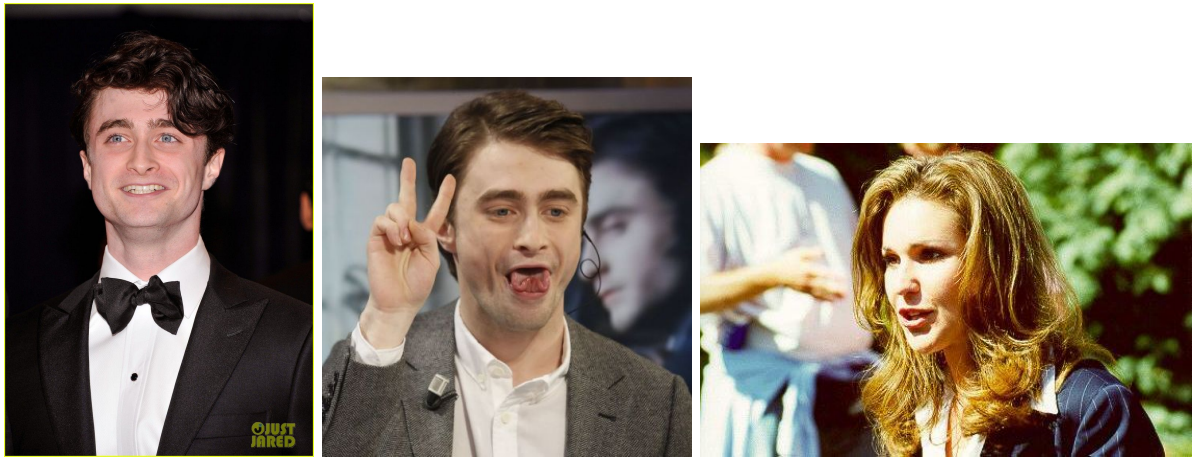
Calvin Zikakis
December 14 2019
Artificial Intelligence
Assignment 5

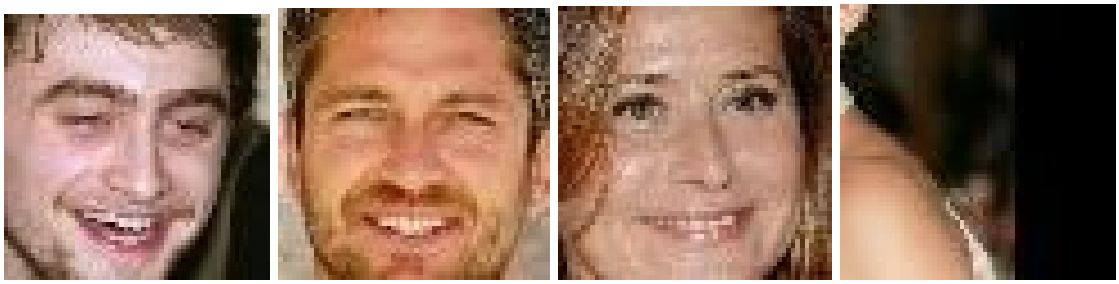
Part 1 -

Uncropped Photos:



Here we can see an example of three uncropped photos from the dataset. As you can see the quality of these photos varies vastly between the images. Most images are from different angles with the actors making different faces.

Cropped Photos:



Above are examples of cropped and resized images. As you can see the bounding boxes do a great job of accurately bounding the faces with the only exception being the image on the right. This preprocessing is important because it allows us to make sure the network is only classifying the faces in the photos and not background data due to our cropping. The resizing of the photos is important because it standardizes all the photos to have the same amount of pixels and also lowers the size of the image to make the algorithm run more efficiently.

Part 2 -

Model Description -

All images were preprocessed by cropping to only the face, standardizing size / down scaling, and grey scaling all images. Grey scaling images is important because it allows our network to not have to focus on multiple values per pixel. This increases performance and standardizes lighting conditions that could change skin tones. The weights were then scaled by dividing the photo array by 255 to get pixel values in the range 0 to 1.

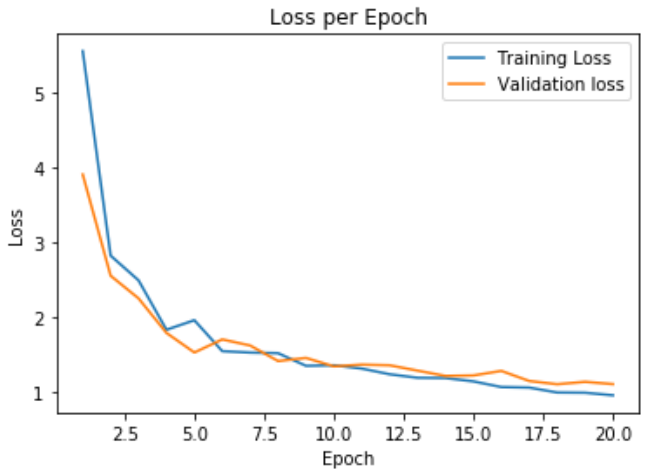
After the photos were preprocessed, they were split into training, testing, and validation sets. This split was done using Sklearn's train_test_split. The data was first split so that 80% was training data and the other 20% was testing data. Next the training data was split once again to get a validation set. My validation set consisted of 10% of the training data.

The training set and validation set was then passed into a neural network with the following architecture:

Input amount of (*, 3600) to account for each pixel in an image (60*60) which provided an output of (*,270) using a ReLU activation function. The node size of 270 was selected because it provided optimal efficiency while providing great performance. This output traveled to another layer with output (*, 6) to accommodate the 6 possible classes which used a softmax activation to provide a probability of each class.

```
 - 1s - loss: 5.5618 - acc: 0.1920 - val_loss: 3.9103 - val_acc: 0.2000
Epoch 2/20
 - 0s - loss: 2.8252 - acc: 0.2195 - val_loss: 2.5564 - val_acc: 0.2444
Epoch 3/20
 - 0s - loss: 2.4956 - acc: 0.2344 - val_loss: 2.2535 - val_acc: 0.2000
Epoch 4/20
 - 0s - loss: 1.8340 - acc: 0.2668 - val_loss: 1.7923 - val_acc: 0.2667
Epoch 5/20
 - 0s - loss: 1.9623 - acc: 0.3017 - val_loss: 1.5301 - val_acc: 0.4222
Epoch 6/20
 - 0s - loss: 1.5481 - acc: 0.4264 - val_loss: 1.7049 - val_acc: 0.3333
Epoch 7/20
 - 0s - loss: 1.5279 - acc: 0.4040 - val_loss: 1.6236 - val_acc: 0.3111
Epoch 8/20
 - 0s - loss: 1.5216 - acc: 0.3940 - val_loss: 1.4157 - val_acc: 0.5111
Epoch 9/20
 - 0s - loss: 1.3544 - acc: 0.5162 - val_loss: 1.4574 - val_acc: 0.4667
Epoch 10/20
 - 0s - loss: 1.3591 - acc: 0.4913 - val_loss: 1.3448 - val_acc: 0.4667
Epoch 11/20
 - 0s - loss: 1.3150 - acc: 0.5561 - val_loss: 1.3681 - val_acc: 0.5111
Epoch 12/20
 - 0s - loss: 1.2398 - acc: 0.5960 - val_loss: 1.3591 - val_acc: 0.4444
Epoch 13/20
 - 0s - loss: 1.1924 - acc: 0.6060 - val_loss: 1.2873 - val_acc: 0.4889
Epoch 14/20
 - 0s - loss: 1.1888 - acc: 0.5411 - val_loss: 1.2167 - val_acc: 0.5111
Epoch 15/20
 - 0s - loss: 1.1442 - acc: 0.5835 - val_loss: 1.2233 - val_acc: 0.6000
Epoch 16/20
 - 0s - loss: 1.0686 - acc: 0.6958 - val_loss: 1.2839 - val_acc: 0.4889
Epoch 17/20
 - 0s - loss: 1.0619 - acc: 0.6708 - val_loss: 1.1490 - val_acc: 0.6444
Epoch 18/20
 - 0s - loss: 0.9959 - acc: 0.7282 - val_loss: 1.1059 - val_acc: 0.6444
Epoch 19/20
 - 0s - loss: 0.9930 - acc: 0.7282 - val_loss: 1.1380 - val_acc: 0.7333
Epoch 20/20
 - 0s - loss: 0.9584 - acc: 0.7431 - val_loss: 1.1074 - val_acc: 0.6889
Test loss: 1.1050421169825964
Test accuracy: 0.6607142857142857
```

These provided outputs show the accuracy and loss of the network per epoch. Accuracy is rather low getting to a maximum of 74% but is acceptable for the low amount of data considering chance is 17%.



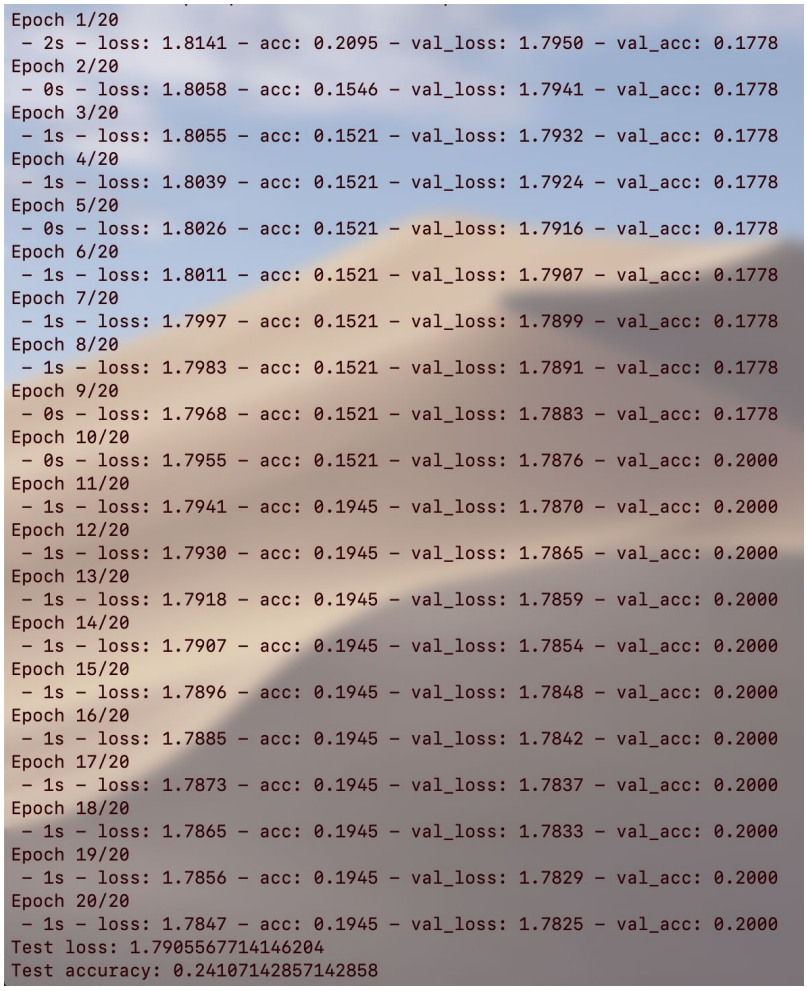
Part 3 -

- Explain your choice of VGG16 layer used as input.
- Compare the performance of your new network which used features from VGG6 to your network using the raw images from part 2. Is it better or worse? Why do you think this is?

I used the block 4 layer as my input. I decided on block 4 because it provides better performance over block 5 for my network due to less time spent computing another 3 convolutions with little benefit.

My network performed significantly worse when using the features extracted from the VGG6 providing a top accuracy of a little over 24%. This is still higher than chance but not by much. I believe this performed badly due to limits in computational power. My input for the network was 100,352, because this input is so large I wanted my next layer to have width of 10,000. Unfortunately, the computational time for this made it so a single epoch took thirty minutes to run. I had to size the layer down to only 100 nodes wide. This causes the algorithm to over compress the data and result in a bad accuracy.

With more computational power, this network would have been set up as 100,352 (input) -> 10,000 -> 1,000 -> 100 -> 6 (output). Unfortunately I had to settle for 100,352 -> 100 -> 24 -> 6. This caused the network to over generalize the features and provide bad results.



Part 4 -

I got my network from part 2 to an accuracy of 83% with variation of around 5%. In order to get this accuracy I changed my architecture to 3600 (input) -> 275 relu -> 72 relu -> 0.1 dropout rate -> 6 softmax.
This architecture was accompanied by an epoch amount of 50 and a batch size of 18.

```
Epoch 40/50
 - 0s - loss: 0.2899 - acc: 0.9027 - val_loss: 0.6404 - val_acc: 0.8222
Epoch 41/50
 - 0s - loss: 0.1641 - acc: 0.9451 - val_loss: 0.5583 - val_acc: 0.7111
Epoch 42/50
 - 0s - loss: 0.1510 - acc: 0.9476 - val_loss: 0.5792 - val_acc: 0.7778
Epoch 43/50
 - 0s - loss: 0.1875 - acc: 0.9277 - val_loss: 0.6433 - val_acc: 0.7111
Epoch 44/50
 - 0s - loss: 0.1606 - acc: 0.9476 - val_loss: 0.6099 - val_acc: 0.7556
Epoch 45/50
 - 0s - loss: 0.1158 - acc: 0.9776 - val_loss: 0.5765 - val_acc: 0.7556
Epoch 46/50
 - 0s - loss: 0.1124 - acc: 0.9626 - val_loss: 0.7802 - val_acc: 0.7778
Epoch 47/50
 - 0s - loss: 0.1338 - acc: 0.9601 - val_loss: 0.7367 - val_acc: 0.7778
Epoch 48/50
 - 0s - loss: 0.2573 - acc: 0.9202 - val_loss: 0.7602 - val_acc: 0.7556
Epoch 49/50
 - 0s - loss: 0.1181 - acc: 0.9576 - val_loss: 0.6354 - val_acc: 0.7778
Epoch 50/50
 - 0s - loss: 0.0761 - acc: 0.9900 - val_loss: 0.6251 - val_acc: 0.7778
Test loss: 0.8463937299592155
Test accuracy: 0.8303571428571429
```

Last 10 epochs show above.

I believe this model showed an improvement over the model from part 2 due to the extra layers being able to better classify features of the images rather than individual pixels. All layers have less nodes than the previous because it forces the network to classify features in the images rather than individual items. For example, my first layer steps down to 275 nodes to force the algorithm to look for individual features such as lines. Next it steps down to 72 nodes to looks for patterns in those lines and then again down to 6 to best classify those patterns to their labels.

The dropout helps make sure the network does not overfit. Using a dropout rate of 0.1 provided the perfect combination of dropped nodes for the network.

I found that adding additional layers reduces the accuracy dramatically from 83% to 18%. I believe this is due to making the model to complex which it results in trying to classify features that do not exist.

```
Epoch 40/50
 - 0s - loss: 1.7779 - acc: 0.2070 - val_loss: 1.7898 - val_acc: 0.2000
Epoch 41/50
 - 0s - loss: 1.7749 - acc: 0.2120 - val_loss: 1.7898 - val_acc: 0.2000
Epoch 42/50
 - 0s - loss: 1.7753 - acc: 0.2120 - val_loss: 1.7897 - val_acc: 0.2000
Epoch 43/50
 - 0s - loss: 1.7741 - acc: 0.2145 - val_loss: 1.7896 - val_acc: 0.2000
Epoch 44/50
 - 0s - loss: 1.7770 - acc: 0.2170 - val_loss: 1.7898 - val_acc: 0.2000
Epoch 45/50
 - 0s - loss: 1.7747 - acc: 0.2120 - val_loss: 1.7900 - val_acc: 0.2000
Epoch 46/50
 - 0s - loss: 1.7749 - acc: 0.2120 - val_loss: 1.7905 - val_acc: 0.2000
Epoch 47/50
 - 0s - loss: 1.7745 - acc: 0.2120 - val_loss: 1.7903 - val_acc: 0.2000
Epoch 48/50
 - 0s - loss: 1.7746 - acc: 0.2120 - val_loss: 1.7904 - val_acc: 0.2000
Epoch 49/50
 - 0s - loss: 1.7745 - acc: 0.2120 - val_loss: 1.7904 - val_acc: 0.2000
Epoch 50/50
 - 0s - loss: 1.7745 - acc: 0.2120 - val_loss: 1.7904 - val_acc: 0.2000
Test loss: 1.7678615025111608
Test accuracy: 0.16964285714285715
```

I also found that increasing layer width resulted in a decrease of accuracy. This is because the network is not forced to classify patterns but instead tries to classify individual pixels.

I believe that with more training data, this network would easily be able to get its accuracy up to at least 90%. This could also be achieved with K-fold cross validation.