 [==============================] - 6s 17ms/step - loss: 0.1198 - accuracy: 0.9611 - val_loss: 0.0168
Epoch 7/100
334/334 [==============================] - 4s 12ms/step - loss: 0.0886 - accuracy: 0.9671 - val_loss: 0.0485
Epoch 8/100
334/334 [==============================] - 4s 12ms/step - loss: 0.0638 - accuracy: 0.9790 - val_loss: 0.0082
```

Instructions on how to use the code

Both portions of the testing and output have been done on colab which will manage all dependencies for you. However, the 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 program portion, you'll have to provide some videos and manually add a folder named temp:



This is the directory I have for the testing portion. You will need to make the folders. I built my program around those filenames, so they probably can't work with most other things. If needed, I can provide my dataset on request and share through google drive so this portion can function: