

Assignment 5

AI-F19

Trevor Buck

Deep Learning Report

1.



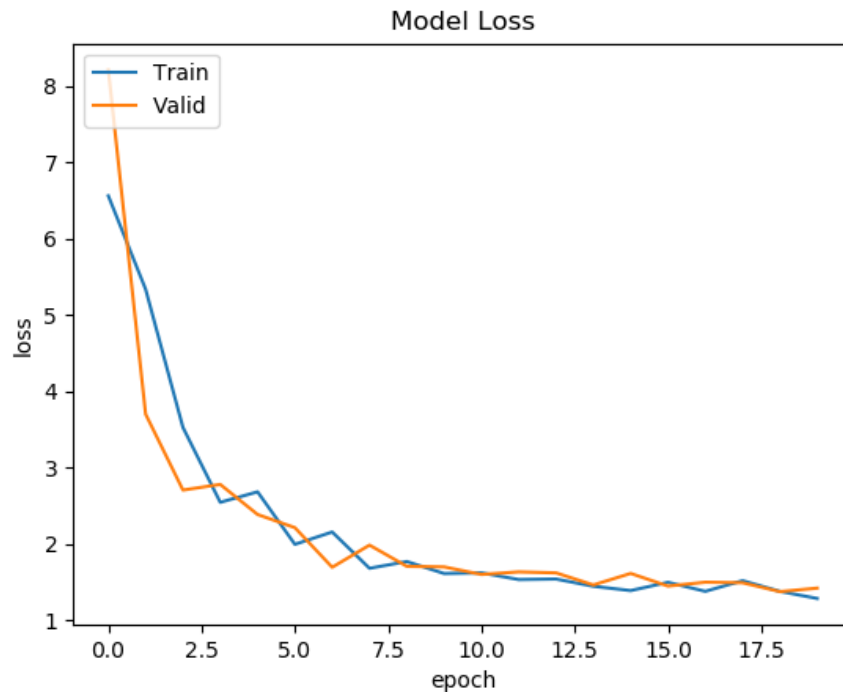
The three pictures above are 3 randomly selected photos from the raw dataset. Because they aren't profile shots, they contain a lot of noise. They have backgrounds, clothes, bodies, other people. Because these photos aren't 'clean', any facial recognition patterns would be severely limited.



These next three pictures were randomly sampled from the cropped and sized dataset. Now the boundaries might be a bit off (as seen in the first picture) but for the most part, the cropping worked rather well. In most cases, you can see the face, and *only* the face. This is very useful because that means that the only data points we have are the data points that matter. By cleaning up the data set and preprocessing it, we are enabling our Neural Network to narrow in on the fine details of the images.

The overall benefit of preprocessing and cleaning the images is that now all of our images have certain features in common. By resizing everything to be the same, we can standardize the data and make it easier for our Neural Network to distinguish the images.

2.



```
Epoch 20/20
- 0s - loss: 1.2823 - accuracy: 0.5440 - val_loss: 1.4182 - val_accuracy: 0.4639
Test loss: 1.4181667595024567
Test accuracy: 0.46385541558265686
```

```
# building a linear stack of layers with the sequential model
model = Sequential()
model.add(Dense(512, input_shape=(3600,)))
model.add(Activation('relu'))

model.add(Dense(6))
model.add(Activation('softmax'))

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

# training the model and saving metrics in history
history = model.fit(X_train, Y_train,
                    batch_size=128, epochs=20,
                    verbose=2,
                    validation_data=(X_test, Y_test))
```

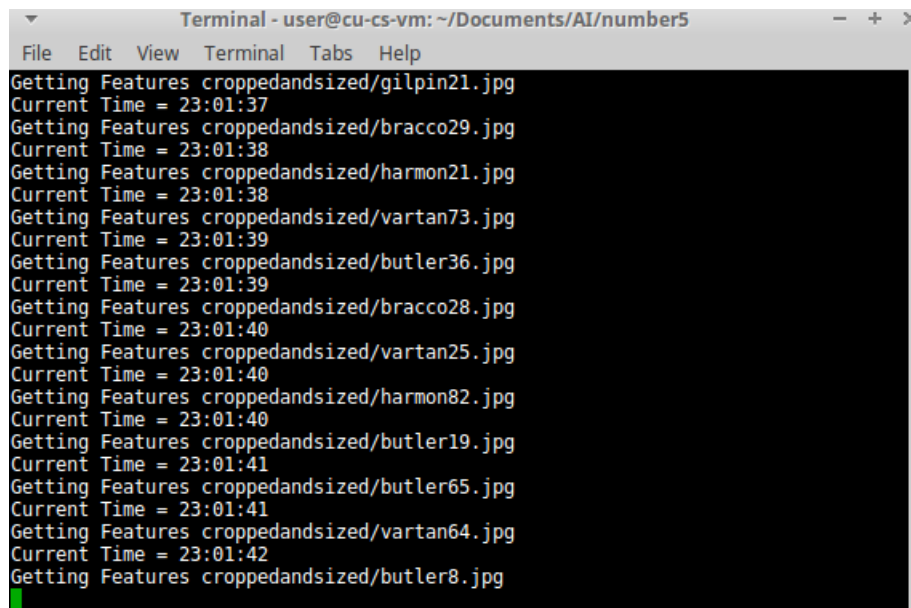
As you can see from the pictures, my overall accuracy was rather low. I ran this multiple times and usually ended up somewhere near 50%.

Process: In part 1 I saved all the cropped images to a folder. This way I didn't have to re-crop the images every time I wanted to run my program. This means that for part 2, all I had to do was open each image in the folder, change it to grayscale, and then put the information in a numpy array. The formed array had the shape (60,60) so I simply reshaped the array into one row of 3600. This formed my xList. To create my list of labels (yList), I went through the names of each image and gave it a value from (1-6) depending on which

of the actors name it matched with. Once I had these two arrays, forming the NN wasn't too hard. I used much of the code from the Keras example. The only things I really changed were the input size (728 -> 3600) and the Dense (10->6) labels. I decided to leave the batch_size, epochs, and test_size the same.

Note* I played with all these values in part 4

3.

A terminal window titled "Terminal - user@cu-cs-vm: ~/Documents/AI/number5" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal displays a series of commands and their outputs. Each command is "Getting Features croppedandsized/[filename].jpg" and the output is "Current Time = [timestamp]". The files listed are gilpin21.jpg, bracco29.jpg, harmon21.jpg, vartan73.jpg, butler36.jpg, bracco28.jpg, vartan25.jpg, harmon82.jpg, butler19.jpg, butler65.jpg, vartan64.jpg, and butler8.jpg. The timestamps range from 23:01:37 to 23:01:42. A green cursor is visible at the bottom of the terminal window.

```
Terminal - user@cu-cs-vm: ~/Documents/AI/number5
File Edit View Terminal Tabs Help
Getting Features croppedandsized/gilpin21.jpg
Current Time = 23:01:37
Getting Features croppedandsized/bracco29.jpg
Current Time = 23:01:38
Getting Features croppedandsized/harmon21.jpg
Current Time = 23:01:38
Getting Features croppedandsized/vartan73.jpg
Current Time = 23:01:39
Getting Features croppedandsized/butler36.jpg
Current Time = 23:01:39
Getting Features croppedandsized/bracco28.jpg
Current Time = 23:01:40
Getting Features croppedandsized/vartan25.jpg
Current Time = 23:01:40
Getting Features croppedandsized/harmon82.jpg
Current Time = 23:01:40
Getting Features croppedandsized/butler19.jpg
Current Time = 23:01:41
Getting Features croppedandsized/butler65.jpg
Current Time = 23:01:41
Getting Features croppedandsized/vartan64.jpg
Current Time = 23:01:42
Getting Features croppedandsized/butler8.jpg
```

Part 3 took the longest to run because I had to run the GetVGG features each time. I must have a slow computer because it usually took around 3-4 minutes just to compute. I decided to go with the standard "block4_pool".

Unfortunately, I couldn't quite get the function "trainFaceClassifier_VGG();" to work... I tried passing the data as flattened, as a python list, as a reshaped numpy array, and a few other things, but none of them successfully worked. The 5 dimensionalities really threw me for a loop! And when I combined that with the amount of time that it took just to extract the features, I simply couldn't finish it in time.

4.

```
def trainFaceClassifier(preProcessedImages, labels):
    X_train, X_test, Y_train, Y_test = train_test_split(preProcessedImages, labels, test_size=0.1, shuffle=True)
    print(X_train.shape)

    # building a linear stack of layers with the sequential model
    model = Sequential()
    model.add(Dense(1024, input_shape=(3600,)))
    model.add(Activation('relu'))

    model.add(Dense(6))
    model.add(Activation('softmax'))

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

    # training the model and saving metrics in history
    history = model.fit(X_train, Y_train,
                        batch_size=32, epochs=20,
                        verbose=0,
                        validation_data=(X_test, Y_test))

Epoch 19/20
- 1s - loss: 0.3626 - accuracy: 0.9012
6
Epoch 20/20
- 1s - loss: 0.3896 - accuracy: 0.8831
6
```

The three circles correspond to the 'test_size', 'dense(layer)', and 'batch_size'. In my experiments, I focused on changing these 3 values as they seemed to have the biggest effect on my accuracy score. I tried varying a few other values such as the verbose, epochs, and activation function, but they didn't work as well. When I changed the relu function to the sigmoid, I got a worse score. When I lowered my epochs, I got a worse score. When I increased the numbers of epochs, I got a slightly better score, but it wasn't worth the tradeoff in time. The verbose value didn't have any effect on the score, as it just changed the way my terminal portrayed the information.

By decreasing the test_size, I allowed my classifier to train on more examples. This was nice and probably had the greatest impact on my score. My guess is that because our dataset was relatively small (~500), our classifier wasn't getting enough data to train on. So although we significantly reduced the number of test examples, we gave our classifier the opportunity to receive more training and that paid dividends.

The other two values that I changed impacted my score but on a slightly smaller scale. I increased the number of layers, and I decreased our batch size. Decreasing the batch size took advantage of the 'mini-batch' theory. Once again, the trade-off here is time, but the results were worth it in this case.