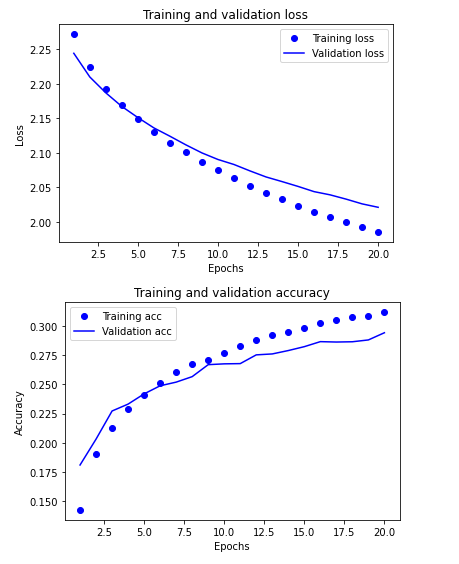
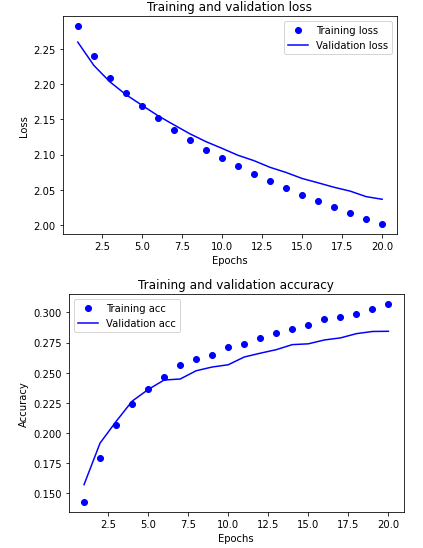
**Problem 1**

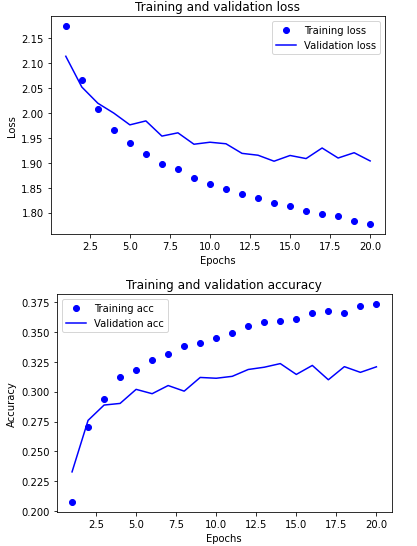
**Figure 1: Baseline**



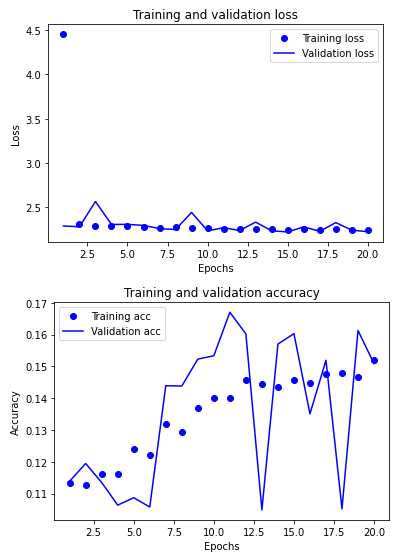
**Figure 2: Drop out attempt 1 (0.01)**



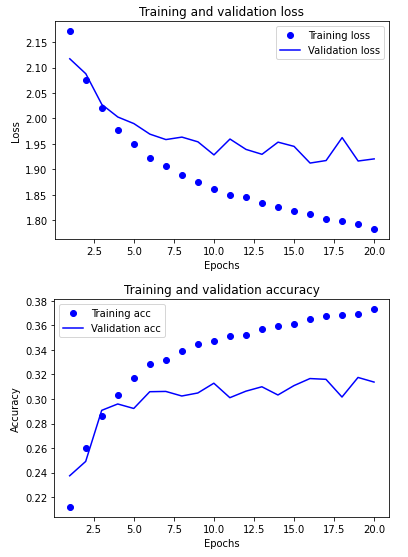
**Figure 3: Drop out attempt 2 (0.1)**



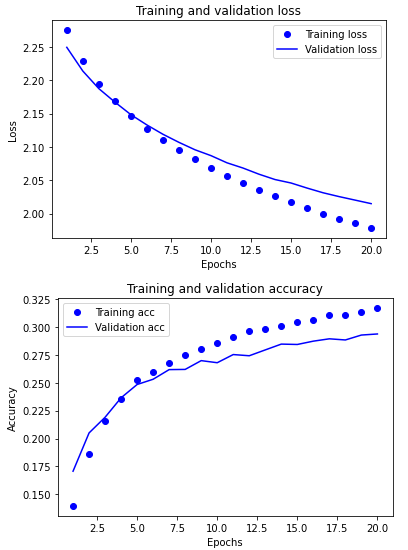
**Figure 4: Learning rate attempt 1 (0.05) NOTE: do not go up with learning rate**



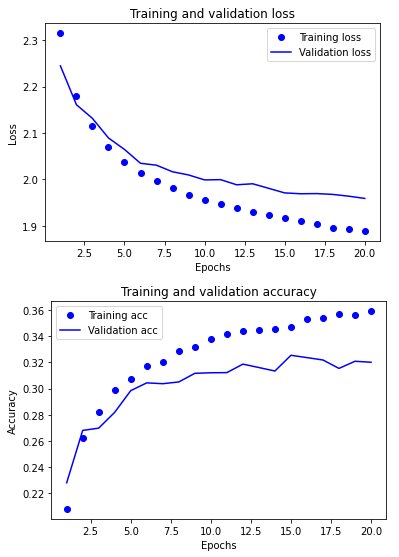
**Figure 5: Learning rate attempt 2 (0.001)**



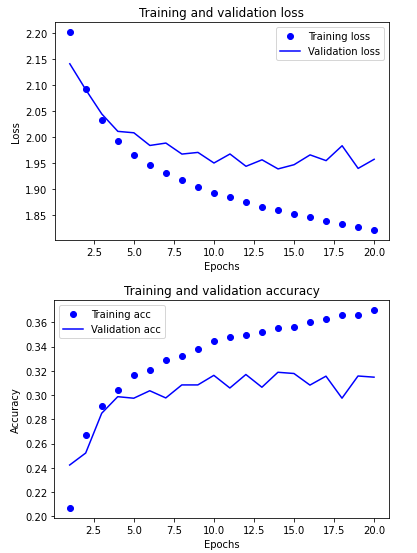
**Figure 6: Learning rate attempt 3 (0.0001)**



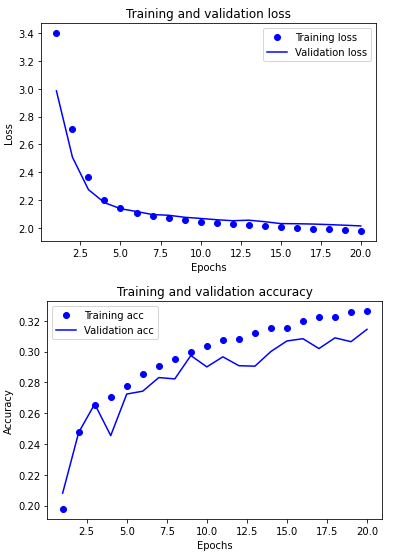
**Figure 7: Weight decay attempt 1 (0.01)**



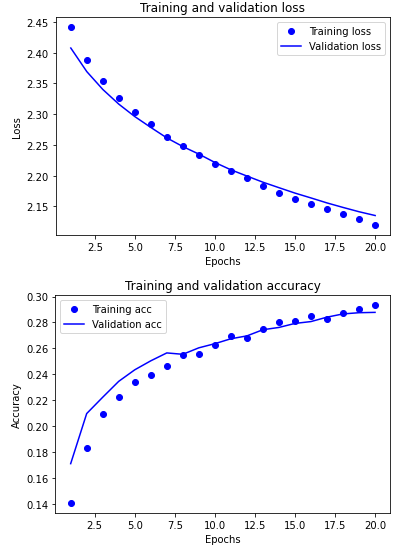
**Figure 8: Weight decay attempt 2 (0.001)**



**Figure 9: Figure Weight decay attempt 2 (0.1)**



**Figure 10: All of it together with best values (drop out 0.1, learning rate 0.0001, weight decay 0.01)**



a) From the figures 1-9 above I demonstrate the results of the different values for the feed forward network. In figure 1 I show what the results are for the baseline network that just uses one 64 node dense layer and one output layer. Its accuracy peaks around 27% which is not great for performance, but the validation accuracy curve is close to the training accuracy curve showing signs of over fitting around the 5th epoch. When comparing this to figures 2-9 we can see the effects of changing the learning rate, dropout rate, and l2 weight decay. For the dropout rate I noticed that by increasing the dropout rate for the model the accuracy of both the validation and training data increased but there where heavier signs of overfitting with the validation accuracy being much smaller then training accuracy past the third epoch, seen in figure 3. I ultimately decided the increase in accuracy was good for the model and chose the higher drop out rate of 0.1 for the second part of problem 1. When changing the learning rate, I noticed two things, the first is that high learning rates are devastating to the model and destroy the model’s ability to learn effectively as seen in figure 4. The second was that a lower learning rate reduces overfitting on the model significantly, so I chose the lowest learning rate I tested with for the second part of problem 1, as seen in figure 6. For the weight decay I learned that the best result was some where in between the highest and lowest value I tested, with the best results being displayed in figure 7 with a weight decay of 0.01 that made the validation accuracy less noisy and less overfitted. I used this weight decay for part 2 of problem 1. One important thing to realize when analyzing the graphs for all these tests is that the network has a bad accuracy basically across all of the tests. This is because we are using a feed forward network to do image recognition with very few layers, I believe that the model is not ultimately designed to handle this type of task but does show how different variables can affect its performance very easily so the low accuracy can ultimately be ignored for this problem as long the model shows signs of improving and not over or under fitting.

b) The results of part b can be seen in figure 10 which only slightly increases the accuracy to about 29% but shows massive improvement in fixing any overfitting that was seen in the baseline model seen in figure 1. This reduce in overfitting and increase in accuracy show that it is an improvement from the original model, but like the original model it still does not perform well. I believe this to be because the model is ultimately not designed to handle image recognition. The model is a feedforward network with very few layers and normally image recognition networks require many more layers and tent use layers like convolutional 2D layers. Because it is trying to look at images of cloths and try and determine what type of cloths they are, with 10 classes to chose from, the fact that it can get above a 10% accuracy for being a simple model that is not optimal to perform this task makes it somewhat successful. Ultimately it would not actually be useful in labeling different images of cloths though.

**Problem 2**

**Model 1**

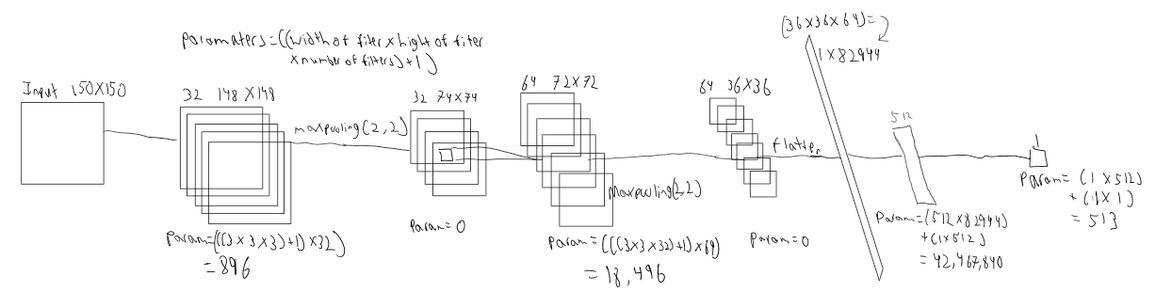
For model 1 I created a model that had 2 conv2D layers, 2 maxpooling layers, and 2 dense layers. The drawing of this model can be seen below in figure 11. In the drawing I show how the number of parameters in each layer are calculated with the equation used on each conv2D layer being **number of parameters = ((width of filter \* height of filter \* number of filters) + 1**) and the equation for the last 2 dense layers being **number of parameters = ((current layer \* previous layer) + 1 \* current layer)**. With the number of parameters per layer calculated as.

**Figure 11: drawing of model 1**

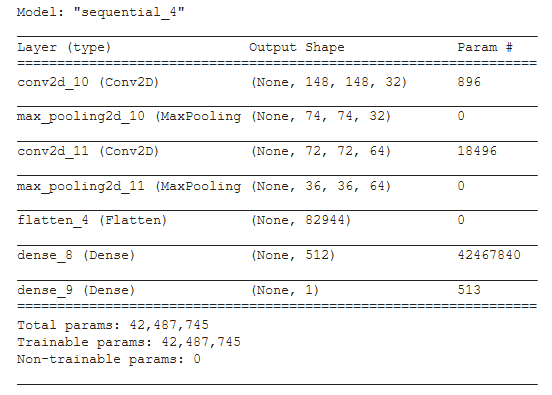
* Layer 1: conv2D (((3\*3\*3) + 1) \* 32) = 896
* Layer 2: maxpooling = 0
* Layer 3: conv2D (((3\*3\*32) + 1) \* 64) = 18496
* Layer 4: maxpooling = 0
* Layer 5: flattener has shape of 1X82944
* Layer 6: Dense (512\*82944) + 1 \* 512 = 42467840
* Layer 7: Dense (1\*512) + 1 \* 1 = 513

These calculation are confirmed to be correct in the model.summary() result shown bellow in figure 12.

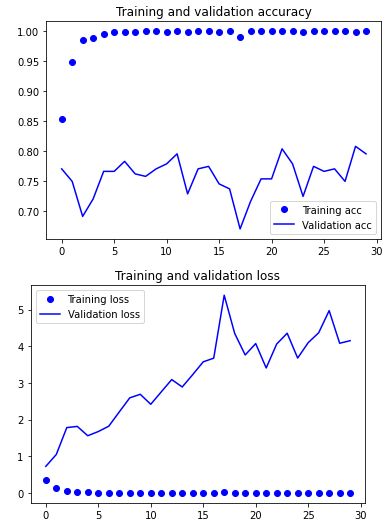
The model was then tested and validated with the results being displayed in figure 13. These results show that model was able to get a maximum peak accuracy of 80 % for validation, but had trouble improving validation over time and had a large amount of overfitting with the validation accuracy sitting around 20% below the training accuracy.



**Figure 12: model 1 summary**



**Figure 13: model 1 training and validation graphs**



The model results seen in figure 13 show that model 1 has massive overfitting which could mean that the parameters are to focused on the training data and are getting information that is not necessary for the actual decision making needed in the model.

**Model 2**

Fore model 2 I had to expand the layers in model 1 to include 2 more conv2D layers, 2 more maxpooling layers and twice the units in the dense layers. Training and validation for this model is shown below in figure 14. The results of this graph show a decrease in noise and overfitting with more consistent validation accuracy results hovering around the 80% accuracy range per epoch. This shows that adding more parameters to the model has allowed it to learn more features about the data that are less specific the training data and more universal allowing the validation accuracy to perform better.

In terms of parameters there are some more ways of decreasing the number of parameters in the model with out adding more layers to the model. These ways are listed below.

Input layer

* To decrease the number of parameters in the model from the input layer I can change the size of the filter to be smaller than what it was before. This will ultimately give less parameters from the input layer and decrease the number of parameters in the model.

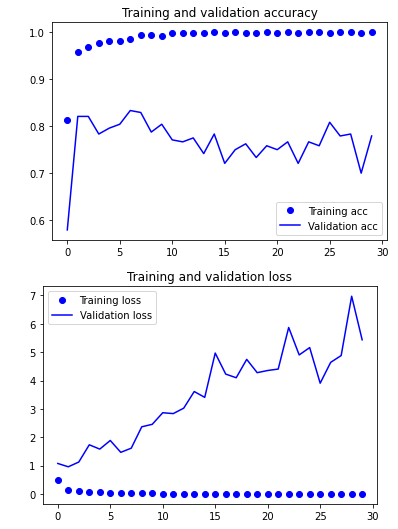
Maxpooling

* For the maxpooling layers I can increase number of pixels being pooled. So instead of pooling by a size of (2, 2) I can pool by a size of (3, 3) which should reduce the overall number of pixels in the end of the training before being flattened which will lead to a smaller sized flattening layer and less parameters in the model overall.

Dense layers

* To decrease the number of parameters from the dense layer it is fairly strait forward. All I need to do is reduce the number of nodes in the layer. This will reduce the number parameters because each node has a parameter that needs to learn.

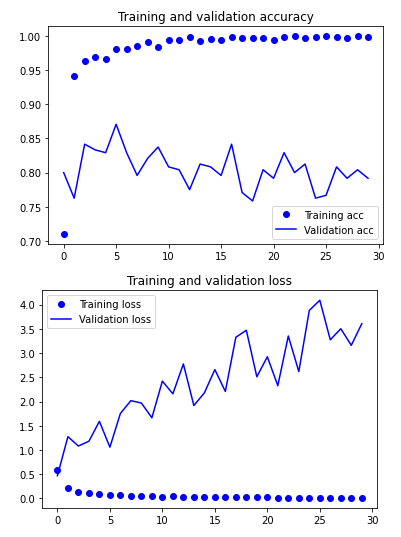
**Figure 14: model 2 training and validation graphs**



**Model 3**

For model 3 I added data augmentation, a dropout layer, and weight decay to model 2. With a data augmentation that rotated, shifted, sheared, zoomed, and flipped images this should make it so that no training data image seen during training should be the same or similar, thus making every image more unique lowering the possibility of the model learning specific features from the training data. This should reduce overfitting with the possibility of lowering accuracy in the model since some images might be warped in a way that ruins the image. For the dropout layer I had a drop out percent of 0.5 which means half of the nodes where being dropped at the end this should fix overfitting by making it so that nodes at the end of the model have a higher chance of not learning the same features from the training data. Lastly, I had a weight decay added to the dense layer of 0.01 at the end of the model, this should reduce overfitting by making the weights change less over iterations while training thus making the features learned by the training data less significant to the model. With all this said the tested and validated result of this model is shown below in figure 15 with the results of the graph showing a reduction to overfitting via an increase in the validation accuracy which has gotten closer to the training accuracy. This proves the above methods indeed reduced overfitting in the model.

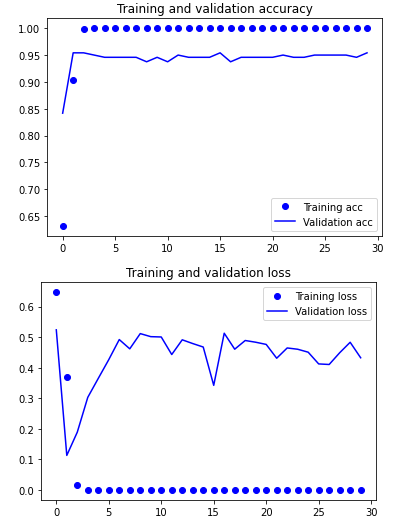
**Figure 15:** **model 3 training and validation graphs**



**Model 4**

Was a network build off the base network of VGG16 with the firs 4 blocks of the model being frozen. The results of this model where then validated and are shown in the graphs of figure 16 below. This is by far the best model in terms of performance and overfitting. With a validation accuracy of about 95% and a very small difference in training and validation accuracy this model performs far better then any other model used problem 2. By the second epoch it already gets better accuracy then the other models, but it never gets to a perfect accuracy. This can be seen in the following possibly miss classified image seen bellow. This image could have been misclassified in model 4 because of persons hair in the image. Because of the long hair of this person and the long hair of a horses main it might misclassify this image as a horse rather then a person since the model seems to be looking at the entirety of the image.

**Figure 16:** **model 4 training and validation graphs**



**Figure 17:** **an image that looks like it can be misclassified**

