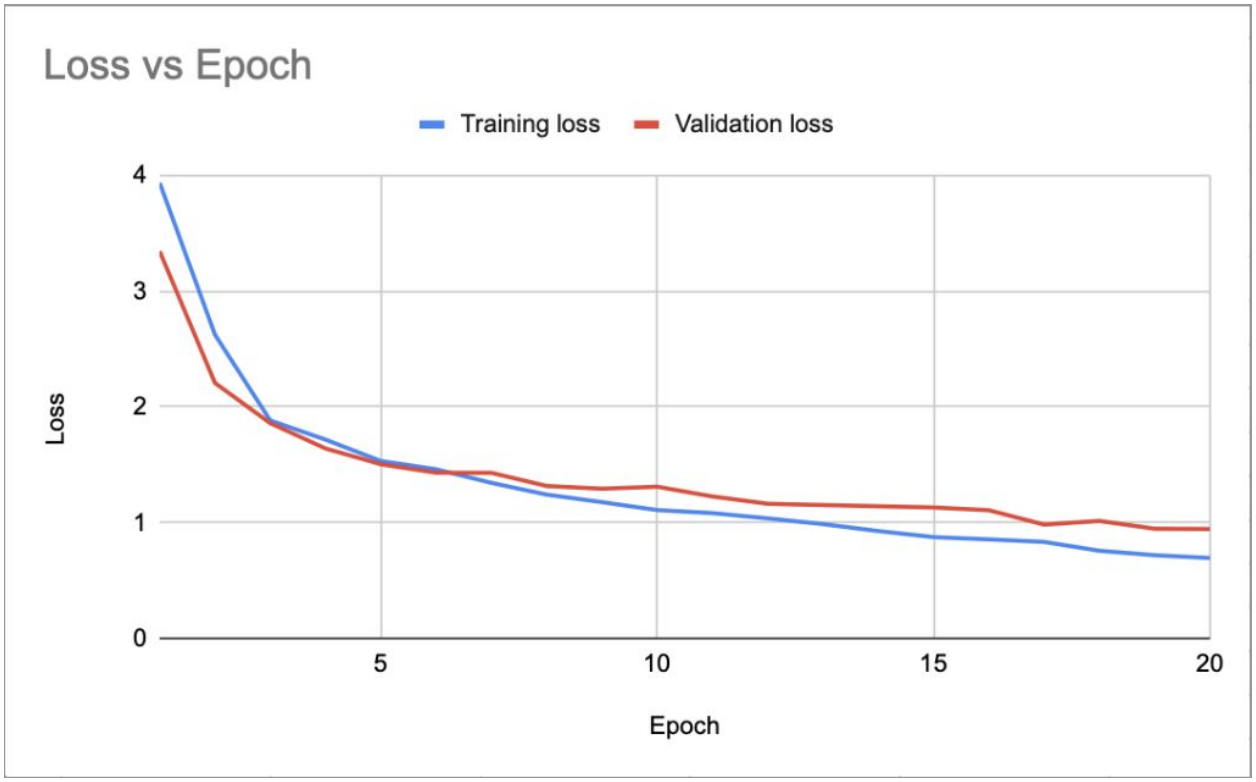
1 The quality of the faces in the dataset were fairly good. After cropping the data I was able to get a nice view of just the face of each of the 6 actors although for a handful of the data had the face either too zoomed in, only part of the face showing, or too zoomed out. This tells me that some of the bounding boxes were inaccurate although for the majority of the data the bounding boxes did just fine. For all of our purposes with this data it did not cause a huge issue when cropping and normalizing(resizing) the image. Pre-processing the data is important for classifying faces because the original uncropped picture had a variety of other features that would distract from the face such as the clothes the actor was wearing, other body parts, full body pictures etc. Our data's sole purpose is to capture the face of an actor and classify it. Including these other features would only increase the complexity of the neural network and increase the bias and variance of your data. In addition, it is necessary to normalize or resize our data because giving the neural network inputs of all different sizes would be a recipe for disaster. We would have issues with converting our image to an array of numbers(grid data) and reshaping everything to fit would be a nightmare. So, we resize all of our pictures to be 60x60 for our neural network to easily process this data. The effect of resizing the images did make all of the images a bit more blurry for the pictures with extremely different resolutions which is somewhat of an issue, but it is important that we resize the data for normalization across the neural network. When reading in all sorts of different data the last thing you want to do is introduce more variance in the datasets besides just the different face classification of the actors.
2.



My test data classification accuracy came out to be 0.777 or 77.7%. In other words, my data was able to classify the faces correctly to one of the six actors 77.7% of the time.Overall we can see that my data generally underfits since validation loss does not increase towards the last few epochs. In other words, both the training and validation data loss do not decrease anymore.

```
Test loss: 0.7231813506646589
Test accuracy: 0.7777777910232544
```

For this data set, I split up my data into train, validation, and test data by using the scikit train_test_split function. First, I split the data into train and test using the train_test_split function and then did another train_test_split to split the training data into a smaller training data set and validation data. This is how they specified to do it online without having to manually split the train test split data. This way the test data is completely separate and the large training data can be split into a still decently large training data set and a validation data set. I preprocessed the inputs by first converting each of my images to a numpy array and using vstack to put them row by row in a preprocessed image array. Next, I declared my preprocessed image to a float32 to allow model fitting to this data and divided by 255 to make the network more interpretable. Note that I converted from RGB(3 channels) to grayscale(1 channel) to make processing of data more simple. Last, I converted each of my labels(the name of the actors) to a number from 0 to 5(6 labels for 6 actors) to identify each of the actors. In my train_face_classifier function, I split up this data into train, validation, and test data as explained above and used the keras_mnist_sample.py as a template for modelling and fitting my data. Instead of using the relu activation function, I decided to use sigmoid function in my model. Sigmoid functions are generally used to introduce non-linearity into the model. The sigmoid function can only lie between 0 and 1 so it is good for probabilistic predictions. I chose to use 1000 hidden layers and 3600 as my input shape representing the 60x60 image.

3. For this data, I decided to use the block_4_pool layer as my choice for the vgg16 layer. The block_4_pool uses 2 dimensional max pooling to downsample our input representation by selecting our max in a certain window/grid of our image and overall reduce the dimensionality. It will return a tuple of 4 with our (num_images, 14, 14, 512). This data can be used for feature extraction from our large data-set of images using the getvggfeatures function provided on piazza. The only difference in block_4_pool and block_5_pool was that one had a 4d tensor output and the other has a 5d tensor output. I chose to use the block_4_pool to make the data as simply processable as possible and attempt to prevent over-fitting similar to the model in part 2 which underfit. If we used block_5 pool we would have more hidden layers to handle.

My performance on the data in the vgg6 network performed slightly better than the network used in part 2 with an accuracy of 0.814 or 81.4%. This is better because it means that we are classifying the actors faces to their names correctly 81.4% of the time. This is because in our model in part 2 uses a fully connected neural network similar to one that classifies handwritten digits to classify our face data whereas our model in part 3 used a convolutional neural network with both max pooling and a feedforward neural network(like the one used in part 2). Vgg uses

the data provided(images) and turns them into the activations of that specific layer. In other words, we can use these activations as features in our CNN/FFN and fit to this data using our same fitting model as in part 2. The difference in part 2 and 3 is that part 2 gives us the actual image as a numpy array(grid) and classifies it using that and part 3 uses features and feature detectors along with max pooling to classify data. The vgg6 network learns features and layers directly from the data. Since CNNs are better used to classify visual/spatial data such as images, the vgg6 network will naturally perform better. In addition, transfer learning allows us to use two methods(CNN AND a fully connected neural network) which will naturally increase our accuracy.