Luna McBride

CSCI 3202

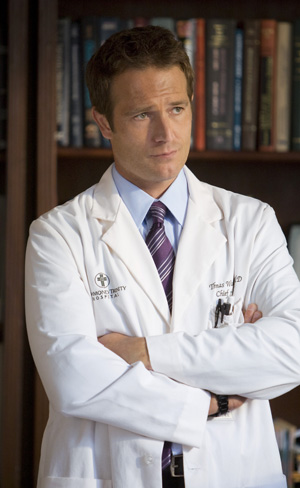
**Assignment 5: Deep Learning**

**Part I**

Preprocessing is extremely important for faces for consistency reasons. Deep neural networks for classification go by features, such as eyes, mouths, and noses on a face. If there are not clear features to go by, it is very likely the network is going to have trouble identifying. If the image is a forehead with the beginning of eyes on the bottom, how can a network identify a face from it when there is no face?

The next important factor is resizing. Even if everything is all cropped correctly, images still need to be even to remove all variance in the data. If we have a face that is, say, 60x120 instead of 60x60, the eyes will take up more room on the image, for example, and thus making different unjust variation in the eyes, making it harder for a network, which does not think how humans think in terms of these classifications, to distinguish where the eye is supposed to be.

Now with those ideas commented on, here are some data points:

Of course, there are bound to be some good ones like this one, Vartan39. The face is cropped well and the features are clear. It probably would have been better to have a straight forward shot, however, since life is not so straight forward and all other key variants are eliminated, this is the best case.

This one is Harmon76. The cropping managed to get the right approximate area, but this is not a face; this is a forehead. There were a surprising amount of forehead images in the cropped set, but I thought this one showed best why we do not want a forehead.

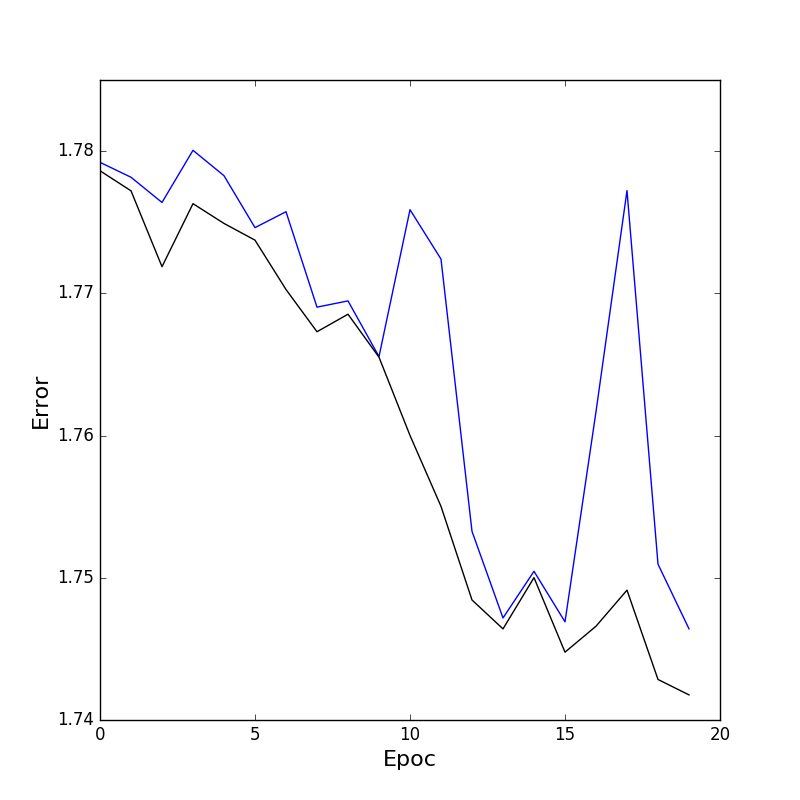
 

And then we have Bracco77, which is not even close. However, it is very funny it zoomed in on the HE for the picture of the actress laughing, despite being no help at all.

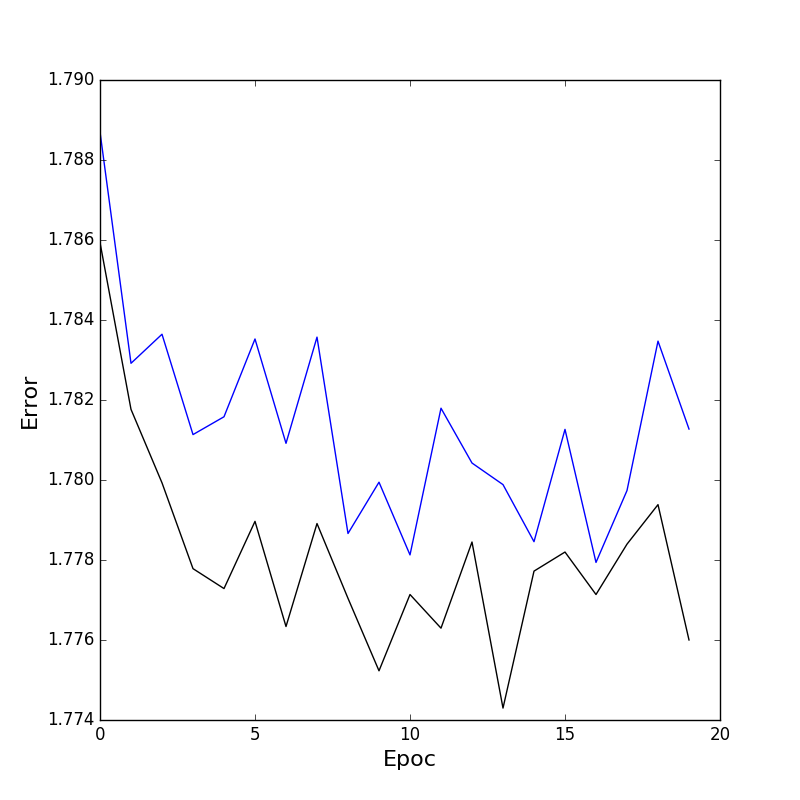
UPDATE: after installing the new getData file, lots of the pictures had changed. The same themes are present and I still love the HE, so I am going to leave these how they are. Just note the cropped are greyscale now because of part 2, and more faces are present than before because of that change.

**Part II:**

The model I used was a Keras Sequential. Data was split with a sklearn train\_test\_split, having one for each loop in nested validation. The model had a hidden dense layer, having 32 nodes per layer and initializing based on the length of the array of images. This was achieved using a sigmoid and a softmax, as relu created a situation with loss=NaN.



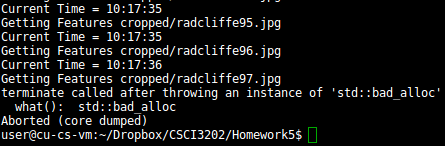
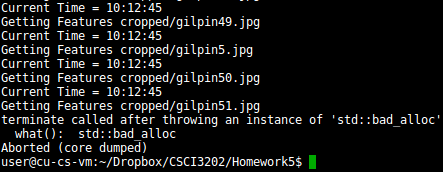
This graph uses the images as inputs for loss, overall lowering in error for each epoch (with a little wiggle room)



This is it using the given vgg output as input. It seems the it is doing a bit of overfitting, but that would need to be seen under more epochs to be sure.

**Part III**

I ended up choosing block 3 as input. I had tried to get block 1 and block 2 to work specifically to speed up the usage, which came up in these:



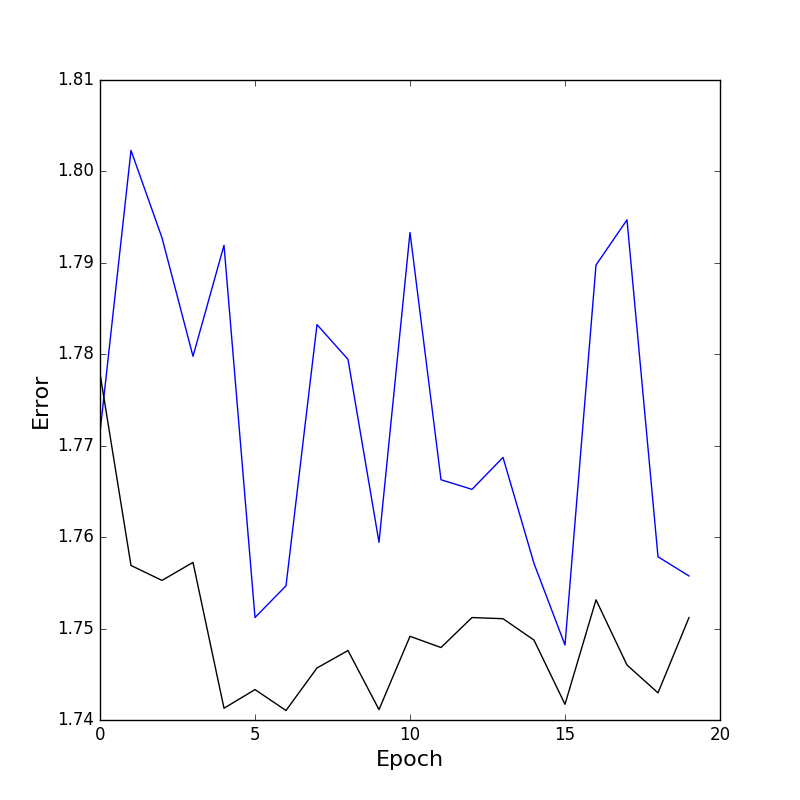
Of course, this did not happen with 3, and since I needed to be as fast as I could so that I could get this done, I chose block 3.

As for the difference between parts 2 and 3, the block 3 seemed to jump around more. I had been scouring the internet because I could see that the accuracy was low and the error decrease was slow. I had realized I had swapped the placement softmax and sigmoid, just to see that having them in the supposed right places made the output more erratic. With the activations put back in place, the graph looked really similar to the vgg in part 2, which I just realized was not for part 2 to do and I had just misinterpreted “Evaluate your final model on the test set” in the instructions for the part 2. I will just leave it there because it is an interesting comparison. (Though I will still use part II for part IV because it is much faster to test with).

**Part IV:**

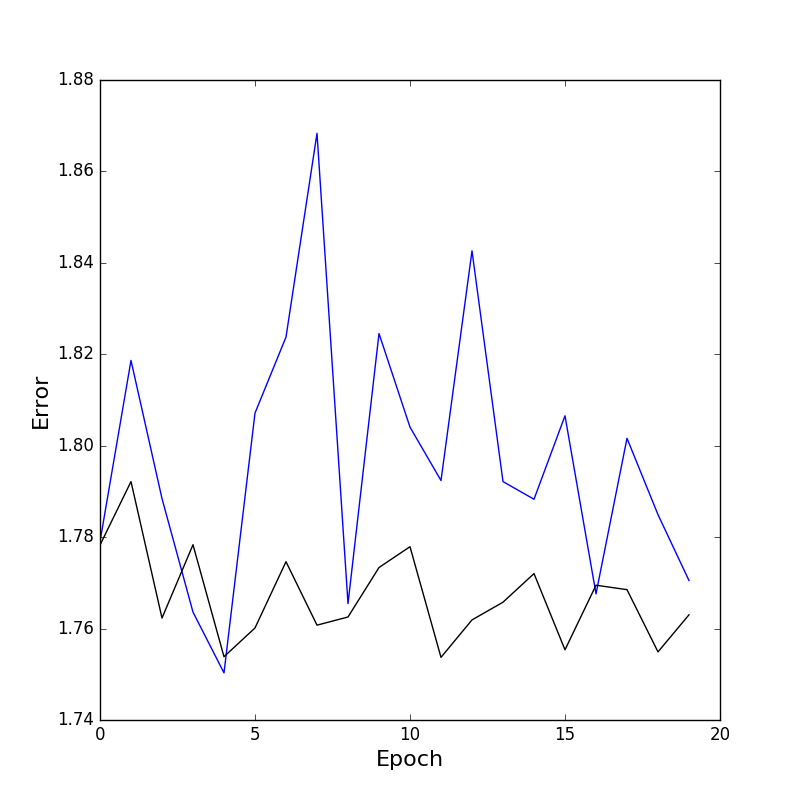
Note: a separate train faces function (just by adding a 2 to the name of the previous one). It has been left commented out, but you can test with it too.

To start, since you cannot really get lower in hidden layers than 1, the first experiment, and thus the baseline, was 3 hidden layers, all with node number 32 and no dropout:



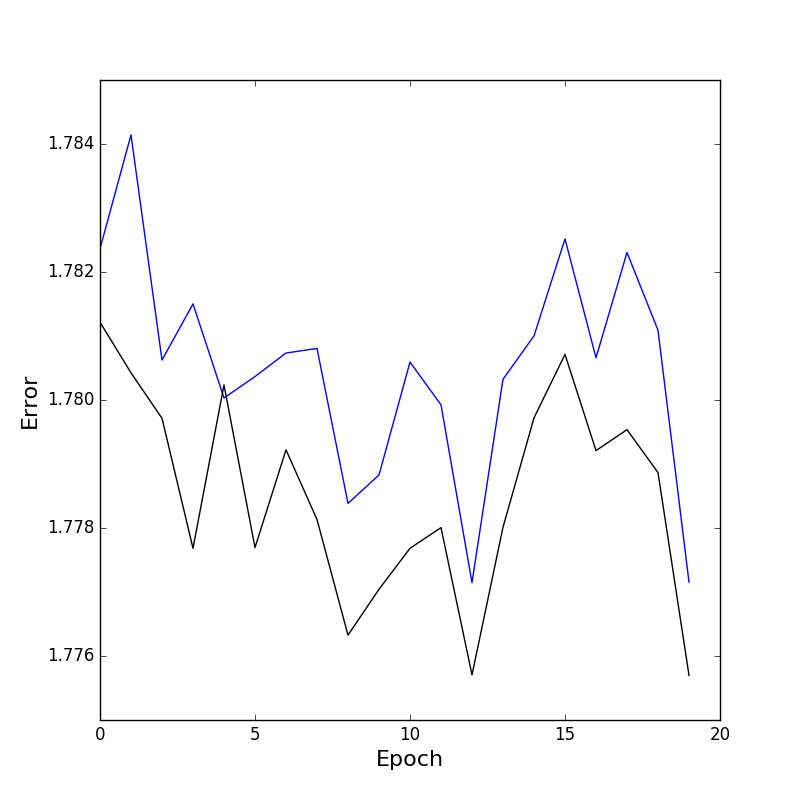
Of course, changes are not smooth as they never are until you get to number of epochs above several hundreds. Overall, it does have a downward trend, but there were not enough epochs to say if it is an underfit or an overfit (with accuracy how it was, it might be an underfit, but we cannot be sure.

Next comes 3 layers with 512 nodes per layer, that being the big change:



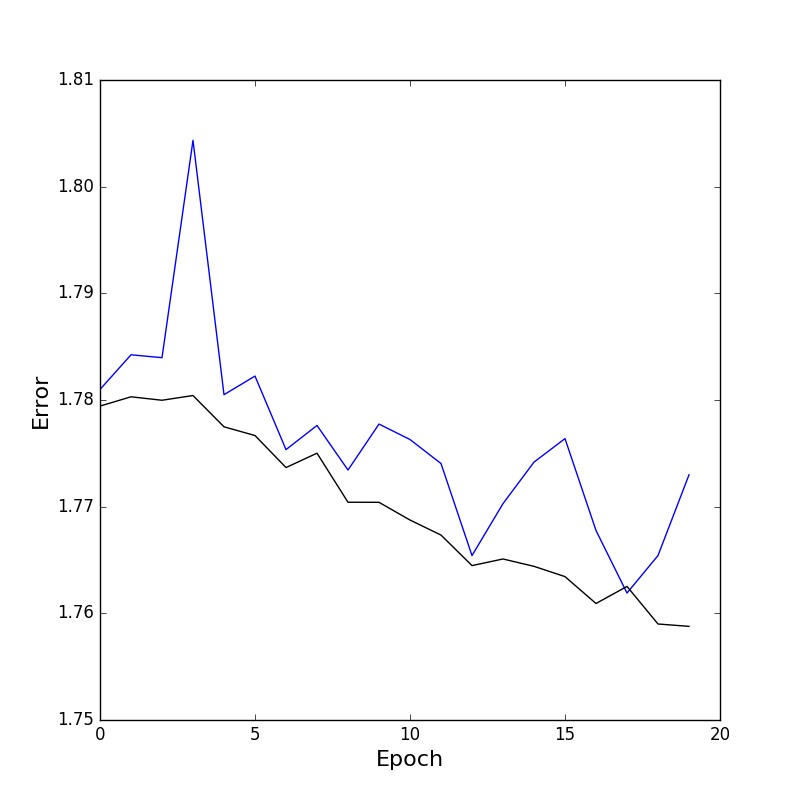
The data is very much not smooth, but I can say that I had noticed accuracy slowly going higher while it was going along. It was too slow to represent in a single screenshot from the terminal and I did not think to screenshot that data until now, but the accuracy was better. There is also a trend going downward as seen toward the end.

The next graph was similar to the previous 3, just with 2 nodes per layer:



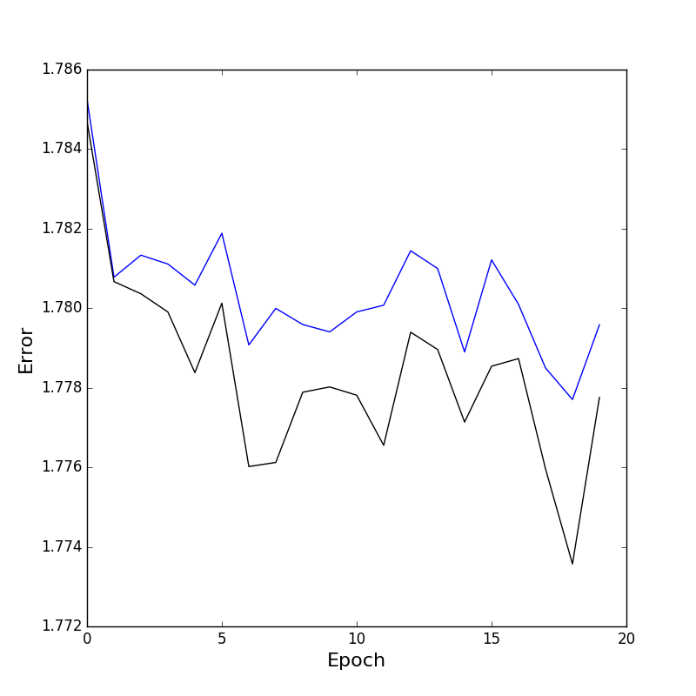
Of course, it is a lot easier for it to generalize with fewer layers with both lines matching up better. Though the overall trend is downward, that peak around epoch 15 was interesting to see, as it could imply issues with the model rather than how it is being handled, but I do not know for sure.

Next, I wanted to look at dropout, so I went for a dropout of half:



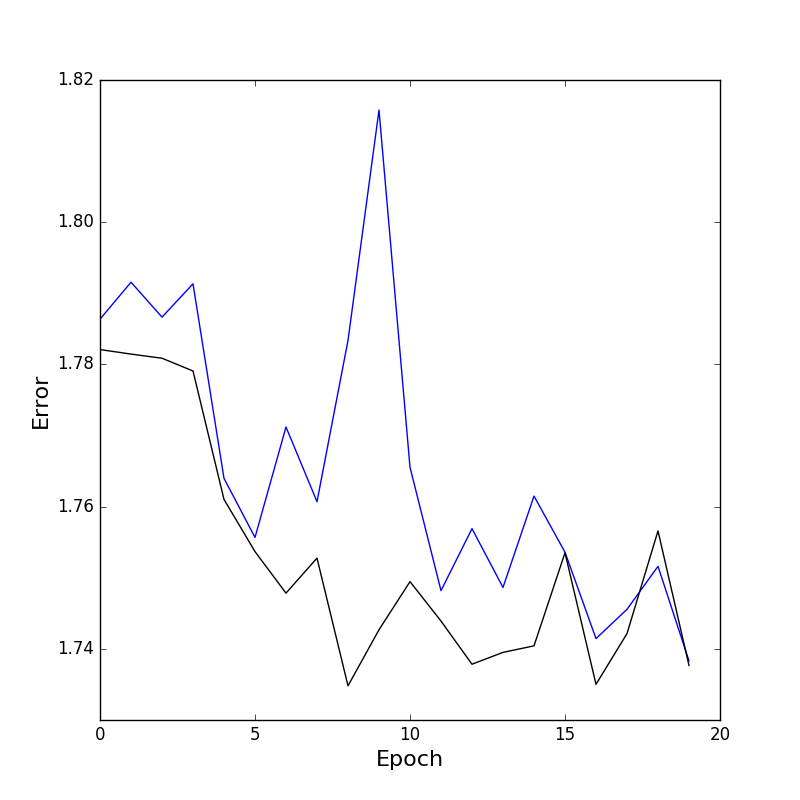
Funnily enough, the one with half dropout has a strong downward trend, almost linearly. I did not expect the dropout to be strong enough to have this strong of an effect, but it is interesting nonetheless.

The next was a dropout of 0.9:



The error has been limited to between 1.78 and 1.77, which is a lot lower change than some of these. However, that would be expected if 90% of your nodes were unused at a time. It also seems to level out more than the others, but once again, 90% dropout. I did like the quick runtime, though.

The last dropout one was 0.1 dropout:



This one definitely has an overall downward trend, capturing more of the data. However, I just noticed that all of them have pretty low drops in error. For what I had given, this is quite nice, but still.

NOTE: After looking at this data and some piazza posts, I seem to have a lot more data than other people. Looking through my cropped photos, it looks like I have several pictures that are repeated. I think something might have happened in the original downloading processes, with using the original getData file and then using the new one as soon as that one came out.