Neural Network Gesture Recognition
Report 1
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

Currently, my dataset is very limited as I had difficulty finding a diverse dataset for these three gestures, and needs more variation. The set involves approximately 60 videos of myself performing the three gestured mentioned, split evenly. These are recorded vertically in order to catch my whole body. I perform these actions normally as well as varying it sometimes such as turning around to get a more realistic scenario. These videos are then split by frames before being classified individually with my neural net. Some example frames are pictured below:

DNN Model

Architecture

Our model will use the sequential model and will include three hidden layers. We will also use some dropout to hopefully 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'))
```

Input Shape of Tensor

Our train and test shape are shown below, the 224 and 56 represent the number of frames split between each dataset.

```
X_train = X_train.reshape(224, 7*7*512)
X_test = X_test.reshape(56, 7*7*512)
```

Output Shape of Tensor

Expect

Y_train: (224,3)

Y_test: (56,3)

Output Shape for the Layers

- As seen above, our first layer is shaped: (224,1000)
- Then the second goes to: (224,500)
- Third goes to this shape: (224,100)
- Finally to represent the three different classes: (224,3)

Hyperparameters

List of hyperparameters

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

Range of hyperparameters tried

- For now, we kept the dropout the same throughout all the layers at 0.5
- Batches varied between 20 to 60, could try larger batches
- Epochs go from 100 to 200. Found it more feasible to stick to 100.

Personal best hyperparameters with my dataset

- Dropout at 0.5
- Batches are 30
- Epochs at 100

Epochs may be too high. I also expect to change these when I switch up my dataset.

Code Snippets

Training portion

```
[29] # 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)
     y_test.shape
```

```
     # 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)

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

```
[21] # Reshaping the training as well as validation frames
     X_train = X_train.reshape(224, 7*7*512)
     X_test = X_test.reshape(56, 7*7*512)

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

Training Time

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

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

Testing

```
# !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
id_num = (int)(videoFile.split('_')[1].split('.')[0])
if id_num >= 352 and id_num <= 368:
    actual.append("pockets")
if id_num >= 369 and id_num <= 392:
    actual.append("erratic")
if id_num >= 394 and id_num <= 413:
    actual.append("drumming")
```

```
[ ] # checking the accuracy of the predicted tags
    from sklearn.metrics import accuracy_score
    accuracy_score(predict, actual)*100
```

Performance from Training and Testing

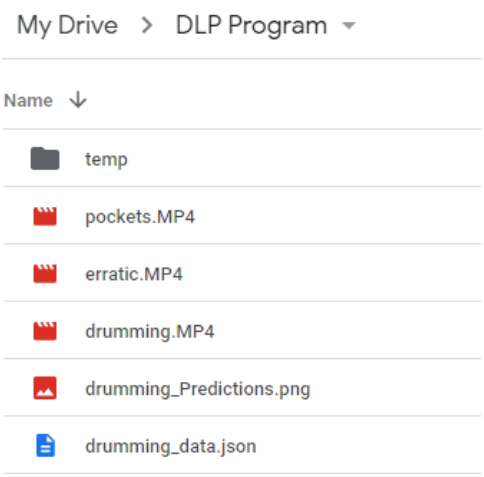
Seen below. I'm wondering if I have too many epochs or if it's possible that there's some overfitting.

```
Train on 224 samples, validate on 56 samples
Epoch 1/100
224/224 [==============================] - 4s 17ms/step - loss: 0.8913 - acc: 0.5804 - val_loss: 0.3478 - val_acc: 0.7857
Epoch 2/100
224/224 [==============================] - 3s 14ms/step - loss: 0.5799 - acc: 0.7277 - val_loss: 0.1802 - val_acc: 0.9821
Epoch 3/100
224/224 [==============================] - 3s 13ms/step - loss: 0.2608 - acc: 0.8795 - val_loss: 0.0811 - val_acc: 0.9821
Epoch 4/100
224/224 [==============================] - 3s 14ms/step - loss: 0.1749 - acc: 0.9375 - val_loss: 0.0374 - val_acc: 0.9821
Epoch 5/100
224/224 [==============================] - 3s 13ms/step - loss: 0.1898 - acc: 0.9286 - val_loss: 0.0163 - val_acc: 1.0000
Epoch 6/100
224/224 [==============================] - 3s 13ms/step - loss: 0.0818 - acc: 0.9687 - val_loss: 0.0458 - val_acc: 0.9821
Epoch 7/100
224/224 [==============================] - 3s 14ms/step - loss: 0.0413 - acc: 0.9777 - val_loss: 0.0057 - val_acc: 1.0000
Epoch 8/100
224/224 [==============================] - 3s 13ms/step - loss: 0.0324 - acc: 0.9911 - val_loss: 0.0095 - val_acc: 1.0000
Epoch 9/100
224/224 [==============================] - 3s 13ms/step - loss: 0.0305 - acc: 0.9911 - val_loss: 0.0079 - val_acc: 1.0000
Epoch 10/100
224/224 [==============================] - 3s 13ms/step - loss: 0.0122 - acc: 1.0000 - val_loss: 0.0070 - val_acc: 1.0000
Epoch 11/100
224/224 [==============================] - 4s 16ms/step - loss: 0.0112 - acc: 0.9955 - val_loss: 6.6124e-04 - val_acc: 1.0000
Epoch 12/100
224/224 [==============================] - 3s 13ms/step - loss: 0.0064 - acc: 0.9955 - val_loss: 0.0021 - val_acc: 1.0000
Epoch 13/100
224/224 [==============================] - 3s 13ms/step - loss: 0.0059 - acc: 1.0000 - val_loss: 0.0011 - val_acc: 1.0000
Epoch 14/100
224/224 [==============================] - 4s 16ms/step - loss: 0.0147 - acc: 0.9955 - val_loss: 7.5242e-04 - val_acc: 1.0000
Epoch 15/100
224/224 [==============================] - 3s 14ms/step - loss: 0.0309 - acc: 0.9955 - val_loss: 0.0334 - val_acc: 0.9821
Epoch 16/100
224/224 [==============================] - 3s 13ms/step - loss: 0.0097 - acc: 0.9955 - val_loss: 0.0055 - val_acc: 1.0000
Epoch 17/100
224/224 [==============================] - 3s 13ms/step - loss: 0.0041 - acc: 1.0000 - val_loss: 1.2878e-04 - val_acc: 1.0000
Epoch 18/100
224/224 [==============================] - 3s 13ms/step - loss: 0.0034 - acc: 1.0000 - val_loss: 1.4144e-04 - val_acc: 1.0000
Epoch 19/100
224/224 [==============================] - 3s 13ms/step - loss: 0.0024 - acc: 1.0000 - val_loss: 3.5841e-04 - val_acc: 1.0000
Epoch 20/100
224/224 [==============================] - 3s 13ms/step - loss: 0.0010 - acc: 1.0000 - val_loss: 8.9689e-04 - val_acc: 1.0000
Epoch 21/100
224/224 [==============================] - 4s 16ms/step - loss: 9.7866e-04 - acc: 1.0000 - val_loss: 3.7327e-04 - val_acc: 1.0000
Epoch 22/100
224/224 [==============================] - 3s 13ms/step - loss: 0.0039 - acc: 1.0000 - val_loss: 0.0010 - val_acc: 1.0000
Epoch 23/100
```

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: