Batu Arda Düzgün

Question 1:

a) Why do people use validation data?

It is usually used to tune the hyper-parameter of an algorithm, such as the learning rate in CNN; this can not be done with the test set because doing so would mean the test set is used in the training process, then it can't be used in testing for realistic results.

b) What is the difference between mean squared error and mean absolute error?

In mean absolute error, the error is the sum of the absolute difference between the prediction and the true value in each dimension; in mean squared error, the error is the sum of squares of the difference between the prediction and the true value in each dimension.

c) What is the main problem of using sigmoid as activation function in an artificial neural network (ANN)?

The main problem with using sigmoid as an activation function in an ANN is it can cause the vanishing gradient problem, which prevents nodes from further updating their parameters.

d) What does it mean to overfit your data model?

Overfitting means the model learns to give the correct answers to just the training data without actually learning the underlying logic; this causes the model to have very high accuracy on the training data while having a terrible accuracy on any additional data (test, validation).

e) Your input image size is 3x64x64. If you apply 3x3 convolution with input_channel=3, output_channel=6, padding=0, stride=2, what would be the size of the output?

The output size formula for one edge is $\frac{W - F + 2P}{S} + 1$, so $\frac{64 - 3 + 0}{2} + 1 = 31$, and there will be 6 output channels so the final size will be 6x31x31.

f) In the previous question, how many trainable parameters are there? (you should also consider bias terms in addition to weights)

The convolution matrix is 3 by 3, so each of them has 9 parameters, there are 3 inputs and 6 outputs, there will be a convolution matrix between each ones so in total there will be 9 * 3 * 6 weights for the convolution matrix, also there will be 6 biases for each of the output channels; adding up all gives 9 * 3 * 6 + 6 = 168 trainable parameters.

Question 2:

Data Loader:

In this part, to normalize and preprocess the data, I turned the images grayscale and divided each pixel's value with 255. 255 is the highest value an RGB channel can get, so this normalized the values between 0 and 1. Turning the image grayscale is a common preprocessing technique. It decreases the computational complexity making the images faster to learn. Also, it can clear unnecessary noise in the images brought in by the colors. Because we don't have a lot of images to train with, I used it.

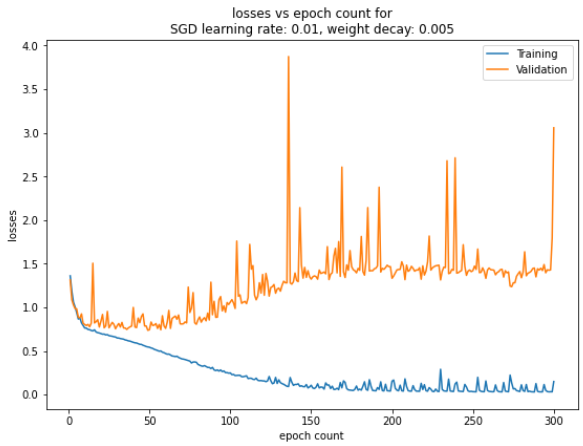
Model Implementation:

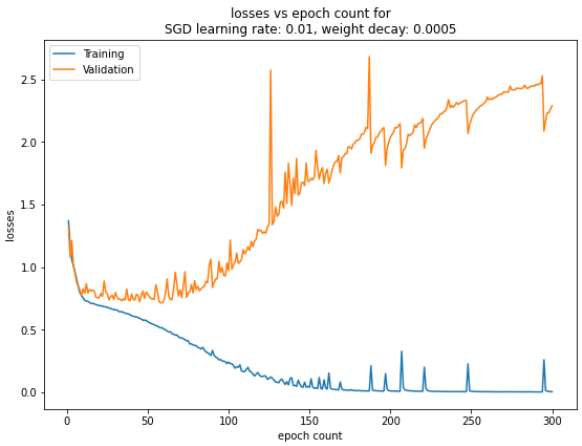
The last linear layer has 7744 input neurons and 4 output neurons. Because I am doing multi-class classification, I used softmax for the activation function of the last layer. However, the softmax function is not written in the model because, as the criterion, I used the cross-entropy loss, and the "nn.CrossEntropyLoss " function does a softmax to its inputs. So, if I added softmax to the end of my model, my results would be wrong as there would be 2 softmax's at the end.

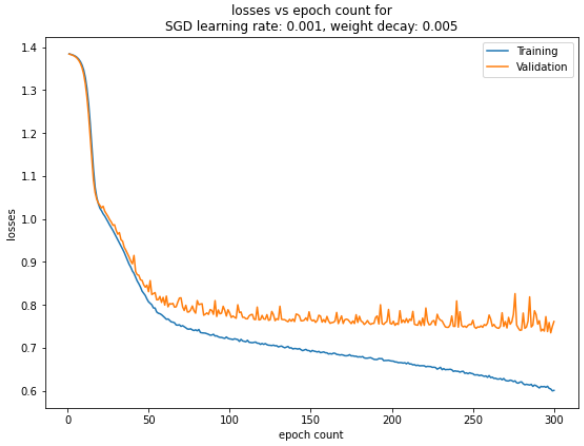
Stochastic Gradient Descent

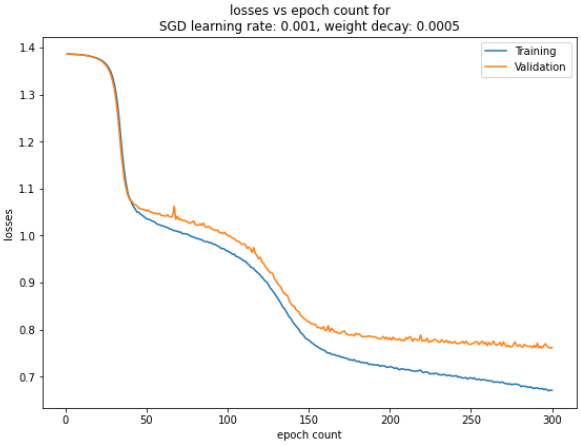
Training with SGD:

I tested the SGD optimized model with 6 pairs of learning rates and weight decay parameter combinations. I picked the best one based on their final loss value on the validation set. Loss is more meaningful than just accuracy as loss contains the information of how confident our model is in its decisions while accuracy only tells the number of correct decisions. As an example, the models for learn rate: 0.01 are severely overfit. However, their accuracy scores are pretty high, showing how the accuracy scores can be deceiving.

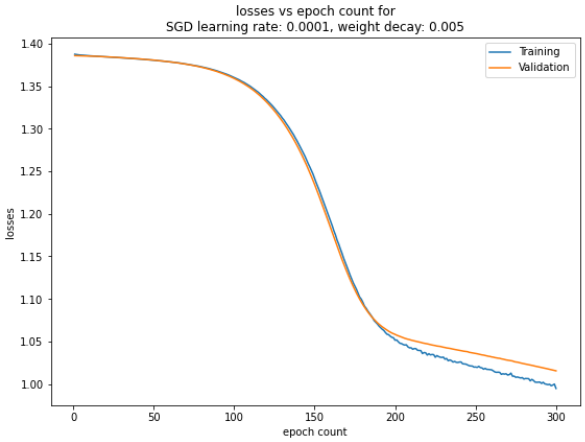


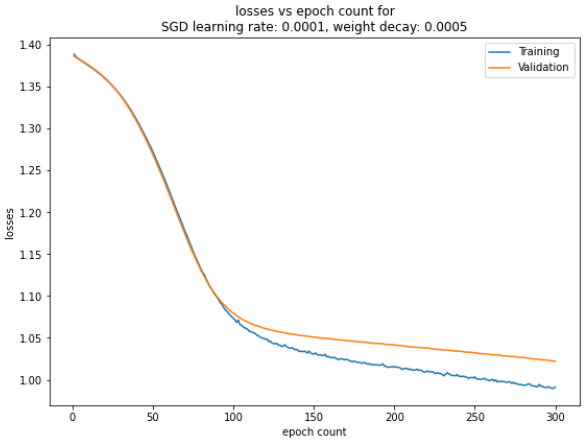


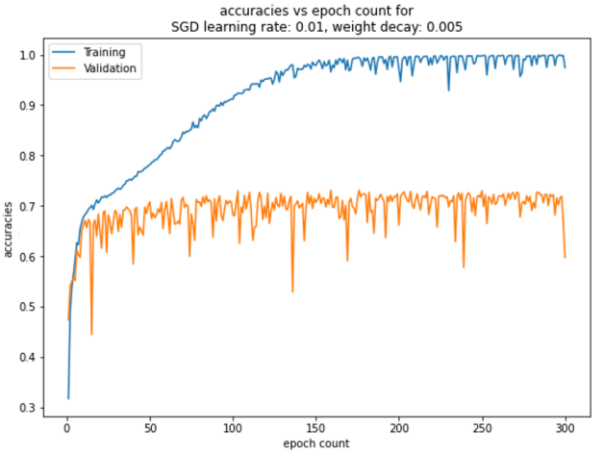


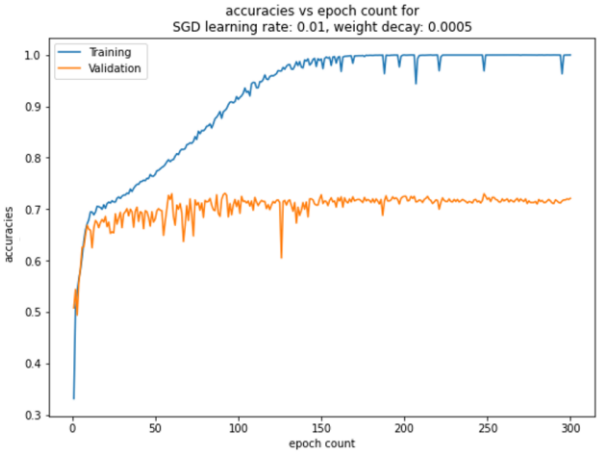


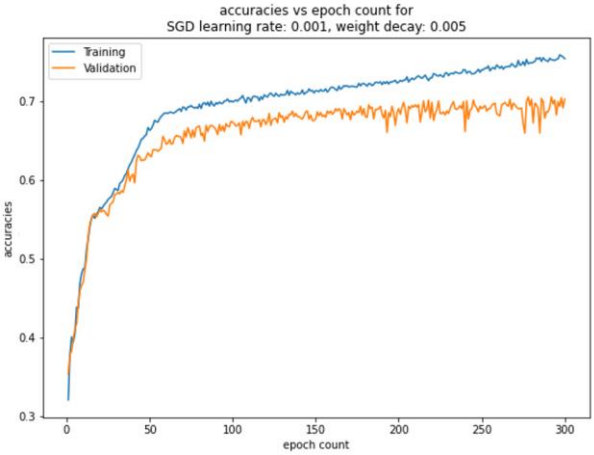
Batu Arda Düzgün

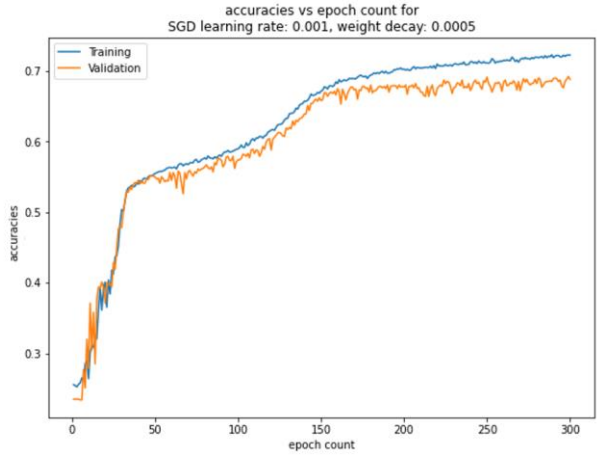


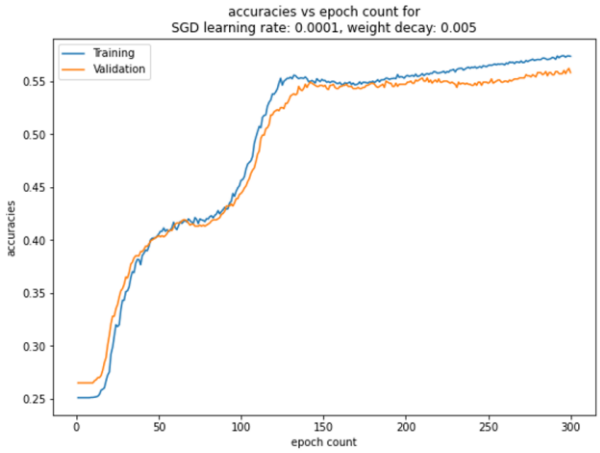


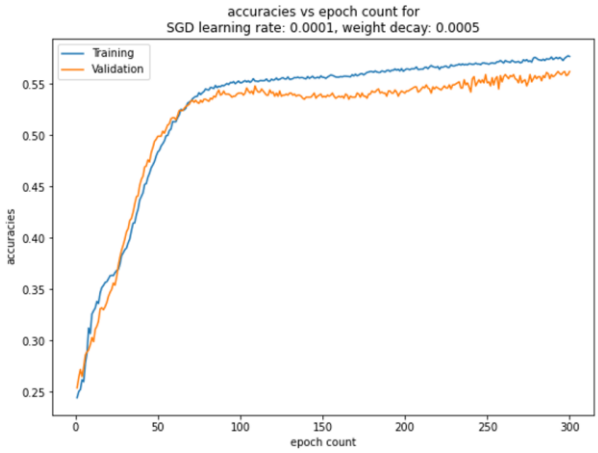








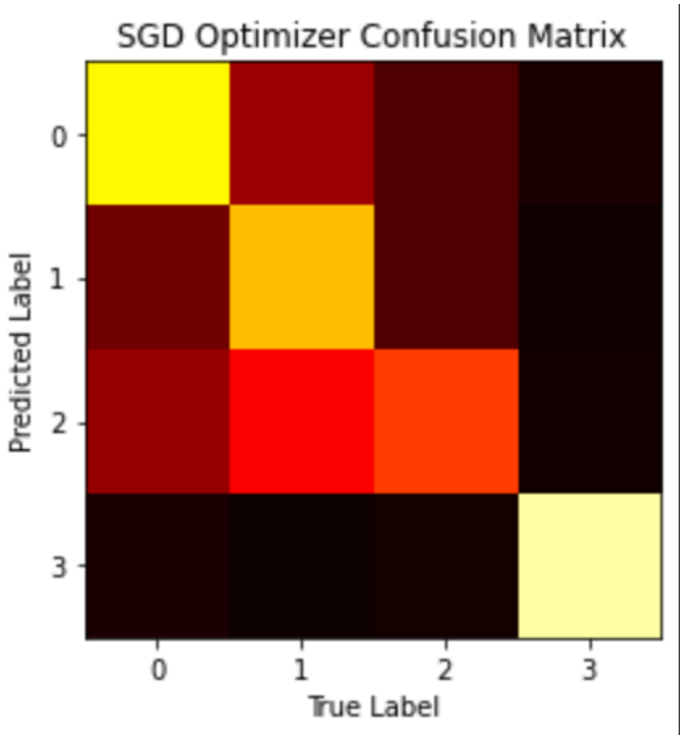




Test with SGD

The print output of the code and the confusion matrix is given below:

```
Mean Loss: 0.7511725425720215
Mean Acc: 0.688
Mean Macro Precision: 0.7003625497192534
Mean Macro Recall: 0.6937548022265857
Mean Macro F1 Score: 0.6874833281019856
```

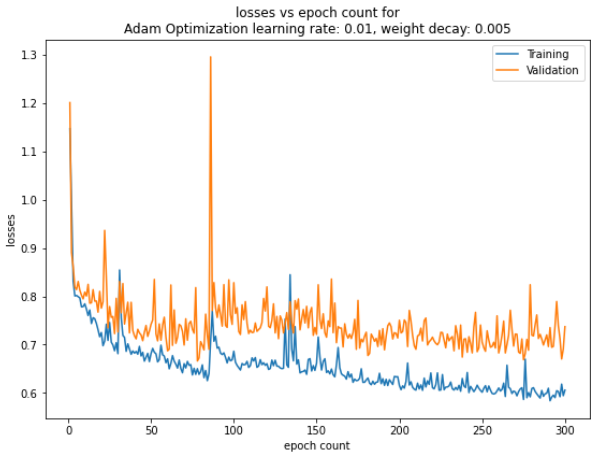


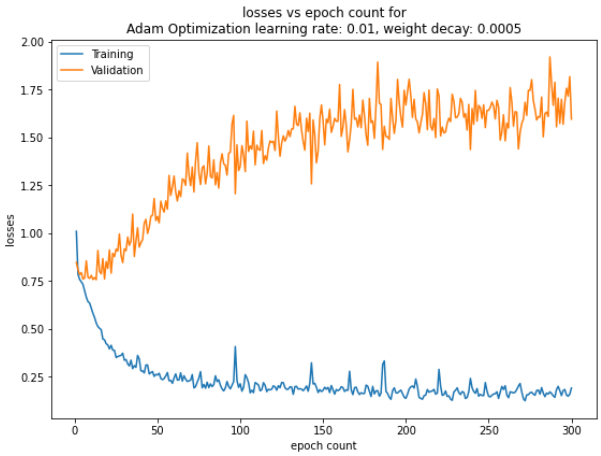
Batu Arda Düzgün

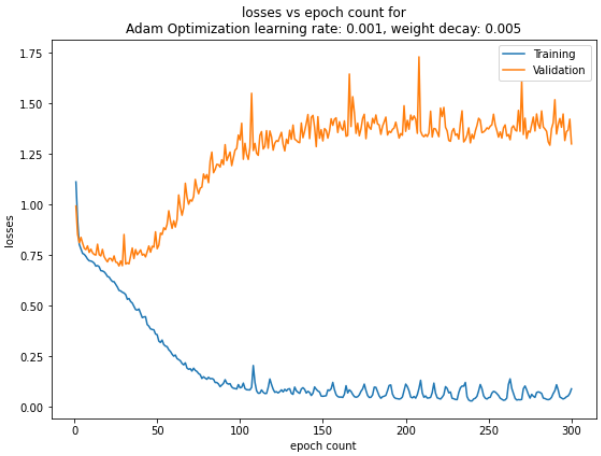
Adam Optimizer

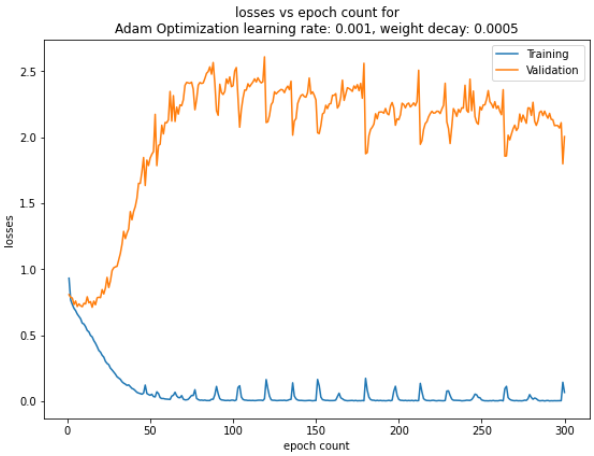
Training with ADAM

Again I tested the Adam optimized model with 6 pairs of learning rates and weight decay parameter combinations. Once again, I picked the best one based on their final loss value on the validation set. Loss is more meaningful than just accuracy as loss contains the information of how confident our model is in its decisions, while accuracy only tells the number of correct decisions. For example, the models for learn rate: 0.0001 are severely overfit. However, their accuracy scores are comparable to the best model's score; this shows how the accuracy scores can be deceiving.

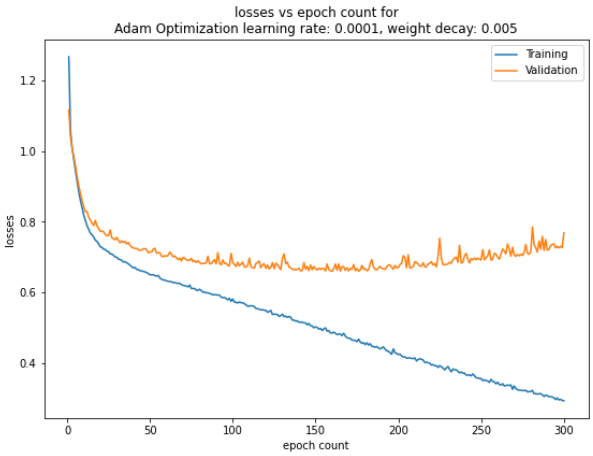


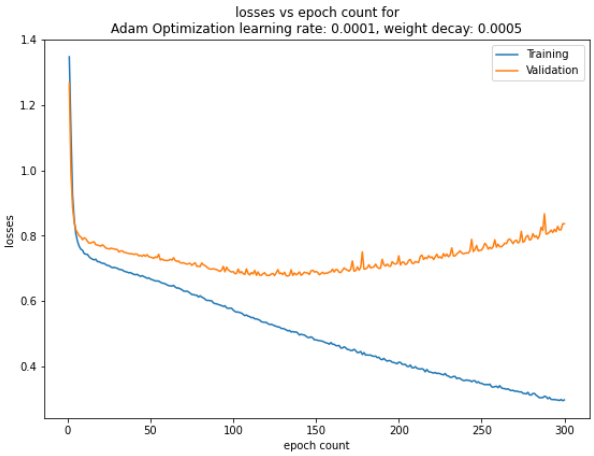


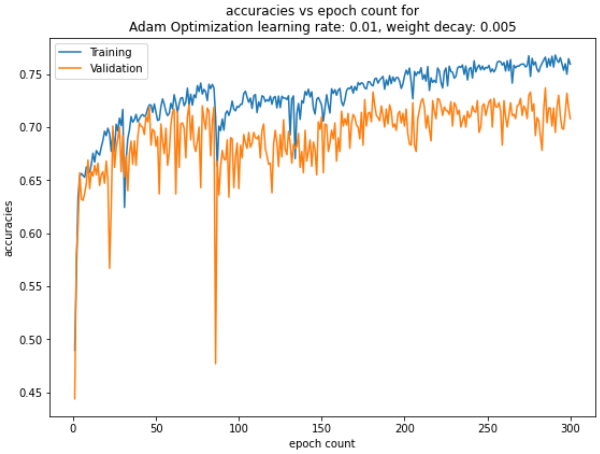


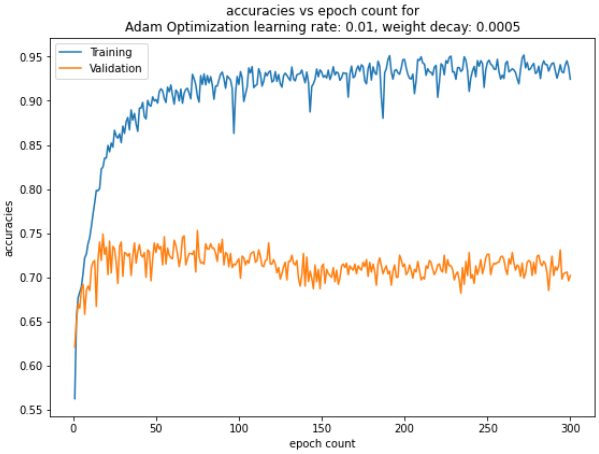


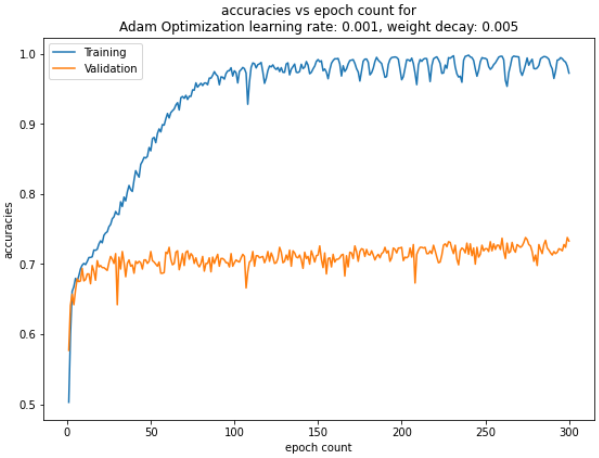
Batu Arda Düzgün

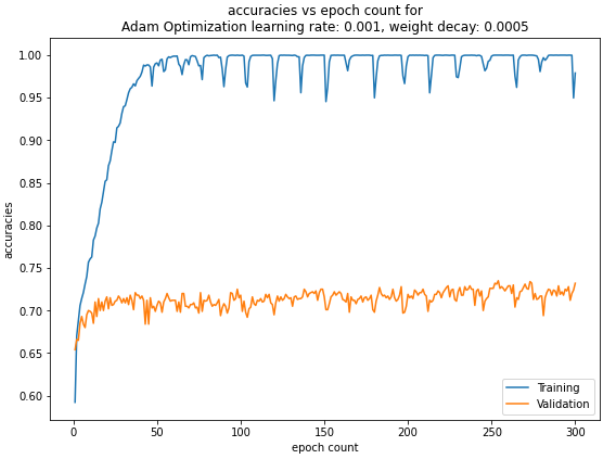


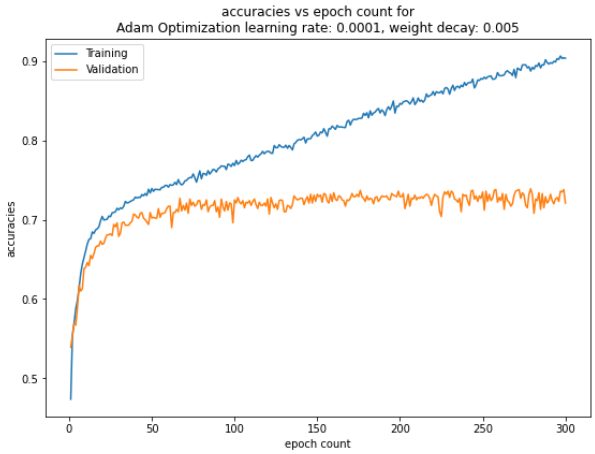


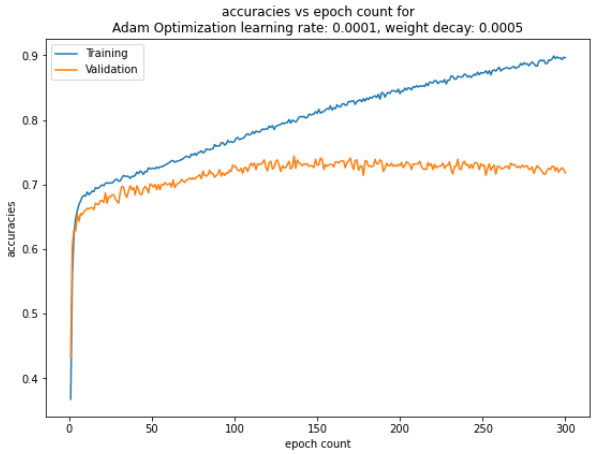








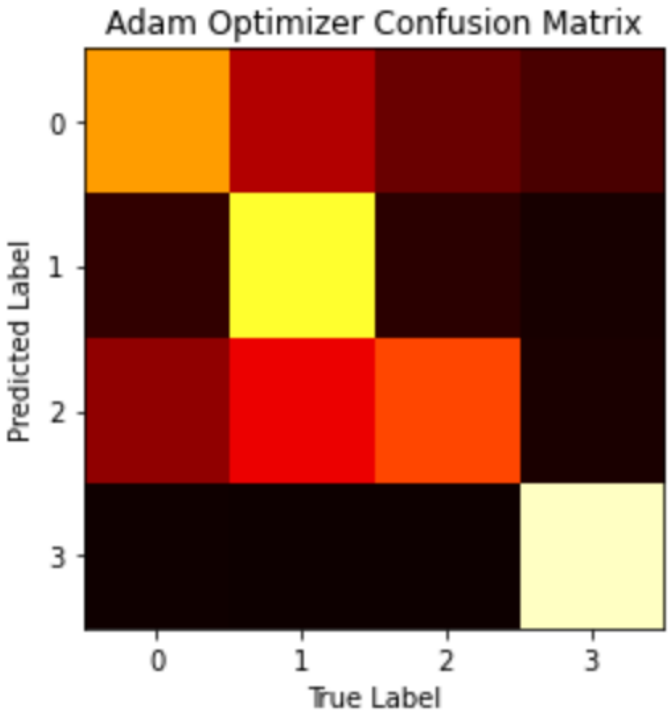




Test with ADAM

The print output of the code and the confusion matrix is given below:

```
Mean Loss: 0.7559253573417664
Mean Acc: 0.6995
Mean Macro Precision: 0.7086721612982326
Mean Macro Recall: 0.7112514355485788
Mean Macro F1 Score: 0.6930850312118912
```



The best learning rates and weight decay parameter pair for SGD are learning rate: 0.001 and weight decay: 0.005, with validation set loss of 0.7613. For Adam they are learning rate: 0.01 and weight decay: 0.005, with validation set loss of 0.7371.

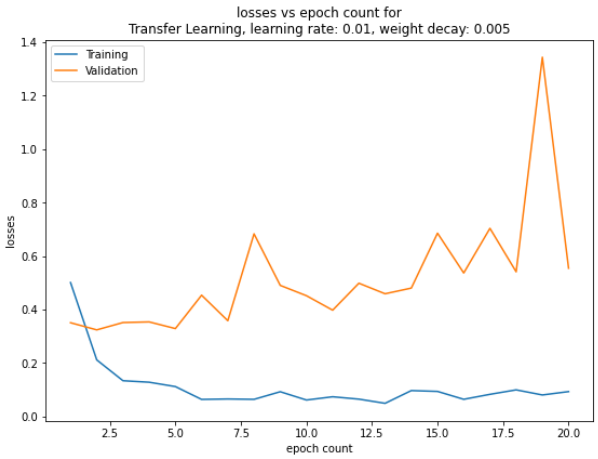
Most models are overfit, which can be attributed to our low number of training samples. All the performance metrics (accuracy, precision, recall, etc.) have very similar results, showing there are no tricks in the classification that make its accuracy look better than it should be. The last thing I want to note is the sea pictures (cluster 3) are rarely mistaken for other clusters, and other clusters are seldom mistaken as a sea pictures.

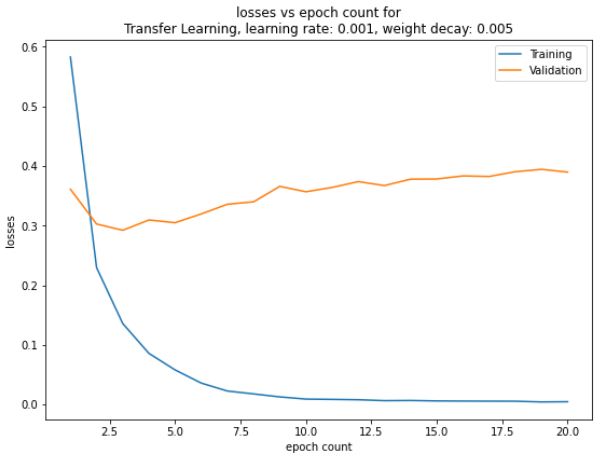
The models had similar run times, SGD took 45 minutes to run, and Adam took 50 minutes to run. Both of the algorithms spend most of their time in the forwards and backpropagation steps. These steps' length is in proportion to how deep our model is. Total run time and max_epoch have almost linear relations. Between the two algorithms, Adam is preferable as it dynamically updates its learning rate to achieve better results, while SGD is a crude and rigid algorithm.

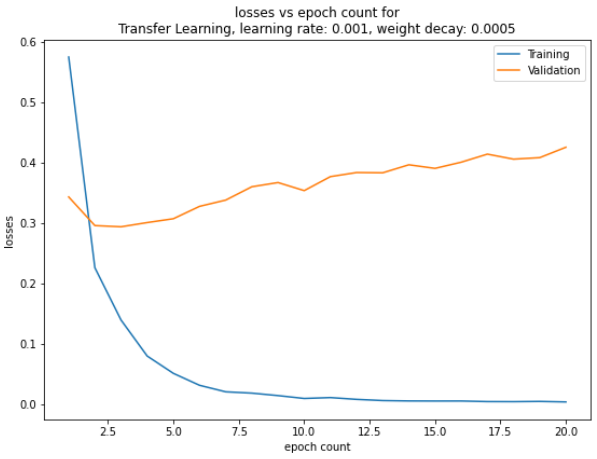
Transfer Learning

Training with Transfer Learning

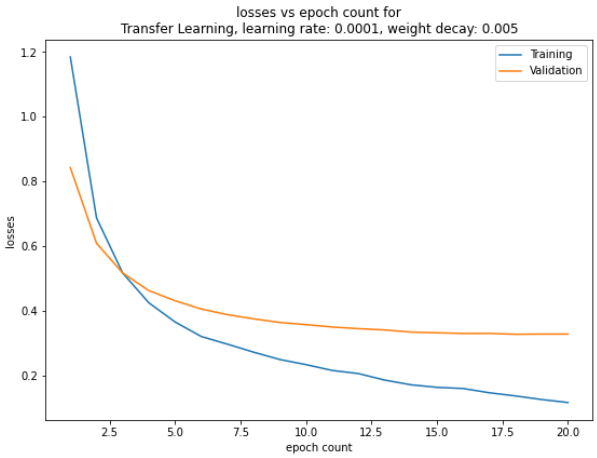
I tested the Transfer Learning model with 6 pairs of learning rates and weight decay parameter combinations. For all of them, I picked momentum as 0.9, as a commonly used value. I picked the best one based on their final loss value on the validation set. Loss is more meaningful than just accuracy as loss contains the information of how confident our model is in its decisions, while accuracy only tells the number of correct decisions.

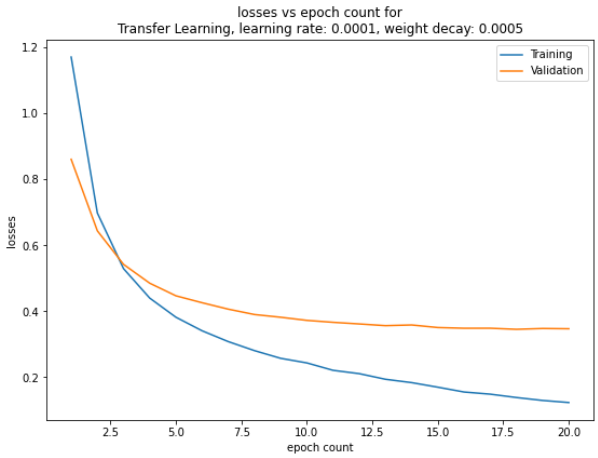


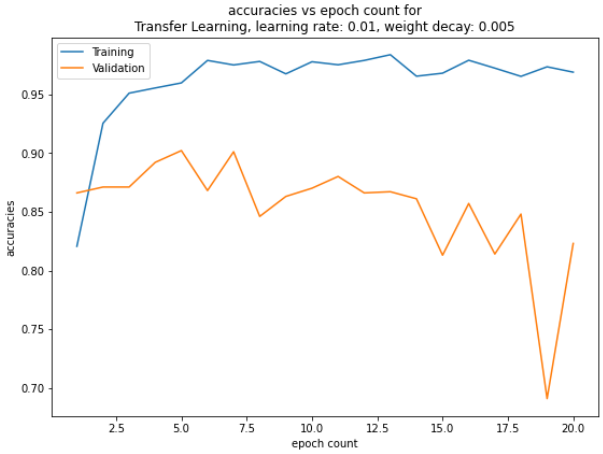


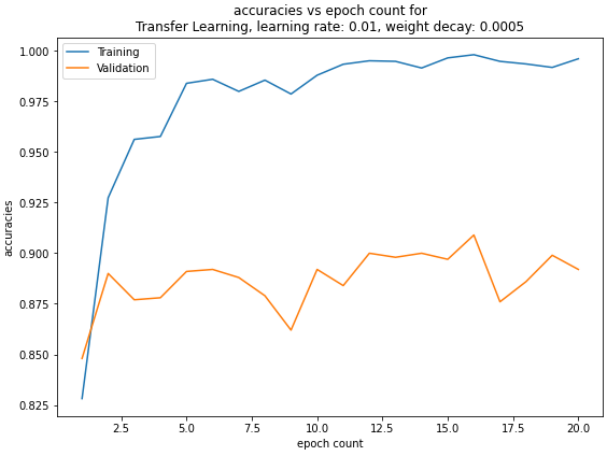


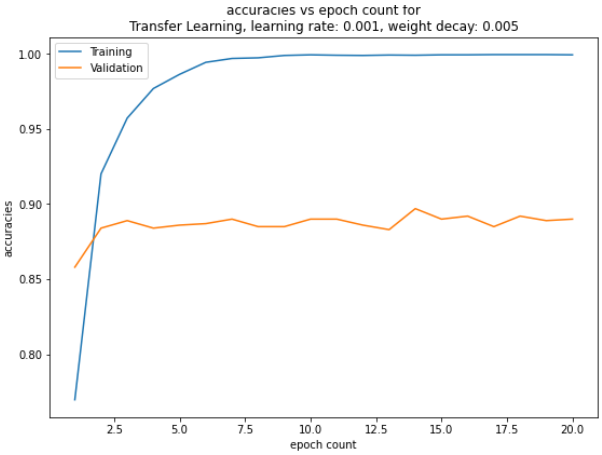
Batu Arda Düzgün

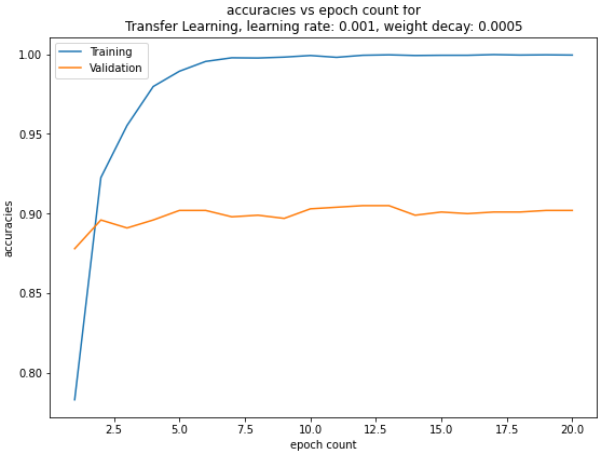


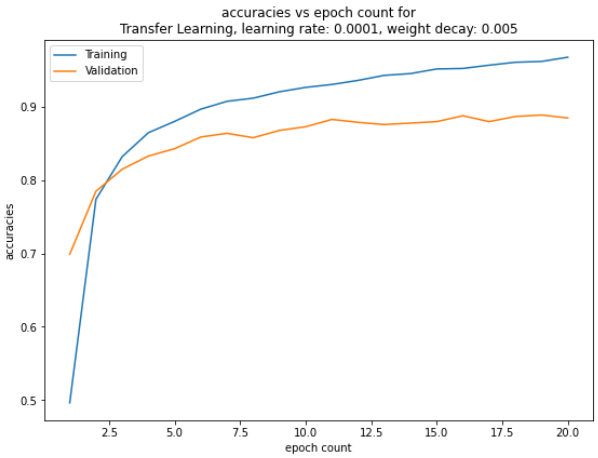


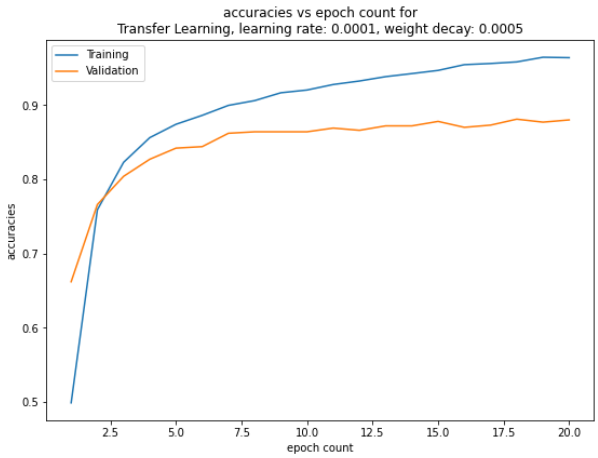








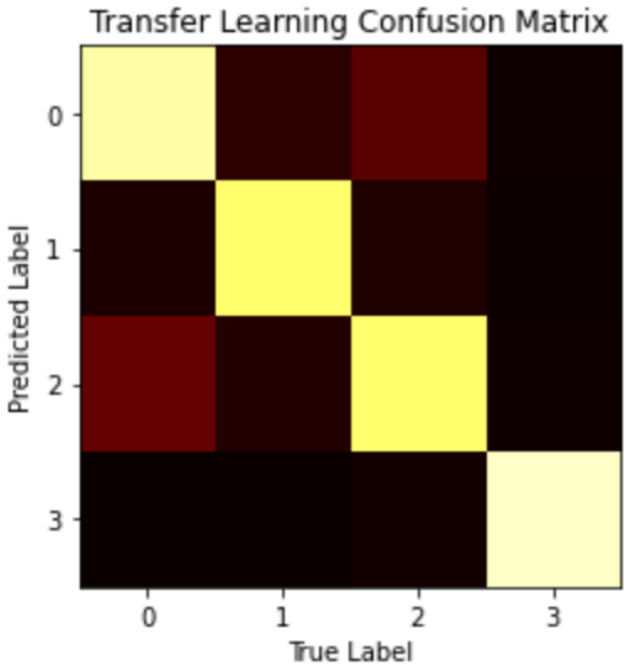




Test for Transfer Learning

The print output of the code and the confusion matrix is given below:

```
Mean Loss: 0.3093887269496918
Mean Acc: 0.891
Mean Macro Precision: 0.8930746560008747
Mean Macro Recall: 0.8945670466879897
Mean Macro F1 Score: 0.8937634002916877
```



The best learning rates and weight decay parameter pair for the Transfer Learning model are learning rate: 0.0001 and weight decay: 0.005, with validation set loss of 0.3094. Once again, even for transfer learning, most models are overfit. This highlights how using low number of training samples is problematic.

Transfer learning is a lot faster to train. The algorithm's run time is linearly related to the max_epoch value as previously discussed, and this model only needs 20 epochs compared to our previous 300. From

this, we would assume the model would take 3 to 4 minutes. However, it takes 8 minutes to train the model. This is probably because the pre-trained model is also a lot deeper than our CNN, so the forward and backpropagation steps take longer in each epoch.

If we have the option, it is better to use a pertained model because these models are trained with a lot of high-quality data, and using them in transfer learning allows us to utilize the learning from that training. This is very apparent from the results, as the results of the transfer learning model are significantly better than our two previous approaches.