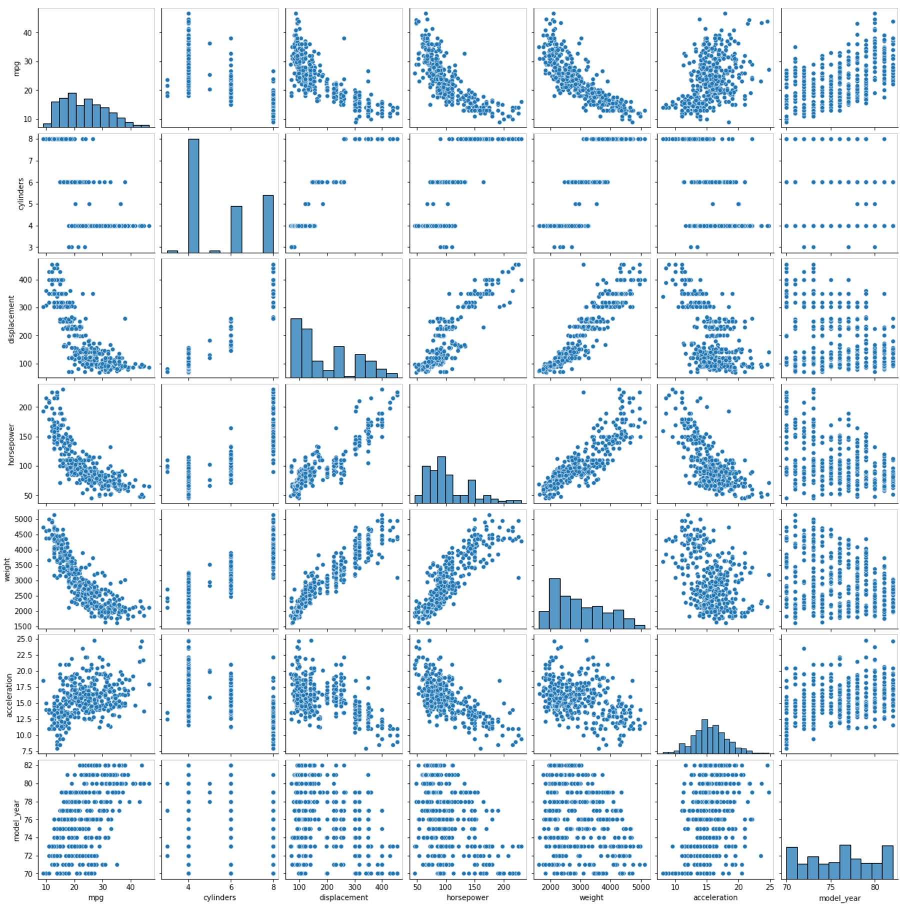
Amelia Bentley

Data 302 Assignment 3

Part 1 Linear Regression [20 marks]

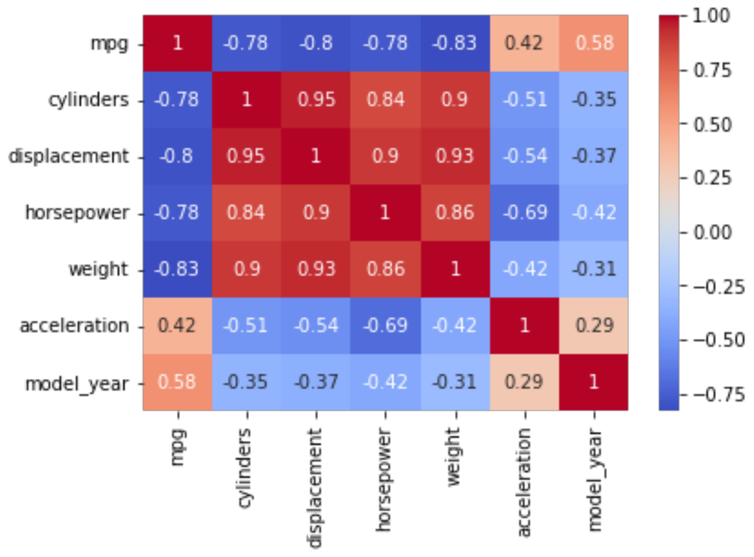
The first step in conducting exploratory data analysis is to look at the data (a number of codes can be used to do this and you will see the different ways in my source code). This allows us to look at the data we are investigating. We can look at the column names and data types in each column and decide what visualisation steps we should take. We can see that this data set shows information about different car models. Our target variable is MPG, which shows Miles per gallon, indicating the fuel efficiency of the car. We have eight predictor variables these are printed out in the source code.

Pairwise plot:



Pairwise plots allow for visualizing relationships between pairs of variables in the dataset. In the diagonal plots, we can see histograms of each variable, These plots show the distribution of each variable individually, helping you understand its shape, central tendency, and spread. A clear skew to the right can be observed for displacement, horsepower, weight, and mpg variables. Skewness coefficients calculated for each variable confirm the right-skewed nature of their distributions, with values exceeding 1.0 for displacement and weight, indicating substantial skewness. This skewness suggests that these variables are concentrated towards the lower end of their respective scales, with a longer tail extending towards higher values. In the context of our analysis, the right-skewness of the MPG variable implies that there are more observations with lower fuel efficiency compared to higher fuel efficiency. Understanding the skewness of these variables is crucial for selecting appropriate analysis techniques and interpreting the results accurately. In the off-diagonal plots, there seems to be a clear linear pattern amongst numerical data (acceleration, weight, horsepower, displacement). Some variables show strong negative correlations to one another while others show positive correlations. I investigate these relationships further in the heatmap below.

Heat map to show correlation:



Heat maps show the Correlation coefficients between variables, these quantify the strength and direction of the linear relationship between two variables. Coefficients range from -1 to 1, Values close to 1 indicate a strong positive correlation. Values close to -1 indicate a strong negative correlation. Values close to 0 indicate little to no linear correlation. The heat map shows quite a lot of variables share strong positive correlations these are as follows: Model Year and MPG: 0.58, Displacement and Weight: 0.94, Cylinders and Displacement: 0.95, Weight and cylinders: 0.9, Displacement and horsepower, Weight and horsepower. These strong positive relationships explain the effects that both variables have on each other, for instance since the variables displacement and weight have a strong positive relationship we can interpret this as the engine displacement increases the weight of the car also increases (larger car engine results in a heavier car). Similar conclusions can be made for all these variables. The heat map also shows Strong Negative correlations between the following variables: MPG & Weight: -0.83, MPG and Displacement: -0.80, MPG and Horsepower: -0.78, MPG and cylinders: - 0.78, Acceleration and horsepower – 0.69 The strong negative relationship between MPG and weight tells us that as the weight of the car increases the fuel efficiency(miles per gallon) decreases. Once again a similar conclusion of a strong negative correlation can be made for all these variables. Understanding the correlations between variables helps us identify key relationships in our data, this allows us to see which variables are likely to be key predictors in our model.

Dealing with missing values: I checked missing values and found there are 6 missing values in the horsepower data. Given that horsepower is a numeric value I used the mean imputation method we learnt in lectures. This approach uses the mean value of the data to deal with missing values and is most appropriate for this data type.

Encoding categorical values: The categorical variables in the dataset are origin and name. For origin, I decide to use one-hot encoding. One-hot encoding is a technique used to convert categorical variables into a numerical value so it can be used in linear regression models. It does this by creating binary (0 or 1) dummy variables for each category in the variable, so for origin it would do this for each category, USA, Europe and Japan. This was an appropriate method for encoding as the ‘origin’ data is a nominal variable and so there is no order/hierarchy to the data therefore we don’t need an encoding method that does this. The name data had lots of categorical values so encoding the data with one hot encoding would be computationally intensive, because of this I used label encoding. Label encoding gives a numeric value to the categorical groups example: mustang becomes 1, Chevrolet becomes 2 etc, this method is a simple method for encoding categorical variables with many values.

Results of all three models:

Training performance evaluation using MSE & R2 for the original model (coefficients in the code)

Mean squared error for training:	11.29
Coefficient of determination(r2) for training:	0.82
Mean squared error for test:	8.35
Coefficient of determination(r2) for test:	0.84

Training performance evaluation using MSE & R2 for Ridge (coefficients in the Python code output)

Mean squared error for training:	11.29
Coefficient of determination(r2) for training:	0.82
Mean squared error for test:	8.34
Coefficient of determination(r2) for test:	0.84

Training performance evaluation using MSE & R2 for lasso model (coefficients in the Python output)

Mean squared error for training:	12.09
Coefficient of determination(r2) for training:	0.81
Mean squared error for test:	9.08
Coefficient of determination(r2) for test:	0.83

Comparison of models	Original model	ridge	Lasso
Mean squared error for training:	11.29	11.29	12.09
Coefficient of determination(r2) for training:	0.82	0.82	0.81
Mean squared error for test:	8.35	8.34	9.08
Coefficient of determination(r2) for test:	0.84	0.84	0.83

When comparing the original model to the two enhanced models we can see that the performance metrics don't differ much between the initial model and the ridge regression model. For the two models, all metrics are the same except for the ridge MSE which is 8.34 as opposed to 8.35. This tells us that the ridge model did not significantly impact the model's performance compared to the original model. The lasso regression model shows a higher MSE for both the test and training set. A higher MSE suggests larger errors in this model compared to the others. This typically happens in lasso regression as it often penalises some coefficients too harshly which can lead to underfitting or oversimplifying. The lasso model also has a lower r-squared value than both models, this suggests that the lasso model is less effective in capturing the variables in the target variable and either under or over-fits. The poorer performance metrics for the lasso model suggest that the feature selection of the lasso model has impacted the model's predictive ability. The feature selection of the lasso model can be seen when we look at the coefficients, the lasso model was able to shrink some coefficients to zero, this creates a simpler model with fewer features. The coefficients of the other models do differ slightly but neither of the other models shrunk coefficients to zero.

To conclude, the ridge model shows better metrics and the lasso model shows the ability to reduce some of the predictors to zero. Deciding what model is better depends on the goals of the analysis, if we wish to create a simpler model with fewer predictors then the lasso model is the best option, however, if we want to retain higher metrics then the ridge model would be preferable.

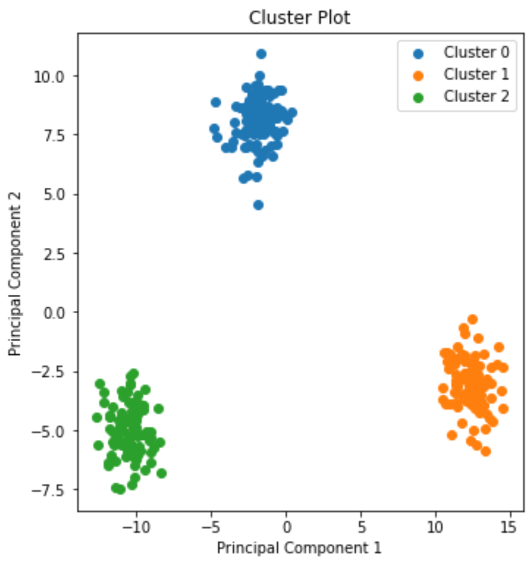
Part 2 Clustering [25 marks]

Determine the *best K* by evaluating silhouette scores for *K* values ranging from 2 to 5:

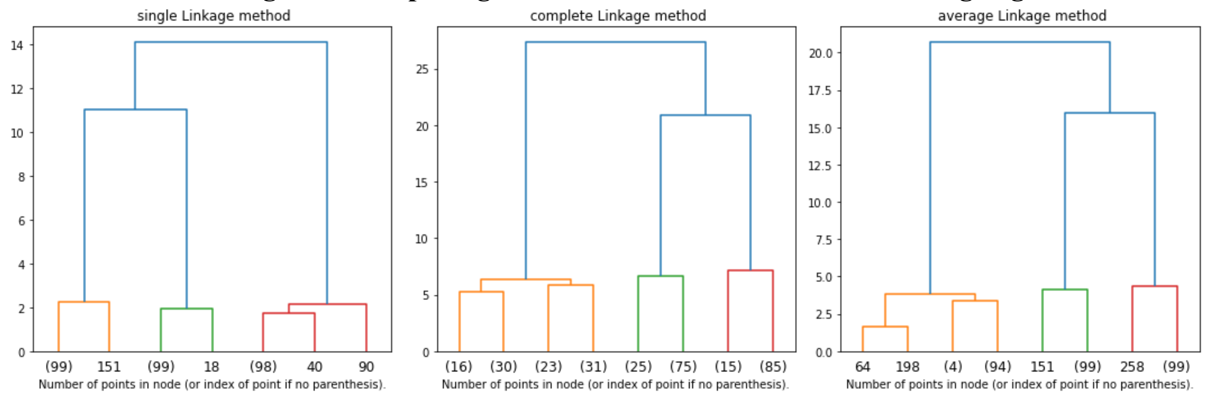
Silhouette score for K=2:	0.6515279371492653
Silhouette score for K=3:	0.838307214726647
Silhouette score for K=4:	0.6335381476038574
Silhouette score for K=5:	0.4220683762143425

A silhouette score measures how similar an object in a cluster is compared to other clusters, (measures cohesion and separation). The scores range from -1 to 1, a value of 1 indicates perfect clustering and -1 shows bad clustering groups. The highest silhouette score from different k values is 3 with a value of (0.842dp). This high score indicates that creating three clusters results in the most cohesive and well-separated clusters, creating the best representation of the underlying data structure.

Scatter Plot Using Two Principal Components:



Dendrograms Comparing The Different Hierarchical Clustering Algorithms



Compare the effect of these linkage methods on creating clusters
Linkage methods are used in Agglomerative clustering to determine how clusters merge during the clustering process, in this assignment we were asked to use single, complete and average linkage methods. Single linkage is a method that computes the minimum pairwise dissimilarity between clusters and then merges them based on this. Complete linkage is a method that computes the maximum pairwise dissimilarity between clusters. It does the opposite of single linkage and cluster data based on the maximum distance. Average linkage is a method that finds the average pairwise dissimilarity between clusters and merges the data based on this.

For the single method graph, it looks like the data is more sparse and elongated compared to the other methods, this can be seen as it creates the merges at low heights. For the complete method, the graph looks more balanced than the single graph showing merging at higher values with more clear clusters. The graph for average looks very similar to the complete graph, it shows balanced and clear merging compared to the average linkage method, however, it does seem to merge the data at slightly lower values than the complete method, this could be a result of this method striking a balance between the extremes of the single and complete methods. In general, both the complete and average linkage methods tend to produce better more balanced dendrograms compared to the single linkage method, however, I believe the average linkage method is probably the best one in this case, given its ability to find a balance between the other two model extremes.

Advantages of hierarchical clustering:
An advantage of this method is that it does not require the number of clusters to be specified beforehand, this results in the method being flexible. Alongside this, You don’t need to re-run the model to obtain clustering with different numbers of clusters. The method can also handle various types of data, including categorical data, without requiring any special pre-processing. This method is also a Deterministic algorithm, this means

that it ensures reproducibility and stability in the clustering results, allowing for consistent interpretation and comparison across different runs

Disadvantages of hierarchical clustering:
One disadvantage is that the clustering is obtained by cutting the dendrogram at a certain height, even if the cluster is necessarily nested within the clustering obtained by cutting at a greater height. This means we can miss out on seeing smaller clusters nested within bigger ones higher up in the dendrogram. Another disadvantage is that this approach is Computationally expensive if we are using a large sample due to the computational complexity of creating pair-wise distances between all pairs of data.

Advantages of k-means clustering:
The main advantage is that this method is computationally efficient, because of this it works well with large datasets without any computational burden. It can deal well with high-dimensional data because it converges data to an optimum level. Another advantage is that it's a simple model that is easy to interpret

Disadvantages of k-means clustering:
One disadvantage is that this method requires the number of clusters (K) to be specified beforehand, which can be a drawback if we are unsure what the optimal number of clusters is. Another disadvantage is that k-means doesn't usually handle nonlinear or irregular data well. In contrast to the hierarchical method, k means has to be run multiple times, this can mean that we can end up with different results every time we re-run. This is not favoured if reputability is important.

When is one preferred:
Typically, the choice of method is based on the goal of the model. If we are interested in the structure of the data or we aren't sure how many clusters we need, hierarchical clustering is preferred. If we want an efficient method and we know the number of clusters(or can determine them easily) then k-means Is preferred. For the make blobs data, either model could be used given the structure of the data.

Part 3- Neural Network [65 marks]

Why I choose the input/out layers:
I set the input size to 64 this is because the digits dataset consists of an 8x8 pixel image of numbers 0-9. For the hidden layer, I set the size to 128 neurons because this is what we were told to do in the assignment handout. For the output layer, I set it to 10 this is because the digits data is made up of 0-9 numbers therefore there are 10 possible classes. For this model, I also used the optim.adam optimizer, I used this because the digits dataset has quite a few variables and the optim.adam model tends to be more efficient in its convergence of data.

For the first model, the first Epoch 1 (starting one) had a Loss of 1.995 and an Accuracy of 59.88%. This shows that at the beginning of the training the model had a higher loss with a moderate accuracy (over 50%). We can then see in the final epoch(15) the loss is 1.561 and the accuracy is 89.99% this shows that by the end of training the model achieves a smaller loss and a high accuracy. From this, we can conclude that after training the model is effective in making predictions. This can be seen once again in the test accuracy, while the test accuracy is a little bit lower than the final training accuracy (88.61% vs 89.99%) it's still a high accuracy. The lower test accuracy compared to training indicates that the model might not generalize to the test data as well as it did during training and therefore we may be able to improve on it. This can once again be supported by the predictions that accurately show the predicted and actual labels.

For the first model the output is:

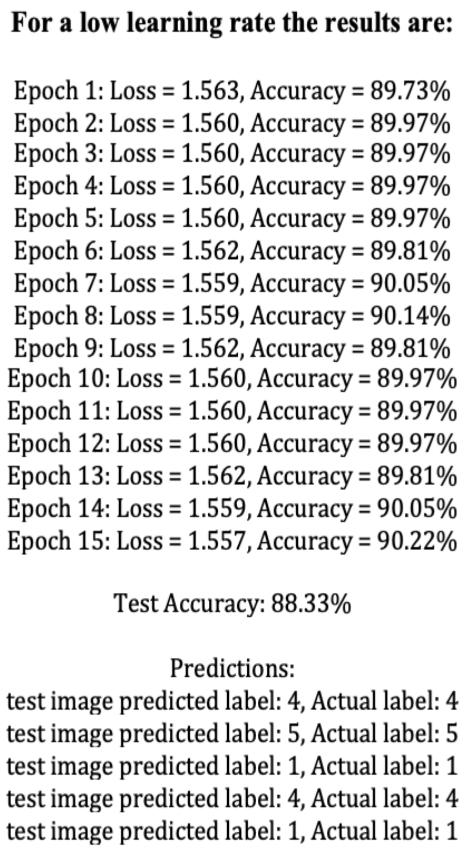
```
Epoch 1: Loss = 1.995, Accuracy = 59.88%
Epoch 2: Loss = 1.638, Accuracy = 84.40%
Epoch 3: Loss = 1.599, Accuracy = 87.20%
Epoch 4: Loss = 1.595, Accuracy = 87.63%
Epoch 5: Loss = 1.588, Accuracy = 87.87%
Epoch 6: Loss = 1.577, Accuracy = 88.98%
Epoch 7: Loss = 1.572, Accuracy = 89.17%
Epoch 8: Loss = 1.567, Accuracy = 89.81%
Epoch 9: Loss = 1.572, Accuracy = 89.10%
Epoch 10: Loss = 1.568, Accuracy = 89.39%
Epoch 11: Loss = 1.567, Accuracy = 89.54%
Epoch 12: Loss = 1.565, Accuracy = 89.69%
Epoch 13: Loss = 1.563, Accuracy = 89.84%
Epoch 14: Loss = 1.565, Accuracy = 89.59%
Epoch 15: Loss = 1.561, Accuracy = 89.99%

Test Accuracy: 88.61%
```

Predictions:

```
test image predicted label: 4, Actual label: 4
test image predicted label: 5, Actual label: 5
test image predicted label: 1, Actual label: 1
test image predicted label: 4, Actual label: 4
test image predicted label: 1, Actual label: 1
```

Comparing different learning rates

A higher learning rate such as 0.01 allows for faster convergence by making larger weight updates, the negative of this is that it increases the likelihood of overshooting and instability. A low learning rate such as 0.0001 gives more precise changes to the weights resulting in slower convergence but better predictors, one negative to this is that It can cause overfitting. For the first model I used a learning rate of 0.01, this is what we saw the optim.adam learning rate set as in the tutorial, this learning rate is commonly used as it strikes a balance between convergence speed and stability. To look at the effects of different learning rates I changed the rate to 0.0001. This resulted in a few differences. Differences in training accuracy, for example, epoch 1 (accuracy 89.73) and the final epoch 15 having an accuracy of 90.22%. This is a higher starting and ending training accuracy. The loss values for the model with the lower learning rate were consistently smaller than those for the original model. This indicates more precise convergence, as smaller loss values suggest that the model’s predictions are closer to the actual target values. Despite these improvements during training, the test accuracy for the model with a learning rate of 0.0001 was 88.33%, this is slightly lower than the 88.61% test accuracy of the model with a learning rate of 0.01(High learning rate). This suggests that while a lower learning rate can lead to more precise weight adjustments and potentially higher training accuracy, it might also result in overfitting. The test predictions show that both models correctly predicted the five test samples, indicating that the overall performance difference doesn’t change the prediction abilities of both models. These results are consistent with the general effects of low and high learning rates. A high learning rate typically shows a rapid decrease in loss and then a rapid increase in the accuracy, this is due to the model making large adjustments to the weights. This means the model can reach the optimal parameters with high precision quicker than the low learning rate. However, if the learning rate is set too high then it can result in the model overpredicting and being less accurate. Given our results the higher learning rate of 0.01 is better for our model, this is because it achieves a better balance of accuracy and loss.

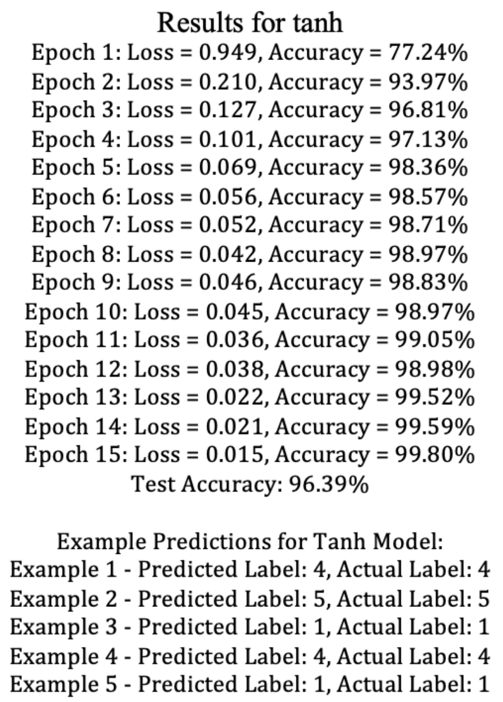


For a low learning rate the results are:

```
Epoch 1: Loss = 1.563, Accuracy = 89.73%
Epoch 2: Loss = 1.560, Accuracy = 89.97%
Epoch 3: Loss = 1.560, Accuracy = 89.97%
Epoch 4: Loss = 1.560, Accuracy = 89.97%
Epoch 5: Loss = 1.560, Accuracy = 89.97%
Epoch 6: Loss = 1.562, Accuracy = 89.81%
Epoch 7: Loss = 1.559, Accuracy = 90.05%
Epoch 8: Loss = 1.559, Accuracy = 90.14%
Epoch 9: Loss = 1.562, Accuracy = 89.81%
Epoch 10: Loss = 1.560, Accuracy = 89.97%
Epoch 11: Loss = 1.560, Accuracy = 89.97%
Epoch 12: Loss = 1.560, Accuracy = 89.97%
Epoch 13: Loss = 1.562, Accuracy = 89.81%
Epoch 14: Loss = 1.559, Accuracy = 90.05%
Epoch 15: Loss = 1.557, Accuracy = 90.22%

Test Accuracy: 88.33%

Predictions:
test image predicted label: 4, Actual label: 4
test image predicted label: 5, Actual label: 5
test image predicted label: 1, Actual label: 1
test image predicted label: 4, Actual label: 4
test image predicted label: 1, Actual label: 1
```

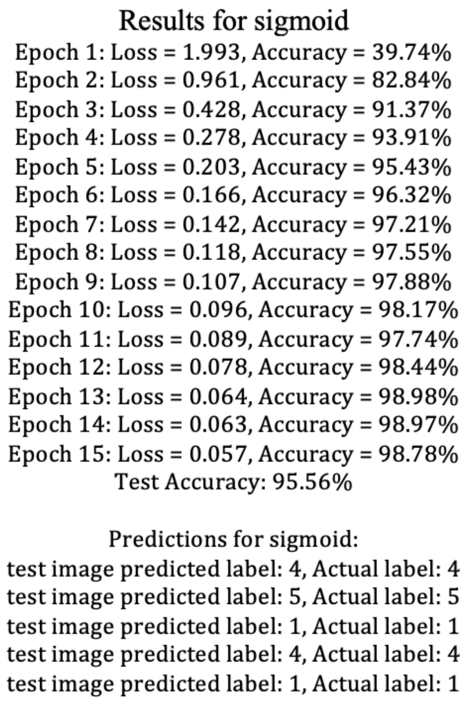
Different activation functions

The results of the Tanh and Sigmoid activation function including output, graphs and test accuracy:



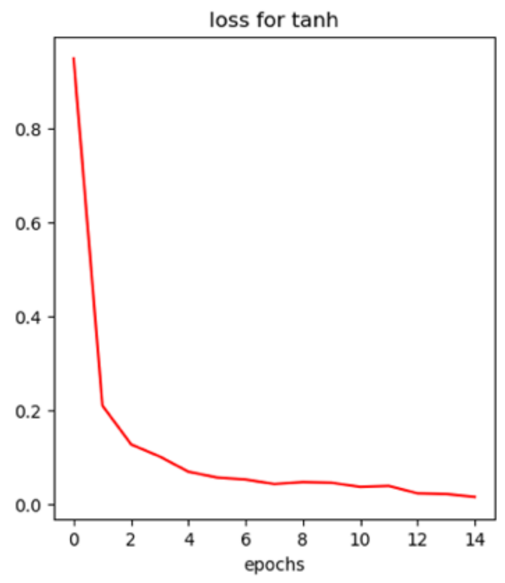
Results for tanh

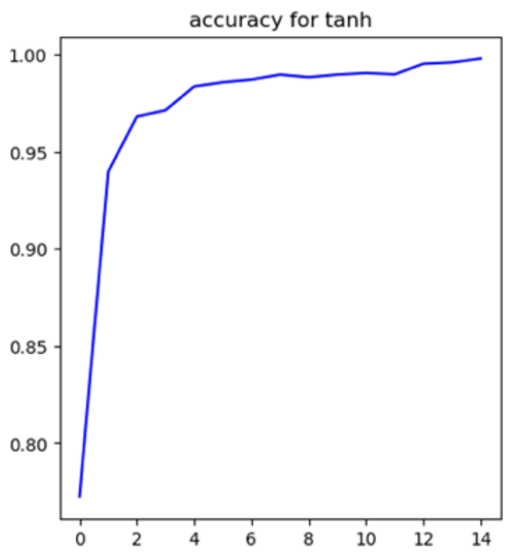
```
Epoch 1: Loss = 0.949, Accuracy = 77.24%
Epoch 2: Loss = 0.210, Accuracy = 93.97%
Epoch 3: Loss = 0.127, Accuracy = 96.81%
Epoch 4: Loss = 0.101, Accuracy = 97.13%
Epoch 5: Loss = 0.069, Accuracy = 98.36%
Epoch 6: Loss = 0.056, Accuracy = 98.57%
Epoch 7: Loss = 0.052, Accuracy = 98.71%
Epoch 8: Loss = 0.042, Accuracy = 98.97%
Epoch 9: Loss = 0.046, Accuracy = 98.83%
Epoch 10: Loss = 0.045, Accuracy = 98.97%
Epoch 11: Loss = 0.036, Accuracy = 99.05%
Epoch 12: Loss = 0.038, Accuracy = 98.98%
Epoch 13: Loss = 0.022, Accuracy = 99.52%
Epoch 14: Loss = 0.021, Accuracy = 99.59%
Epoch 15: Loss = 0.015, Accuracy = 99.80%
Test Accuracy: 96.39%

Example Predictions for Tanh Model:
Example 1 - Predicted Label: 4, Actual Label: 4
Example 2 - Predicted Label: 5, Actual Label: 5
Example 3 - Predicted Label: 1, Actual Label: 1
Example 4 - Predicted Label: 4, Actual Label: 4
Example 5 - Predicted Label: 1, Actual Label: 1
```

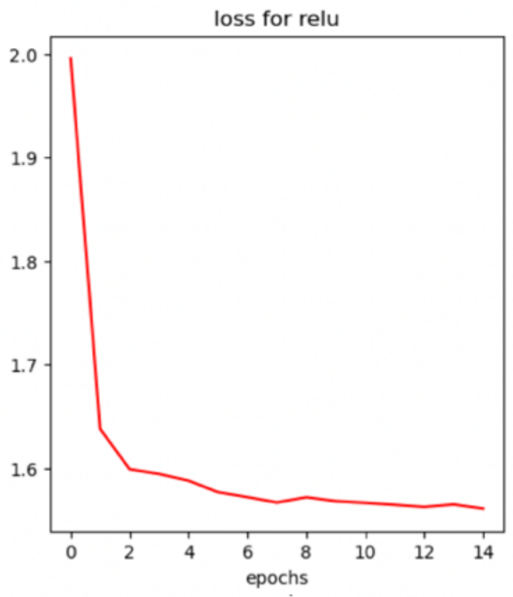


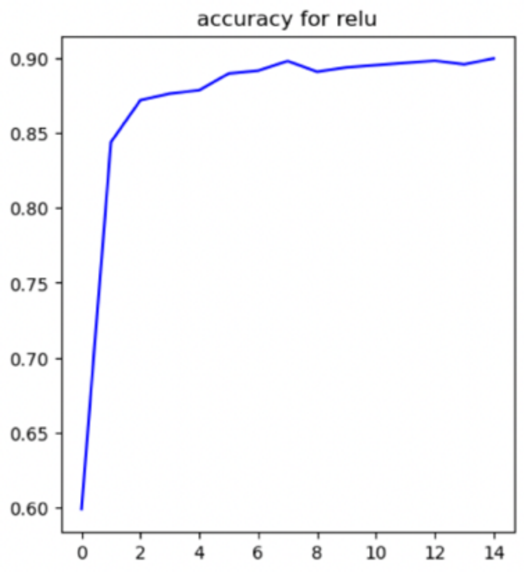
Results for sigmoid

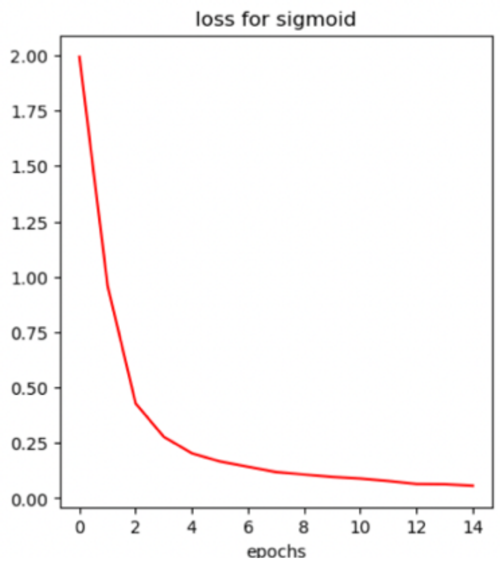
```
Epoch 1: Loss = 1.993, Accuracy = 39.74%
Epoch 2: Loss = 0.961, Accuracy = 82.84%
Epoch 3: Loss = 0.428, Accuracy = 91.37%
Epoch 4: Loss = 0.278, Accuracy = 93.91%
Epoch 5: Loss = 0.203, Accuracy = 95.43%
Epoch 6: Loss = 0.166, Accuracy = 96.32%
Epoch 7: Loss = 0.142, Accuracy = 97.21%
Epoch 8: Loss = 0.118, Accuracy = 97.55%
Epoch 9: Loss = 0.107, Accuracy = 97.88%
Epoch 10: Loss = 0.096, Accuracy = 98.17%
Epoch 11: Loss = 0.089, Accuracy = 97.74%
Epoch 12: Loss = 0.078, Accuracy = 98.44%
Epoch 13: Loss = 0.064, Accuracy = 98.98%
Epoch 14: Loss = 0.063, Accuracy = 98.97%
Epoch 15: Loss = 0.057, Accuracy = 98.78%
Test Accuracy: 95.56%

Predictions for sigmoid:
test image predicted label: 4, Actual label: 4
test image predicted label: 5, Actual label: 5
test image predicted label: 1, Actual label: 1
test image predicted label: 4, Actual label: 4
test image predicted label: 1, Actual label: 1
```

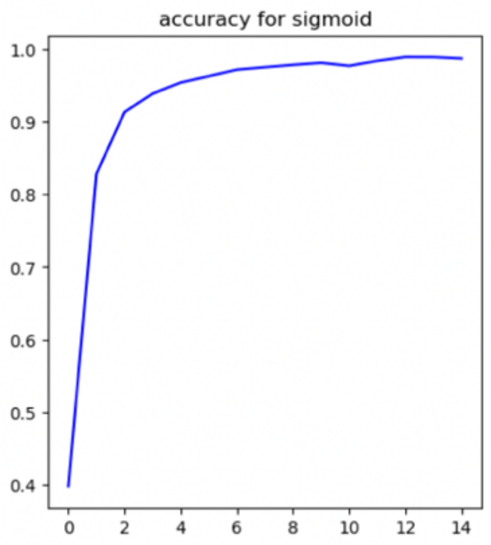












	RELU model	**Sigmoid model**	**Tanh model**
Test Accuracy score	88.61%	95.56%	96.39%

Convergence and Stability:
Tanh method shows the fastest convergence & highest final accuracy
Sigmoid method shows stable performance but slower convergence
ReLU method can be unstable towards the end but is efficient early on.

Types of activation functions:

Activation functions are chosen based on the different problems, architectures and datasets we are working with, I will define each function and what data/problem their best for. The sigmoid function works by mapping input values into a range using smooth terms, given the way the function works it is best used for binary classification problems. A negative to the sigmoid function is that it can cause a vanishing gradient problem which results in slow convergence. The tanh function works using a similar method to sigmoid by mapping input values to (-1,1), given this, the tanh function works best for neural networks that have a recurrent architecture, similar to the sigmoid model the tanh function can cause vanishing gradients which can be a con to the model. The reLU function works by returning the input if it is positive and zero if it is not, this is good as it avoids vanishing gradient and is the most computationally efficient, however, the negative to this model is that it can incorrectly return all zero values when it's not meant too.

Comparison:

The Sigmoid and Tanh models have higher test accuracy scores compared to the ReLU model. This suggests that the Sigmoid and Tanh activation functions might be more effective compared to ReLU. The tanh activation function led to the fastest convergence going from77% on the first epoch to 96.39% on the last. The tanh model also had the highest final accuracy (seen in the table above comparing them). In terms of stability, the sigmoid model seems the most stable and the relu model seems the most unstable with the curve becoming more unstable towards the end. The Prediction scores were all the same for each activation function, showing that despite the varying test accuracy scores and convergence levels the overall prediction abilities of the models are the same. Overall, based on the performance of the tests and the architecture of our data, the best activation function is the Tanh function. It achieved the highest test accuracy score and best convergence, making it well-suited for this specific task and dataset.

Adding More Hidden Layers and neurons

The architecture of a neural network directly impacts the models' ability to learn and predict. Hidden layers are one or more layers that perform computations through neurons and are not exposed to the input or output directly. Neurons are the basic units of a neural network. Adding more hidden layers will increase the model's ability to learn the complex patterns in the data. adding more neurons within a layer can allow the model to learn more and extract more features. For the digits data set this would help the model understand line thickness and possible curves in the data better, which could lead to better predictions. The negatives to this are that it can result in overfitting and increases the time to train the network therefore adding more hidden layers is a design choice that has to be made by considering the trade-off between underfitting and overfitting. For the digits data set the network I made has one hidden layer with 128 neurons, I don't believe adding more hidden layers or neurons Is necessary, well it may improve the model's ability to learn patterns, our model already has high predictive accuracy and so increasing the neurons or hidden layers will just result increase the risk of overfitting and amplifying the noise in the data.