Bio Computing Report

https://colab.research.google.com/drive/1XrH4_sZkVaQbOqU_onxkU6C8MO_mVYSK?usp=sharing

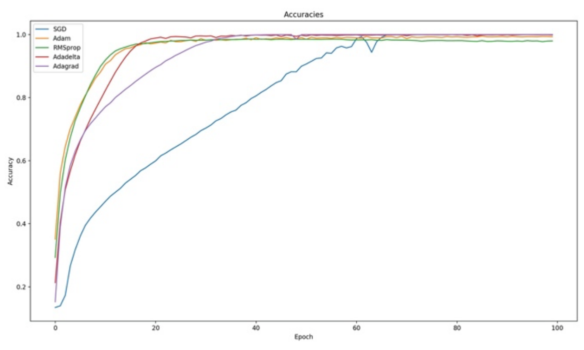
This project involved me taking a very basic convolutional neural network and building it into a more sophisticated and accurate model. The goal was to achieve a much higher degree of accuracy on the CIFAR-10 dataset than the original 10%. During this project, I conducted a wide variety of tests to discover what would help the model make good predictions. These tests included comparing various epochs, batch sizes, dense layers as well as many other factors. Along with individual tests, I conducted a large amount of research to guide my analytical progress. The research really helped my understanding of convolutional neural networks and had a huge impact on the improvement of the model. My model currently averages 86% accuracy, which is a huge improvement to the model I was given. I am very happy with this result.

Going into this project I was already aware of a few things I would need to test. The three main areas where the training parameters (epochs and batch size), image augmentation and the convolutional neural network layers themselves. Once I had familiarised myself with the code, I setup TensorBoard. TensorBoard is an interface that allows you to store model results and compare them using an automatically plotted graph. This is very important as it would save me time writing the results into a spreadsheet and enabled me to compare the accuracy with a readable graph. TensorBoard was also very useful at helping me identify over and underfitting. My main strategy for testing involved writing a script that would change one aspect or value using a loop which I would then compare using TensorBoard. For example, I would loop through a range of batch sizes and see which returned the best result. It was important that I only tested one variable at a time, otherwise there was a risk of clashing, hence swaying the comparison.

At the outset, I knew that I would not be able to test everything due to time and processing restraints. Though, I was not going to spend my time testing prewritten architectures such as GoogLeNet. If I wanted to tune my model to improve the accuracy for this specific dataset, I need to test as many aspects of the model as possible. This is why I decided to spend my time evaluating the hand-crafted model instead.

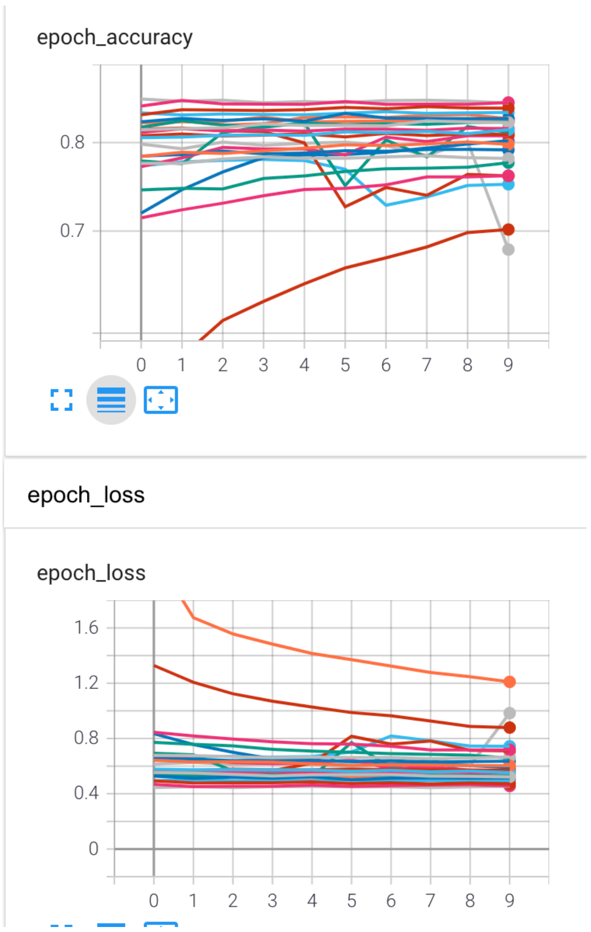
Optimizer

The first experiment I conducted was focused on the optimizer. The optimization algorithm is incredibly important as it guides the model to its most accurate form by adjusting the weights. After a little research, I found this image with the best optimizers for multiclass classification.

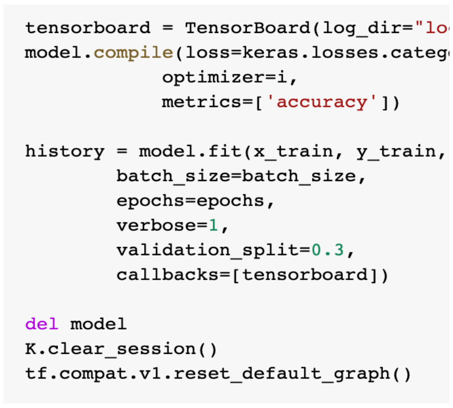


(Introduction to Optimizers, 2018)

Although this image was helpful, I needed to conduct the comparison on my model to find the best optimizer for this dataset. I programmed a loop that would replace the optimizer with each new iteration so I could view the individual results on TensorBoard.

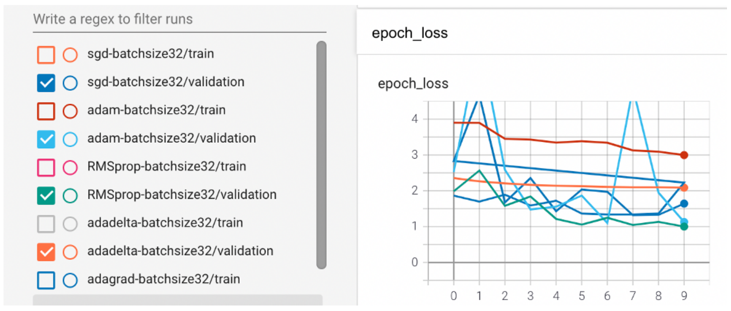


It was immediately obvious that over-fitting had occurred as a lot of the models were instantly achieving very high accuracy without any learning curve. This occurs when the model begins memorising the results. To combat this, I added some code that would clear the session and delete the model after each iteration.



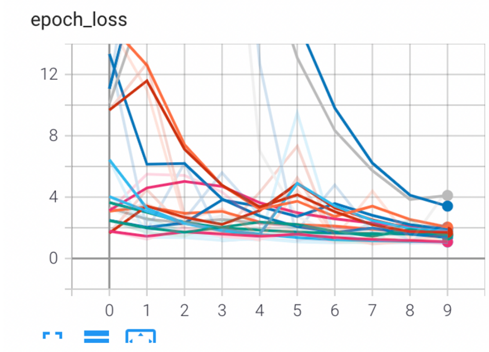
```
tensorboard = TensorBoard(log_dir="lo
model.compile(loss=keras.losses.categ
            optimizer=i,
            metrics=['accuracy'])

history = model.fit(x_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        verbose=1,
        validation_split=0.3,
        callbacks=[tensorboard])

del model
K.clear_session()
tf.compat.v1.reset_default_graph()
```

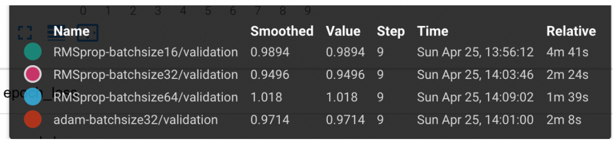
My results can be seen in the graph below


Write a regex to filter runs

- [ ] ○ sgd-batchsize32/train
- [✓] ○ sgd-batchsize32/validation
- [ ] ○ adam-batchsize32/train
- [✓] ○ adam-batchsize32/validation
- [ ] ○ RMSprop-batchsize32/train
- [✓] ○ RMSprop-batchsize32/validation
- [ ] ○ adadelta-batchsize32/train
- [✓] ○ adadelta-batchsize32/validation
- [ ] ○ adagrad-batchsize32/train

From my research, I had discovered that it was often useful to compare optimizers using the loss value. Although RMSprop seemed to be the winner, adam was only short by a few tenths. I wanted to double-check this, so I conducted another experiment on the batch size. I took the leading optimizers (RMSprop, adam and adadelta), and tested them with a few different batch sizes.



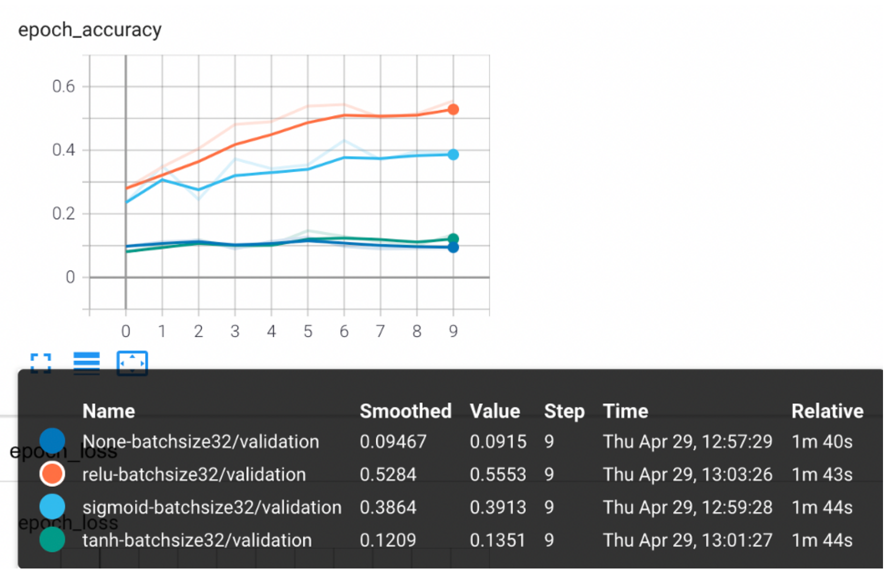


	Name	Smoothed	Value	Step	Time	Relative
●	RMSprop-batchsize16/validation	0.9894	0.9894	9	Sun Apr 25, 13:56:12	4m 41s
●	RMSprop-batchsize32/validation	0.9496	0.9496	9	Sun Apr 25, 14:03:46	2m 24s
●	RMSprop-batchsize64/validation	1.018	1.018	9	Sun Apr 25, 14:09:02	1m 39s
●	adam-batchsize32/validation	0.9714	0.9714	9	Sun Apr 25, 14:01:00	2m 8s

I was happy with these results as they solidified RMSprop as the best optimizer for this dataset. The accuracy was around 50% which was already a huge increase on the original model. Although 3 out of those 4 results are implementing the RMSprop optimizer, there is still a significant difference in runtime. As the only change between those 3 models is the batch size, I assume this is causing the varying runtime. I will conduct an experiment on batch size at a later stage. RMSprop will be my optimizer going forward. To make this change, I simply edited one line of code. I learnt that it is very important to check for under and over-fitting as this will completely sway the results.

Activation Functions

My next experiment involved testing various activation functions. An activation function helps the “network learn complex patterns in the data” (Jain, 2019). I decided to test no activation function, ReLu, sigmoid and tanh as from my research these were used frequently online. The aim of this experiment is to see if a specific activation function will result in a more accurate model.



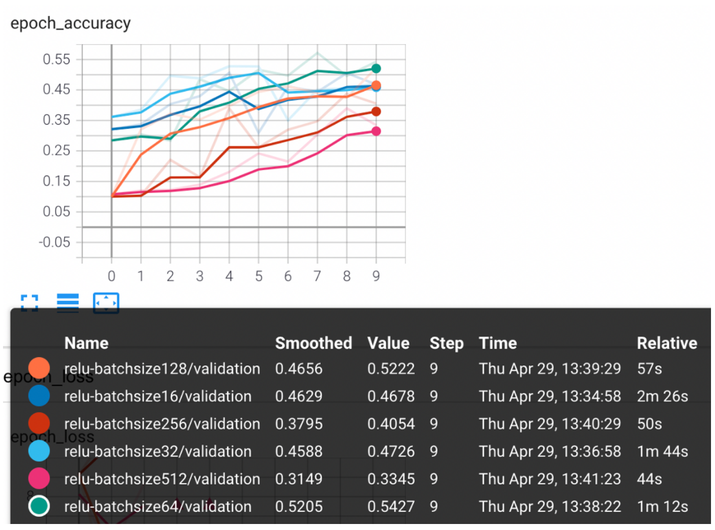
	Name	Smoothed	Value	Step	Time	Relative
●	None-batchsize32/validation	0.09467	0.0915	9	Thu Apr 29, 12:57:29	1m 40s
●	relu-batchsize32/validation	0.5284	0.5553	9	Thu Apr 29, 13:03:26	1m 43s
●	sigmoid-batchsize32/validation	0.3864	0.3913	9	Thu Apr 29, 12:59:28	1m 44s
●	tanh-batchsize32/validation	0.1209	0.1351	9	Thu Apr 29, 13:01:27	1m 44s

These results make it very clear that ReLu is the best activation function for this dataset by quite a lot with an accuracy of 15% more than the second-best activation function. I noticed that the activation function seems to have had almost no effect on the model's runtime so there is nothing to analyse in that area for this experiment. ReLu meaning Rectified Linear Unit is a function that will output the positive input or otherwise output zero (Brownlee, 2020). Although it sounds complicated to program, the implementation was very simple as it only requires the alteration of a few lines of code using predefine Keras library functions. I was already using ReLu, though I am happy I validated its use in my model. I learnt that even though something is performing well, it is still important to compare it against other methods in case another is superior.

Batch Size

Batch size is arguably one of the most important parameters to "tune in modern deep learning systems" (Shen, 2018). A large batch size allows the GPU to process more data at a time but can often lead to "poor generalization" (Shen, 2018). My aim for this test is to find the perfect median. To accomplish this, I trained the same model with 6 different batch sizes. I conducted this in the same way as the previous tests by training multiple models with the various values and comparing the results using TensorBoard.

```
for batch in [16, 32, 64, 128, 256, 512]:
```



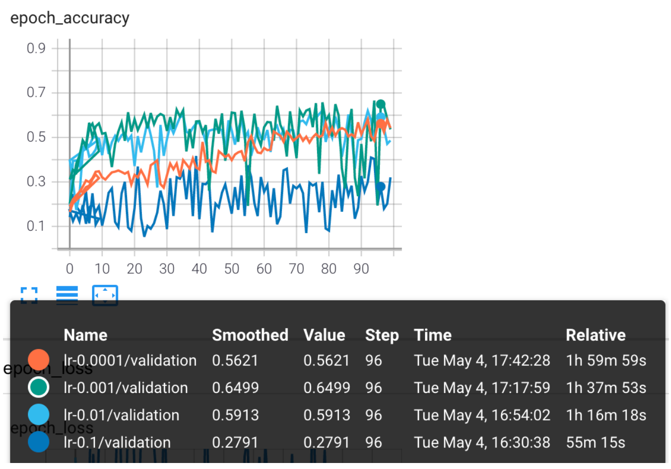
	Name	Smoothed	Value	Step	Time	Relative
●	relu-batchsize128/validation	0.4656	0.5222	9	Thu Apr 29, 13:39:29	57s
●	relu-batchsize16/validation	0.4629	0.4678	9	Thu Apr 29, 13:34:58	2m 26s
●	relu-batchsize256/validation	0.3795	0.4054	9	Thu Apr 29, 13:40:29	50s
●	relu-batchsize32/validation	0.4588	0.4726	9	Thu Apr 29, 13:36:58	1m 44s
●	relu-batchsize512/validation	0.3149	0.3345	9	Thu Apr 29, 13:41:23	44s
○	relu-batchsize64/validation	0.5205	0.5427	9	Thu Apr 29, 13:38:22	1m 12s

The plotted models clearly put the 64 batch size model at the top with 54% accuracy. This was great news as I had now improved my model by around 5%. The runtime has also been reduced from 1 minute 44 seconds to only 1 minute and 12 seconds. This supports my hypothesis from the first experiment that the batch size was impacting the runtime. Although small, this is still a benefit to the model. Similarly to the first experiment, this only required a one-word fix which is nice and easy. I learnt here that the two middle values performed the best. Although this might appear to be obvious, this is often not the case which is why I am glad I checked.

Learning Rate

Learning rate is very important for a few reasons. If the learning rate is too small, then the training time can be very long and will sometimes cause the model to get stuck. On the other hand, if the learning rate is too high, the "weights associated with the synapse connection will change too frequently" (Brownlee, 2020) and the training process will be unstable. A small learning rate is often used with a large number of epochs and vice-versa. The aim of this experiment was to identify a learning rate that would not cause the model to get stuck but would also provide a stable learning process.

```
for lr in [0.1, 0.01, 0.001, 0.0001]:
  train_model(lr)
```



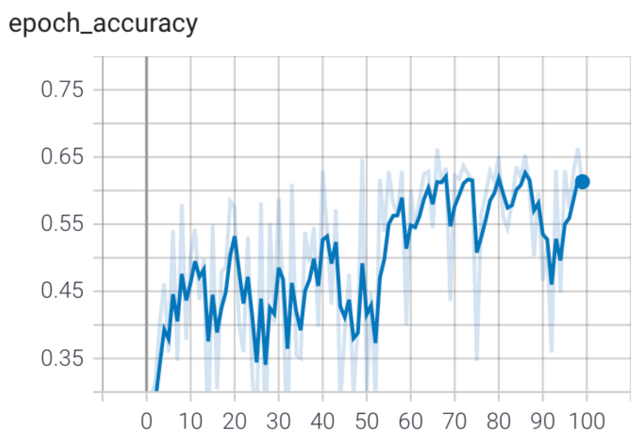
	Name	Smoothed	Value	Step	Time	Relative
●	lr-0.0001/validation	0.5621	0.5621	96	Tue May 4, 17:42:28	1h 59m 59s
○	lr-0.001/validation	0.6499	0.6499	96	Tue May 4, 17:17:59	1h 37m 53s
●	lr-0.01/validation	0.5913	0.5913	96	Tue May 4, 16:54:02	1h 16m 18s
●	lr-0.1/validation	0.2791	0.2791	96	Tue May 4, 16:30:38	55m 15s

As we can see from the graph, a learning rate of 0.001 produces the best accuracy when left for 100 epochs. The downside is that it has a very slow runtime at 1 hour and 38 minutes. I am willing to accept this long runtime as it produces significantly better results than the rest. It is also important to note that the huge jump in runtime is also due to the fact that I have trained the model for 100 epochs in this test. The reason I increased the epochs was to allow the model some more training time. I was concerned that the low number of epochs might not be giving the model enough time to learn for a proper evaluation. This learning rate has created my model's highest accuracy yet with 65%. This was a great result, and I am glad I performed this test. I have added the 'lr=0.001' code to my optimizer which was easy to implement. Since finishing the implementation of this project, I have discovered learning rate schedulers. This is when the learning rate will be varied depending on the number of epochs carried out. This would really help solve the issue with instability and would have improved my accuracy. This has taught me that I should spend more time researching other important factors instead of focusing on one specific area at a time.

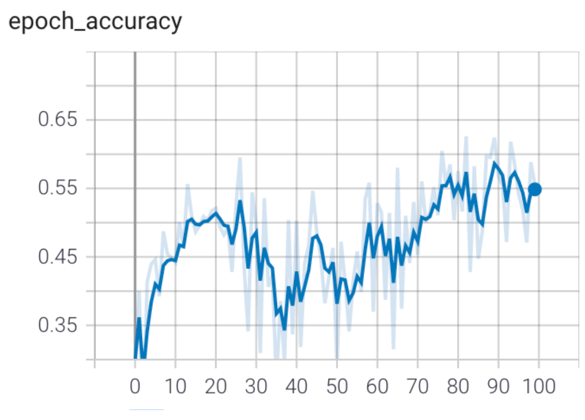
Data Augmentation

Data augmentation is a method of creating a more diverse dataset from the data you already have. This can be accomplished in a variety of ways such as image rotating, cropping or flipping to name a few. If your dataset is not diverse, the accuracy will diminish when given new images. As the data augmentation code is separate from the model's training, I manually tested a few different combinations to see which gave the best results.

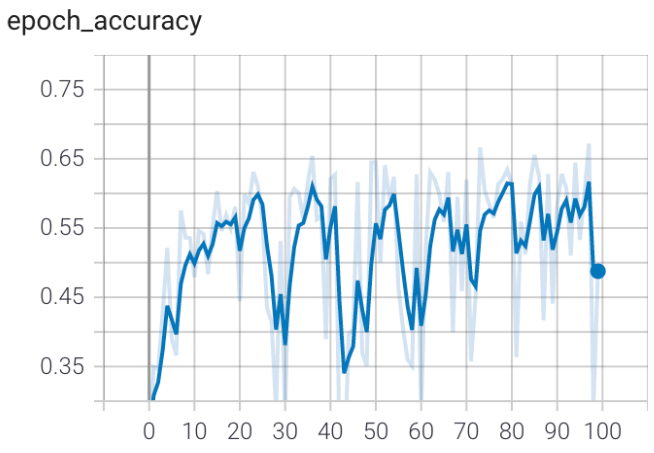
```
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=False)
```



My first change was not very good as I the accuracy decreased. I then decided to change the horizontal flip to 'True'.



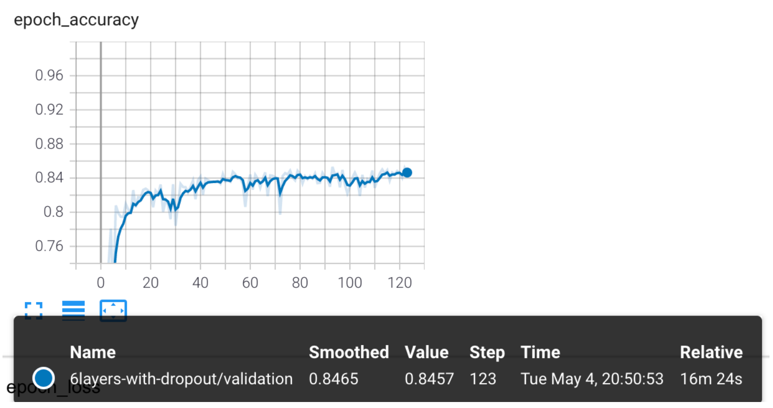
This decreased the accuracy even more. I changed the horizontal flip back to ‘False’ and changed the rotation range to 15.



Although this appears to be similar to the first attempt, the model reached a 67% accuracy. The graph shown has smoothing on which shows the line of best fit. The faded graph line is the most accurate. This is a great result as 67% is my highest accuracy yet. As I tested these values manually, it was more awkward to implement. Looking back, I should have programmed a quick loop to test more combinations of values. If I had done this, I may have had an even better result. At the time, I was happy with the improvement and wanted to move on to a new experiment.

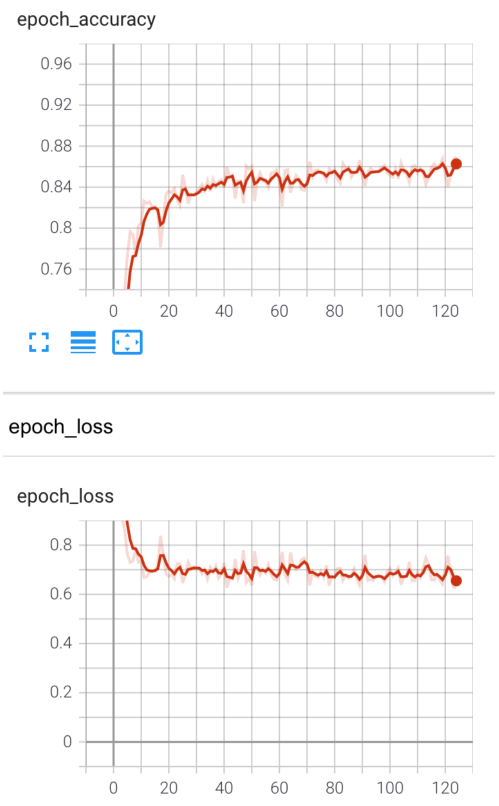
Convolutional Layers

In my previous experiments I had hardly modified the individual layers. I decided that I should double the number of convolutional layers and study the impact of this.



Name	Smoothed	Value	Step	Time	Relative
6layers-with-dropout/validation	0.8465	0.8457	123	Tue May 4, 20:50:53	16m 24s

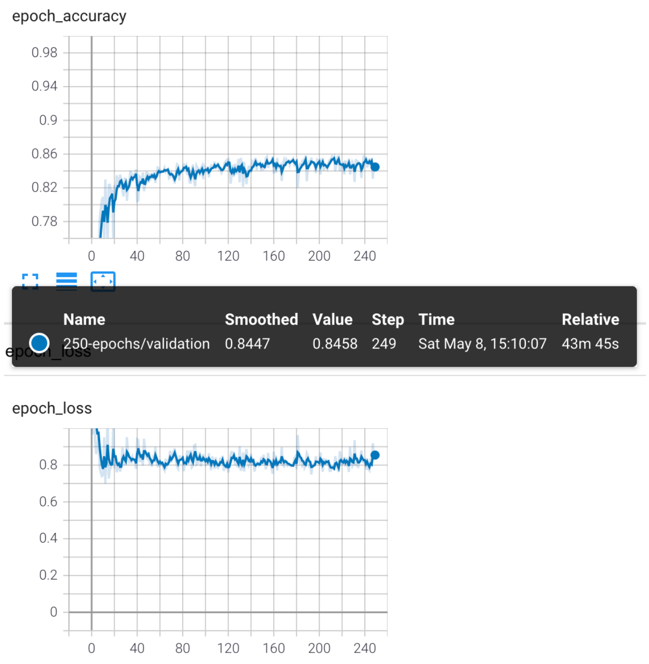
This result was incredible. By doubling the number of layers, I managed to gain an additional 20% improvement in accuracy. Thinking I had found the best result first time, I didn't test another variation of layer count. This was a mistake and something I would change in the future. If I had continued trying a different number of convolutional layers, I may have found an even better model. Nevertheless, I still made a few changes. From my previous model, I had a few dropout layers to prevent overfitting. The dropout layer accomplishes this by randomly ignoring layer outputs to give the model a slightly different view of the data each time. Realising I had forgotten to continue this, I added in the dropout layers. I set all the dropout layers to 0.2 and trained the model again.



I am extremely happy with this result. My model has reached 86% accuracy with the validation set. I didn't expect adding the dropout layers would have such a big impact on the accuracy. Adding the dropout layers was very easy as it was just a matter of copy and pasting at the end of each convolutional layer. An accuracy of 86% is much higher than I ever expected to reach with this model in the time we were given, so I will be stopping after one last experiment.

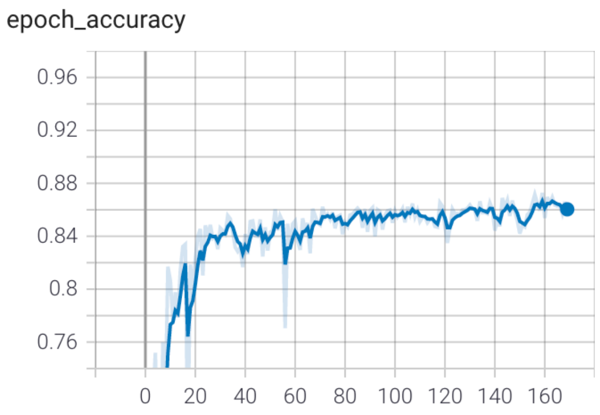
Epochs

My final test involved the number of epochs. The number of epochs “defines the number of times that the learning algorithm will work through the entire training dataset” (Brownlee, 2018). An epoch value too low will result in underfitting, whereas an epoch value too high will result in overfitting and a long runtime. In order to find the perfect in-between, I will be running the model up to 250 epochs and analysing the results. For this experiment no loops were needed as I would not need to change any values during runtime.



Name	Smoothed	Value	Step	Time	Relative
● 250-epochs/validation	0.8447	0.8458	249	Sat May 8, 15:10:07	43m 45s

The first thing I noticed is that after around 170 epochs, the accuracy did not seem to improve. It was also obvious that a runtime of nearly 44 minutes in total is relatively very long. To stop the model overfitting or wasting time, I have decided that an epoch value of 170 is best. It might be useful in future to run this experiment a handful of times as the results can vary. Overall, I am very happy with this model and proud to show my final results below.



```
              precision    recall  f1-score   support

           0       0.84      0.87      0.86      1000
           1       0.96      0.90      0.92      1000
           2       0.78      0.84      0.81      1000
           3       0.78      0.71      0.74      1000
           4       0.87      0.83      0.85      1000
           5       0.82      0.76      0.79      1000
           6       0.83      0.94      0.88      1000
           7       0.94      0.86      0.90      1000
           8       0.85      0.94      0.89      1000
           9       0.90      0.90      0.90      1000

    accuracy                           0.86     10000
   macro avg       0.86      0.86      0.86     10000
weighted avg       0.86      0.86      0.86     10000
```

Conclusion

Throughout this project, I have conducted a wide range of experiments in order to improve the accuracy of my model. By comparing the results of these tests with clearly plotted graphs, I focused in on the optimum training process.

The requirements for this project allowed me to make my own decision on what methods I wanted to evaluate. That being said, optimizers, data augmentation, batch size and epochs were all suggested as potential factors to test. I decided to evaluate all of these and a couple more aspects I felt I could within the time of the project. I also evaluated a combination of strategies when I tested the optimizers with a few different batch sizes. This was also suggested in the project scope. The project highlights the importance of a tuning process backed by "well thought out experiments" instead of trial and error. I believe my tests accomplished this with the research conducted before each one. Being able to test multiple values for a certain parameter at a time removed any luck element and replaced it with clear graphs for logical comparison.

In the future, there are a couple of changes I would make. There were a few times when I needed to sacrifice one aspect for another. For example, a learning rate that is too big can cause instability whereas a small learning rate can cause the model to get stuck. Because of this, a small learning rate is usually paired with a high number of epochs and vice-versa. In my code, I had to sacrifice a little from each end to produce an optimum learning rate. I instead could have researched the issue further and discovered a learning rate scheduler can make these adjustments as the model trains. This would have allowed me to get the best of both sides. To prevent this in future, I will research any problems I have further before making a decision. Any issues I have will likely have been discovered and fixed by others with the solution published online or in literature. I will be sure to utilise this more in the future. Another suggestion for the future would be to evaluate multiple strategies at a time more often. A certain optimizer might work best with a specific learning rate, but a different optimizer might outperform that with another learning rate. The more combinations you can test, the better the tuning process will be. In this project, I was limited with time and processing power, but in future, this would be something I would like to dedicate more time to.

Overall, I am very happy with the final model and the steps I took to get there and what I learned along the way. From the beginning, I found a great way to compare models, and I effectively used loops to evaluate various parameter values. Another aspect I am happy with was my note-taking throughout. I would write down any ideas I had before I tried them, and I made sure to keep screenshots of all the results. This made the report a lot easier. An area I could definitely improve on is my research. Although I did spend a lot of time researching methods to improve the accuracy, I should have spent more time researching factors that were not already included in the project scope. Nearly all of my tests involved changing a value that was already in the model. If I had spent more time looking for additional factors, I may have come across the learning rate scheduler along with others.

Bibliography

Algorithmia Blog. 2021. *Introduction to Optimizers*. [Online]. Available at: <https://algorithmia.com/blog/introduction-to-optimizers#what-is-an-optimizer-in-machine-learning> [Accessed 9 May 2021].

Brownlee, J. 2020. 'A Gentle Introduction to the Rectified Linear Unit (ReLU*)'. Machine Learning Mastery* [Online]. Available at: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural networks/#:~:text=The%20rectified%20linear%20activation%20function,otherwise%2C%20it%20will%20output%20zero.&text=The%20rectified%20linear%20activation%20function%20overcomes%20the%20vanishing%20gradient%20problem,learn%20faster%20and%20perform%20better [Accessed 12 May 2021].

Brownlee, J. 2018. 'Difference Between a Batch and an Epoch in a Neural Network', *Machine Learning Mastery* [Online]. Available at: <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/#:~:text=The%20number%20of%20epochs%20is%20a%20hyperparameter%20that%20defines%20the,of%20one%20or%20more%20batches.> [Accessed 12 May 2021].

Brownlee, J. 2020. 'Understand the Impact of Learning Rate on Neural Network Performance', *Machine Learning Mastery* [Online]. Available at: <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/> [Accessed 12 May 2021].

Jain, V. 2019. 'Everything you need to know about "Activation Functions" in Deep learning models', *Towards Data Science* [Online]. Available at: <https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253> [Accessed 12 May 2021].

Shen, K. 2018. 'Effect of batch size on training dynamics', *Medium* [Online]. Available at: <https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e#:~:text=Batch%20size%20is%20one%20of,from%20the%20parallelism%20of%20GPUs> [Accessed 12 May 2021].