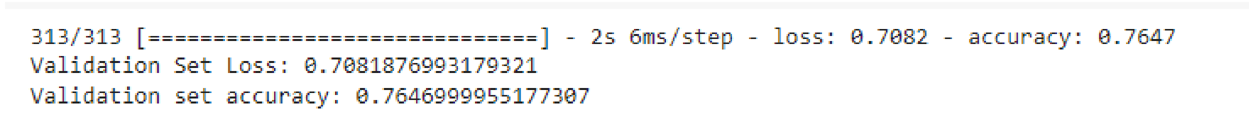
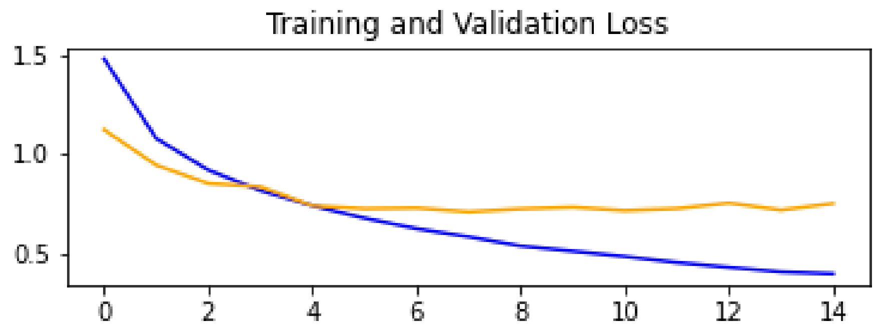
Colin Nordquist
Machine Learning Final

- Basic CNN Model
    - 2 layers, 1 pooling layer, 2 layers, 1 pooling layer, 1 flattening layer, 1 dropout layer (to prevent overfitting), 2 dense layers



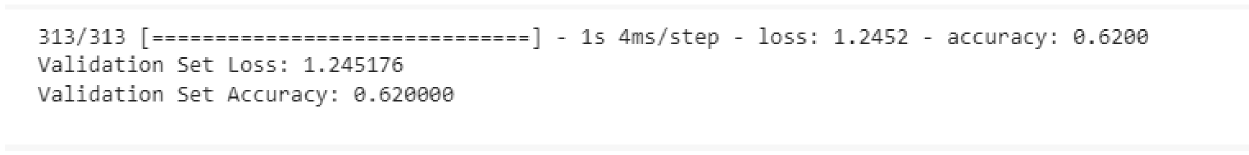
```
313/313 [==============================] - 2s 6ms/step - loss: 0.7082 - accuracy: 0.7647
Validation Set Loss: 0.7081876993179321
Validation set accuracy: 0.7646999955177307
```

    -



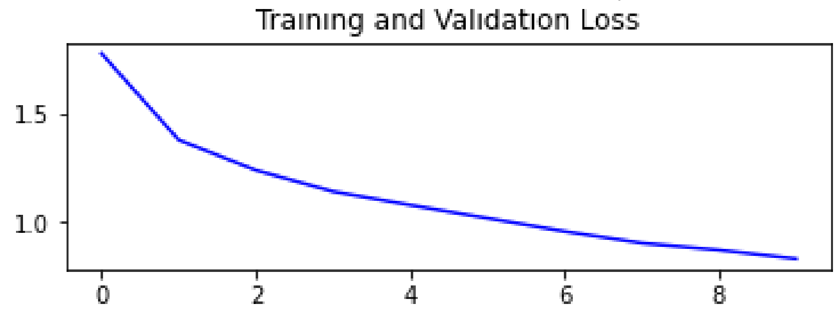
    -
    - Uses the Keras library
    - Building and fine tuning this model resulted in me achieving the highest accuracy on assessing the validation set of the three models I was able to implement. I started with a model based upon the one presented in the book, but specified to this dataset. After trying a few alternatives of patterning convolution layers and pooling layers, I ended up going with 2 convolution layers, followed by a pooling layer, then repeated. For the convolution layers, I found that the ReLu activation function tended to work best. When trying other activation functions they performed far worse, for example when using softmax on all convolution layers the accuracy of the model dropped to under 10%. This makes sense as to why ReLu is most commonly used among different well known CNN architectures. For the pooling layers, max pooling was used as it always tends to perform better than average pooling as only the most important features are taken into account.
    - After flattening, I also made sure to use a dropout layer in order to prevent overfitting. This regularization is important because without it the model may perform very well on the training data, but not work as well on the test data when we use the model in the end. I also noticed a slight drop in accuracy of 1-2% when omitting this layer. I ended up using the Adam optimizer instead of stochastic gradient descent (SGD) because it is a superior optimizer, and my results also showed that it performed better. When using SGD I only was able to achieve an accuracy of 69%. I also settled on 15 epochs because it was what I could manage given the ram and gpu usage needed to to train the model in a reasonable amount of time. Adding more epochs had extremely diminishing marginal returns as well in accuracy improvement.

    - After tweaking and trying variations of this model, the best accuracy I was able to achieve on the validation set was 76%, which given this is a basic CNN, I believe to be a pretty good level of accuracy. I believe I tried to modify all the parameters and aspects of the CNN I could to bring this model to this level of accuracy.

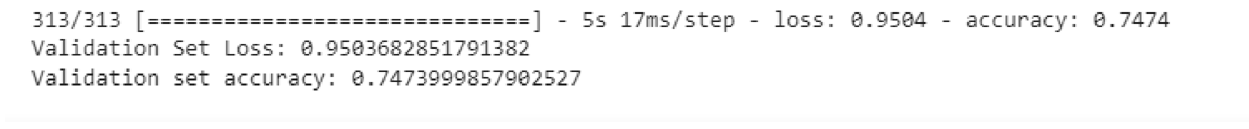
- LeNet-5 Architecture
    - 

```
313/313 [==============================] - 1s 4ms/step - loss: 1.2452 - accuracy: 0.6200
Validation Set Loss: 1.245176
Validation Set Accuracy: 0.620000
```

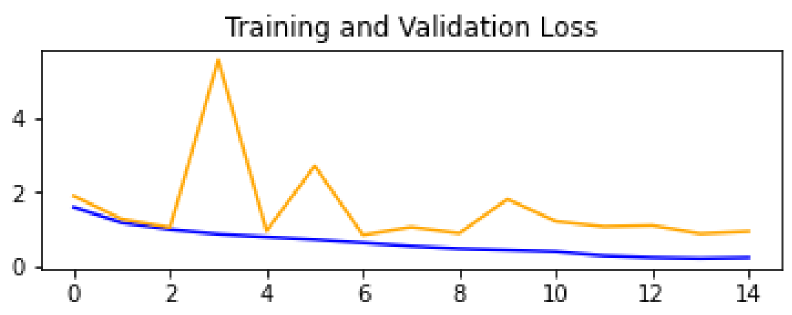
    - (had trouble plotting validation loss, however since the accuracy was much lower than the first model, I did not think it was too important)



    - Uses the Keras library
    - I wanted to try other architectures to see how they compare, this was the first I wanted to try. As one of the original CNN architectures, I wanted to see how LeNet-5 would perform.This model performed fairly poorly compared to my existing model. I also had issues with calculating the validation loss over the epochs run, however with an accuracy of over ten percent less than my existing model, I did not think it was worth my time to fix it.

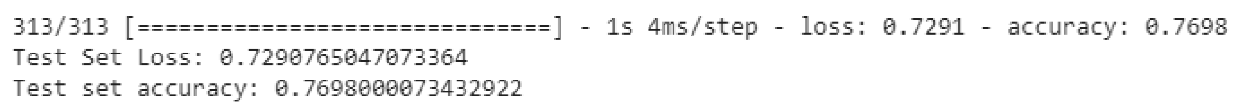
- ResNet-34 Architecture
    - 

```
313/313 [==============================] - 5s 17ms/step - loss: 0.9504 - accuracy: 0.7474
Validation Set Loss: 0.9503682851791382
Validation set accuracy: 0.7473999857902527
```



  -
  - Uses the Keras library
  - I wanted to try the ResNet-34 architecture as it is an extremely deep CNN composed of 34 layers. I thought that with a deeper model I could achieve a higher accuracy. To my surprise, the model performed slightly worse than the CNN I built, if only slightly. This model also appeared to perform slightly more inconsistently, sometimes I could achieve close to 75% percent accuracy, other times it would as low as 70%. If I had more time and more computing power I would also try ResNet-152, which is similar but goes even deeper than ResNet-34 (152 layers vs 34 layers).

- For the final evaluation on the test set I decided to use the model I built, as it resulted in the highest accuracy on the validation set. I believe I took proper regularization precautions as well to prevent overfitting so that it will perform similarly on the test set. As we can see, the model performs similarly well on the test set as it did on the validation set, even slightly better. I am overall pleased with the performance of the classifier, without using much more advanced techniques I think that 77% accuracy is a very good level to have achieved.



```
313/313 [==============================] - 1s 4ms/step - loss: 0.7291 - accuracy: 0.7698
Test Set Loss: 0.7290765047073364
Test set accuracy: 0.7698000073432922
```