MNIST Digit Classification

By Victoria Porter, Will Holbrook, Laura Cope, Lauren Cooper, Joel Wolinsky and Oisin Tunney

Agenda

- Method of Evaluation
- Non-neural techniques:
  - Random Forest
  - Multinomial Logistic Regression
  - K-nearest Neighbours
  - Support Vector Machine
- Neural techniques:
  - Shallow Neural Network
  - Deep Neural Network
  - Convolutional Neural Network
  - Recurrent Neural Network
- Conclusion
- Teamwork

Extract from the MNIST dataset

Method of Evaluation

- Constant Environment
  - Google Colab
  - GPU Hardware Accelerator
- Same Test and Training Dataset
- Multiple runs of tests
- Varying Parameters to find optimal solutions
- Performance metrics: confusion matrix and classification report

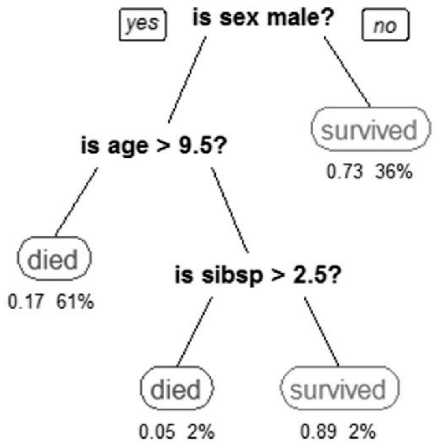
Non-Neural Techniques

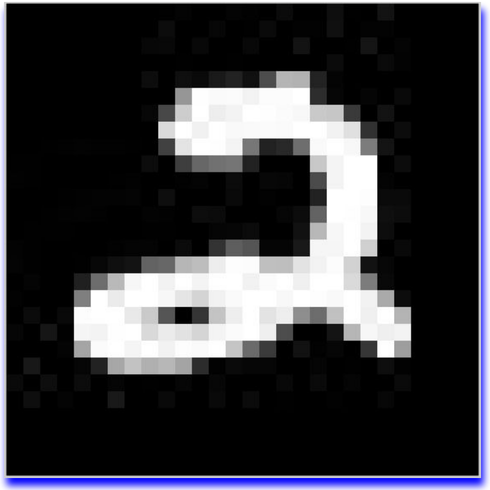
Accuracy: 0.9720
Overall Rank: 5th out of 8

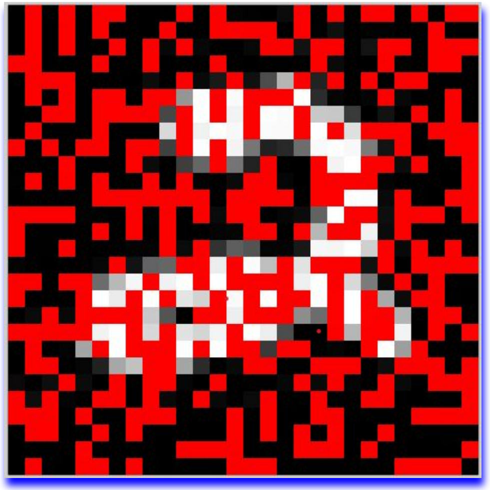
Non-Neural Technique: Random Forest Classifier

Random Forest works by generating a number of decision trees which split up the data into multiple classes

Here’s an example of a decision tree for survival of the titanic shipwreck:

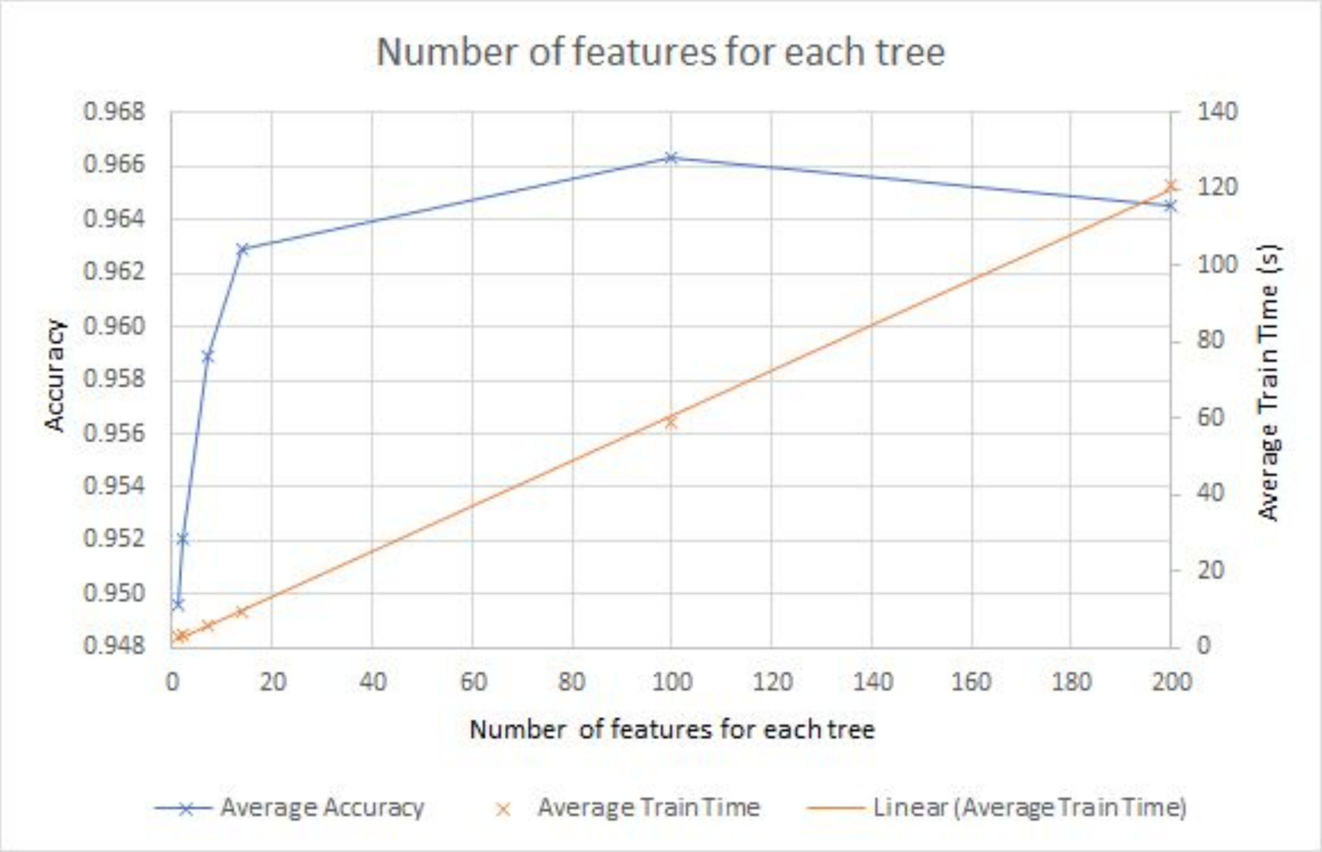






Accuracy: 0.9720
Overall Rank: 5th out of 8

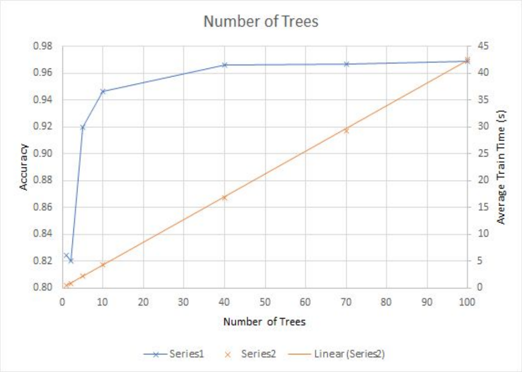
Non-Neural Technique: Random Forest Classifier

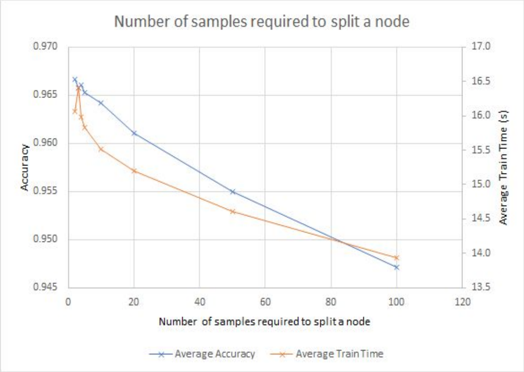


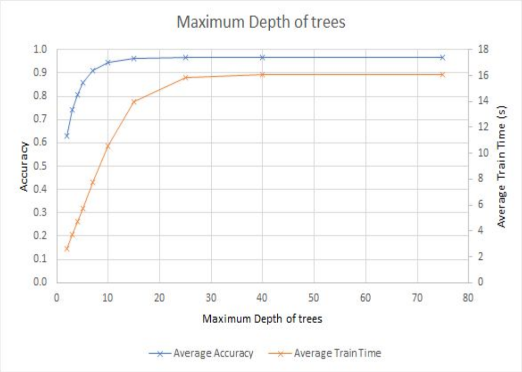
Accuracy: 0.9720
Overall Rank: 5th out of 8

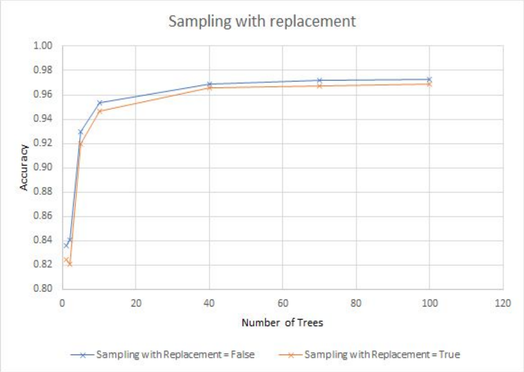
Non-Neural Technique: Random Forest Classifier

Effect of fine tuning Parameters







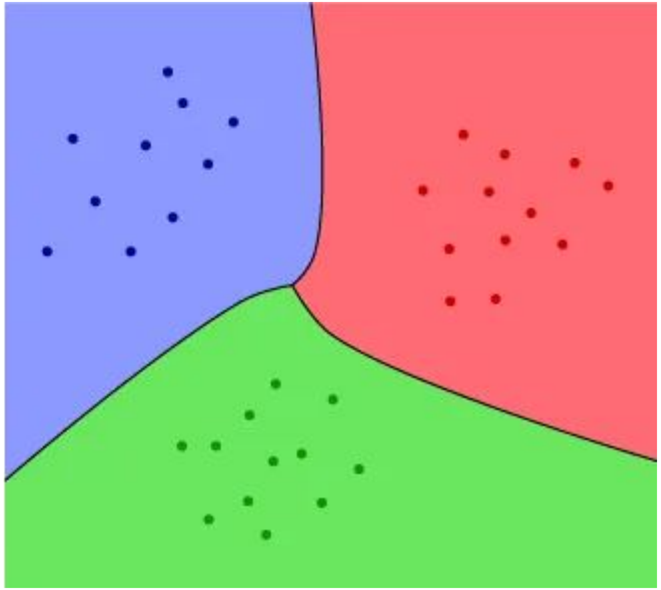


Optimised Parameters	
Number of Trees	100
Number of features for each tree	28
Sampling with replacement	FALSE
Number of samples required to split a node	2
Maximum Depth of trees	No Max Depth
Av. Training Time	69.3106
Av. Testing Time	0.5486

Accuracy: 0.9276
Overall Rank: 8th out of 8

Non-Neural Technique: Multinomial Logistic Regression

Multinomial Logistic Regression *is a supervised machine learning technique which works by generating a boundary line between the classes assuming that similar things are grouped together*

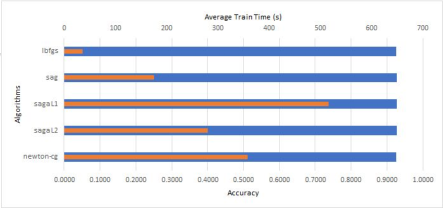


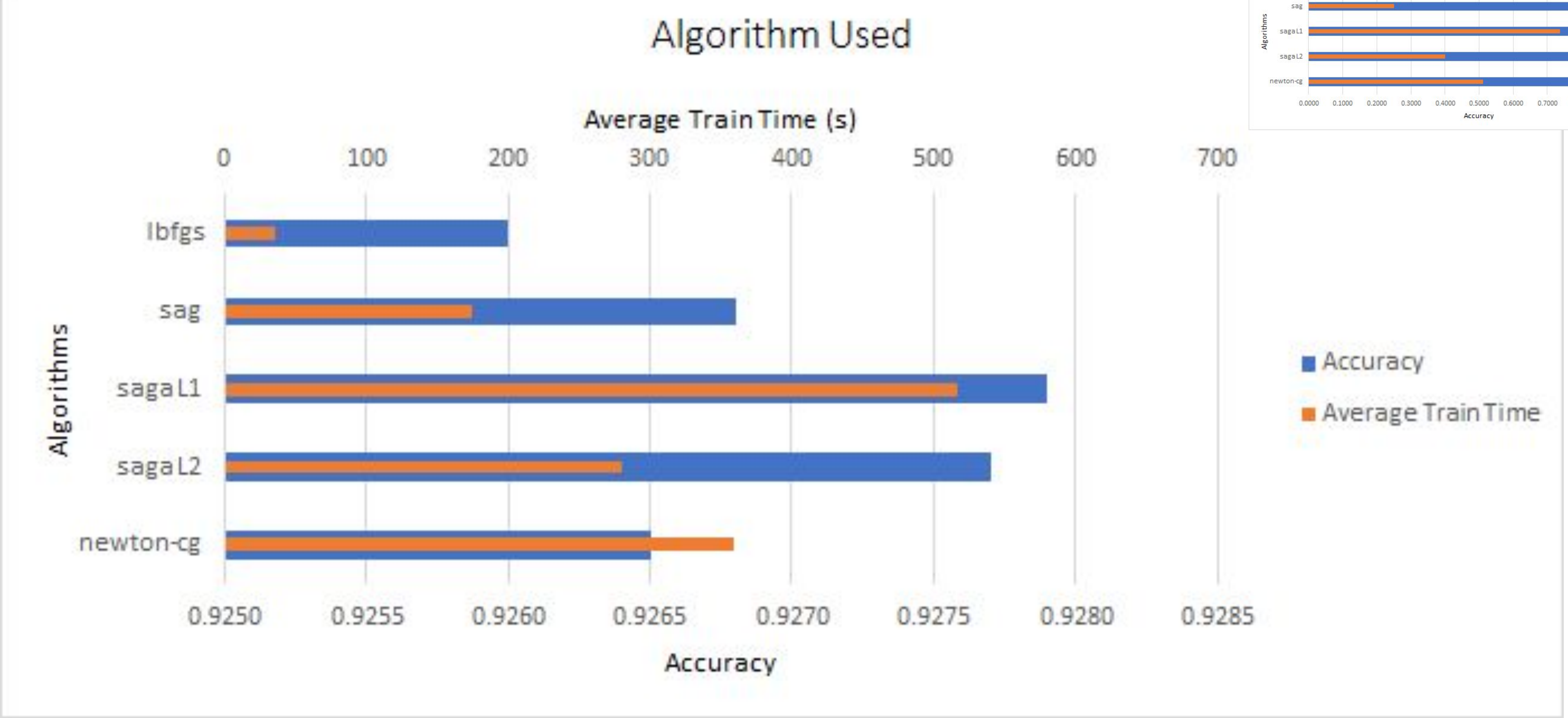
Parameters considered:

Inverse of regularization Strength (C)
Tolerance for Stopping criteria
Algorithm and type of penalty used

Accuracy: 0.9276
Overall Rank: 8th out of 8

Non-Neural Technique: Multinomial Logistic Regression

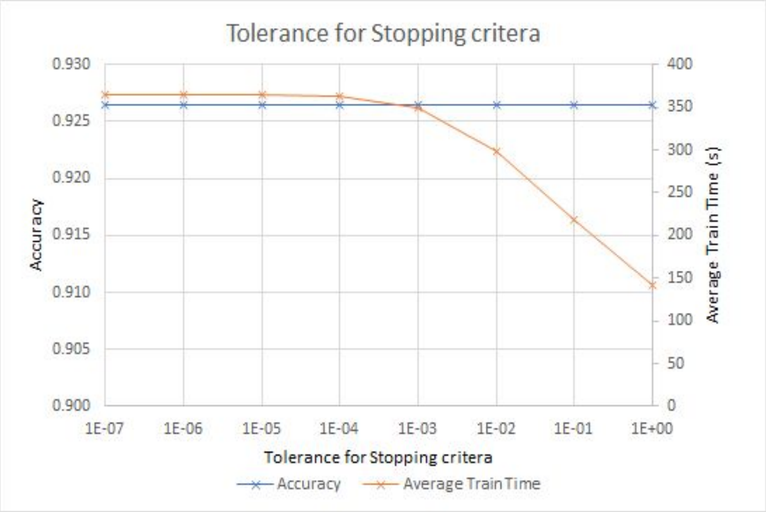


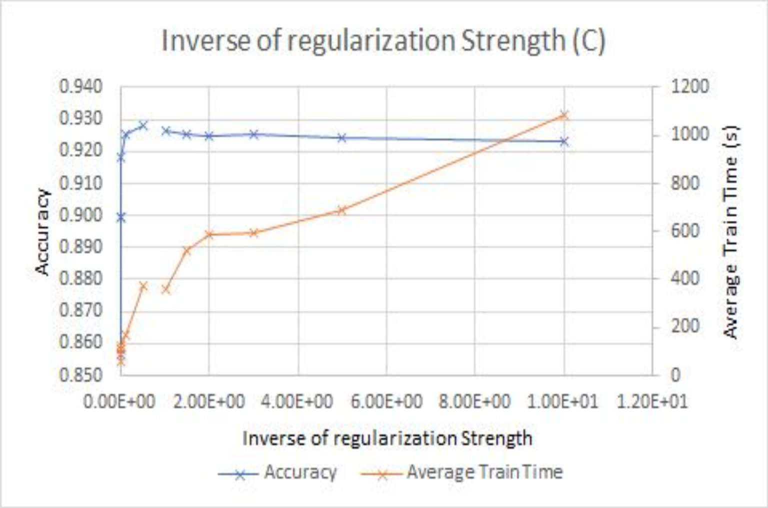


Accuracy: 0.9276
Overall Rank: 8th out of 8

Non-Neural Technique: Multinomial Logistic Regression

Effect of fine tuning Parameters



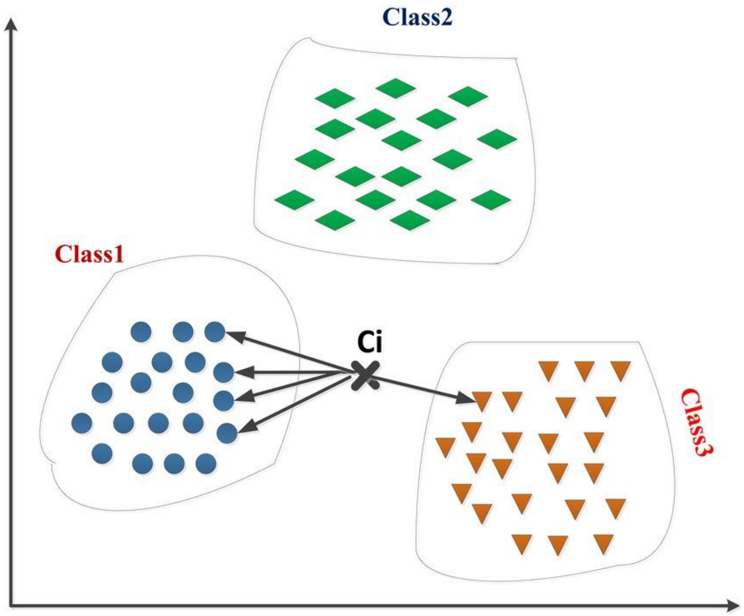


Optimised Parameters	
C	5.00E-01
Tolerance for Stopping Criteria	1.00E-04
Algorithm	sag
Type of penalty used	l2
Av. Training Time	58.9474
Av. Testing Time	0.0221

Accuracy: 0.9709
Overall Rank: 6th out of 8

Non-Neural Technique: K-nearest Neighbours (KNN)

KNN is a simple and versatile *supervised* machine learning algorithm which assumes that *similar things exist in close proximity*.



Parameters considered:

N_neighbours (K): number of neighbors to use.

Weights: weight function used in prediction.

Algorithm: algorithm used to compute the nearest neighbors.

Accuracy: 0.9709

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.99	0.98	980
1	0.97	1.00	0.98	1135
2	0.99	0.97	0.98	1032
3	0.97	0.96	0.97	1010
4	0.97	0.97	0.97	982
5	0.96	0.97	0.96	892
6	0.98	0.99	0.98	958
7	0.96	0.97	0.96	1028
8	0.99	0.94	0.96	974
9	0.96	0.96	0.96	1009
accuracy			0.97	10000
macro avg	0.97	0.97	0.97	10000
weighted avg	0.97	0.97	0.97	10000

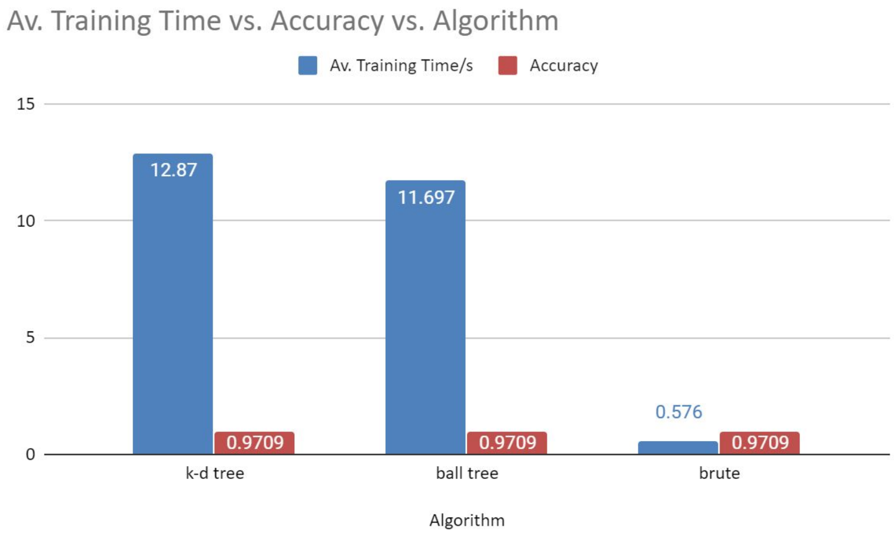
Accuracy: 0.9709
Overall Rank: 6th out of 8

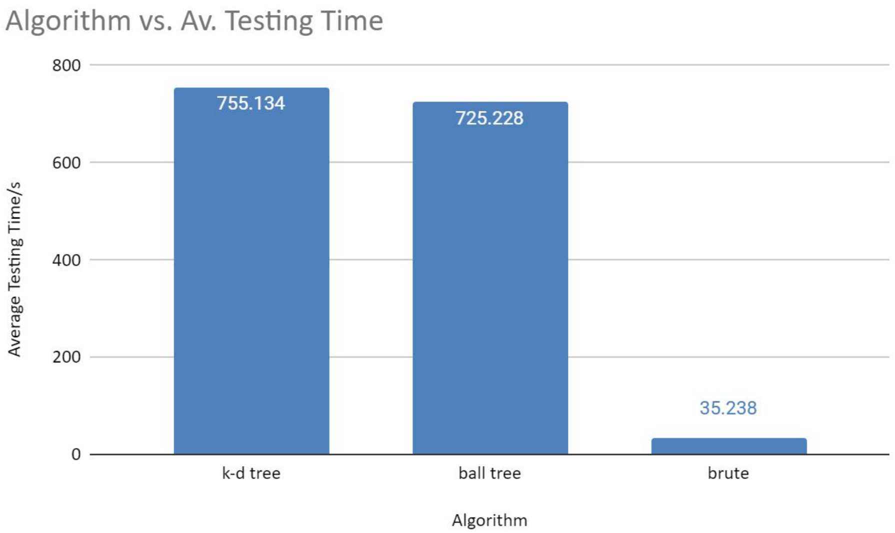
Non-Neural Technique: K-nearest Neighbours (KNN)

Optimal set-up for MNIST dataset:

K	Weights	Algorithm	Av. Training Time	Av. Testing Time	Accuracy
3	distance	brute	0.576s	35.238s	0.9709

- \+ Fastest total execution time.
- \+ Good accuracy.
- \+ Simple and versatile.
- \- Relatively slow prediction time.





Accuracy: 0.9788
Overall Rank: 4th out of 8

Non-Neural Technique: Support Vector Machine (SVM)

The objective of the SVM algorithm is to find the optimal *hyperplane* in an N-dimensional space.

Parameters considered:

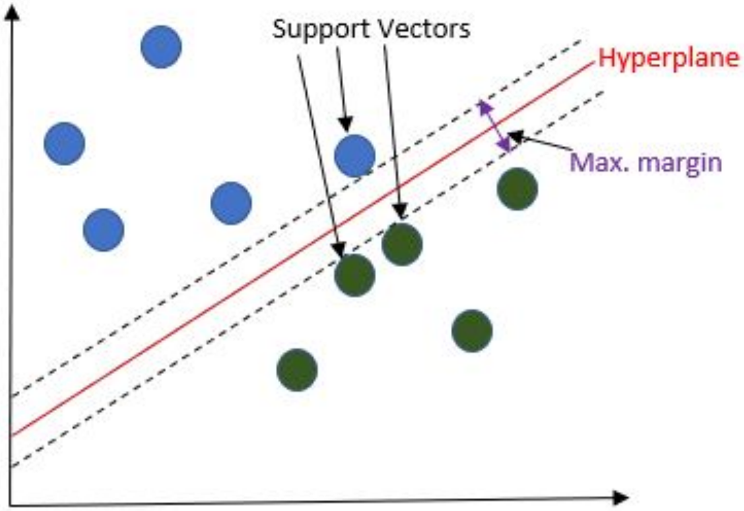
Kernel: A function which takes two data points as inputs and returns a **similarity score**.

C: A high C value makes correctly classifying all the training points more significant than leaving wiggle room for future data.

Gamma: A high gamma value will try to exactly fit the training dataset.

Class 7

Class 9



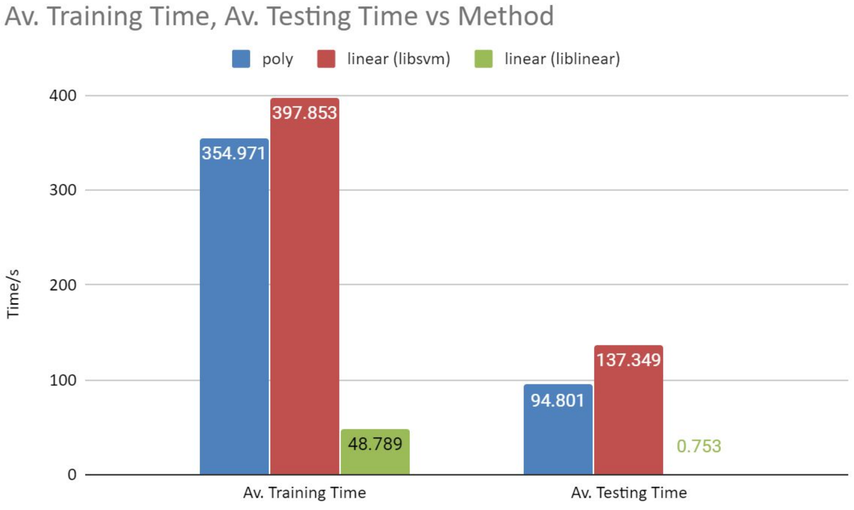
Accuracy: 0.9788
Overall Rank: 4th out of 8

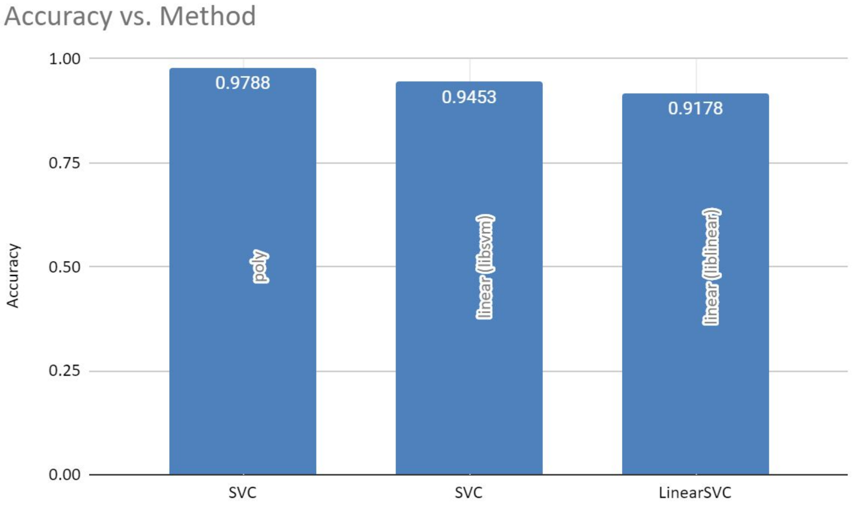
Non-Neural Technique: Support Vector Machine (SVM)

Optimal set-up for MNIST dataset:

Kernel	C	Gamma	Av. Training Time	Av. Testing Time	Accuracy
Poly	0.001	10	354.971s	94.801s	0.9788

- \+ Best non-neural implementation.
- \+ Memory efficient.
- \+ Good at classifying extreme samples.
- \- Slowest training and prediction times.

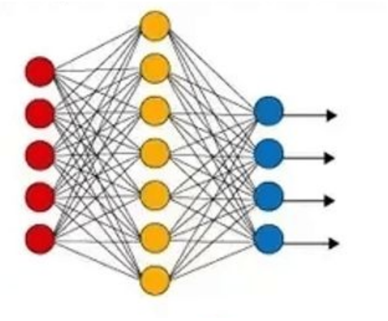




Neural Techniques

Accuracy: 0.9804
Overall Rank: 3rd out of 8

Neural Technique: Shallow Neural Network

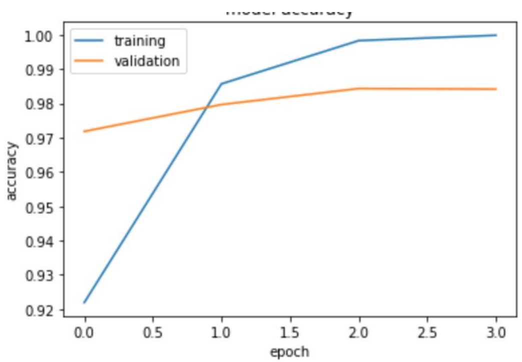


Parameters considered:

Layer width: A high hidden layer width led to a high accuracy

Batch size: A small batch size increased accuracy

Epochs: A small number of epochs tended to work better when in combination with a large layer width



Width	Batch Size	Epochs	Training time (s)	Testing Time (s)	Accuracy
100,000	60	3	91.137	0.889	0.9804

Accuracy: 0.963
Overall Rank: 7th out of 8

Neural Technique: Deep Neural Network

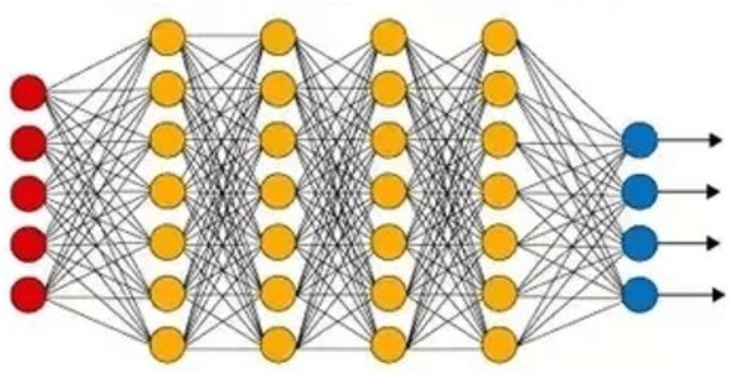
Parameters considered:

Width: The number of nodes on each layer

Depth: The number of layers

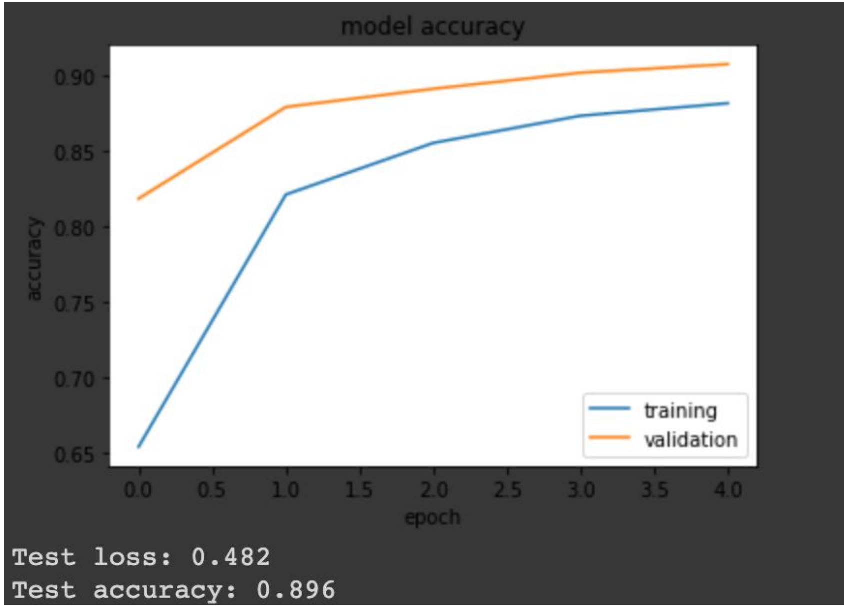
BatchSize: The number of samples processed before the model is update

Epochs: The number of complete passes through the training dataset

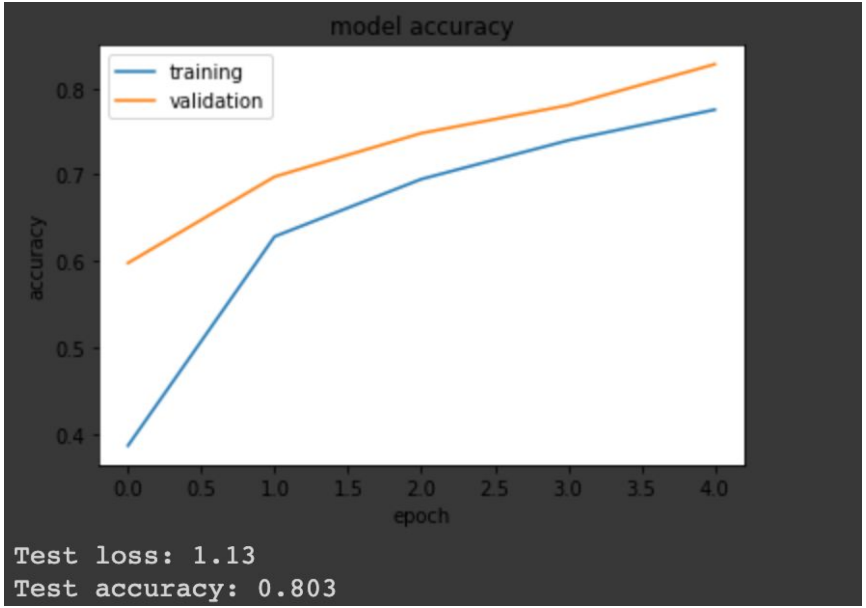


Accuracy: 0.963
Overall Rank: 7th out of 8

Neural Technique: Deep Neural Network



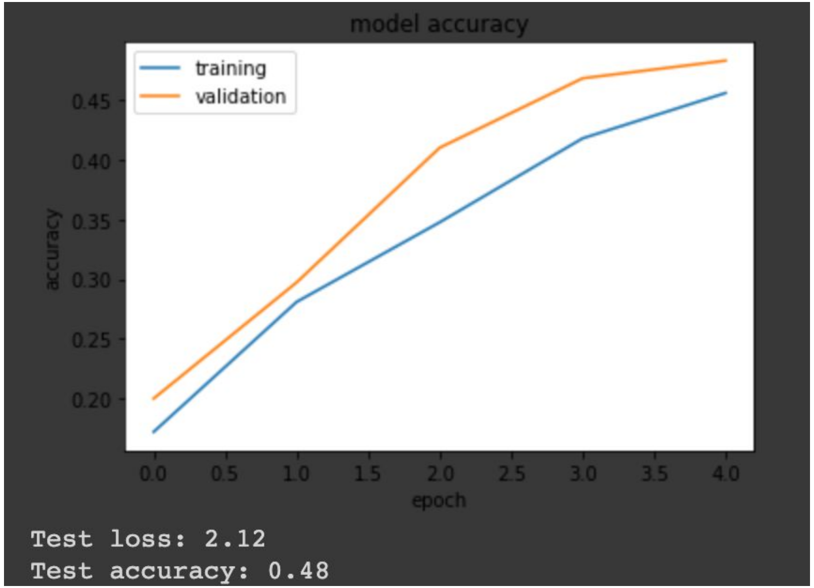
1 hidden layer



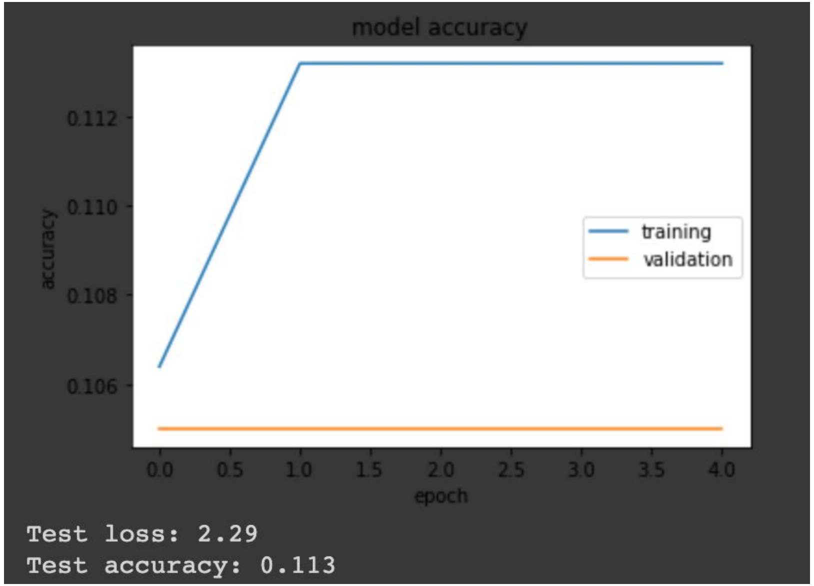
2 hidden layers

Accuracy: 0.963
Overall Rank: 7th out of 8

Neural Technique: Deep Neural Network



3 hidden layers

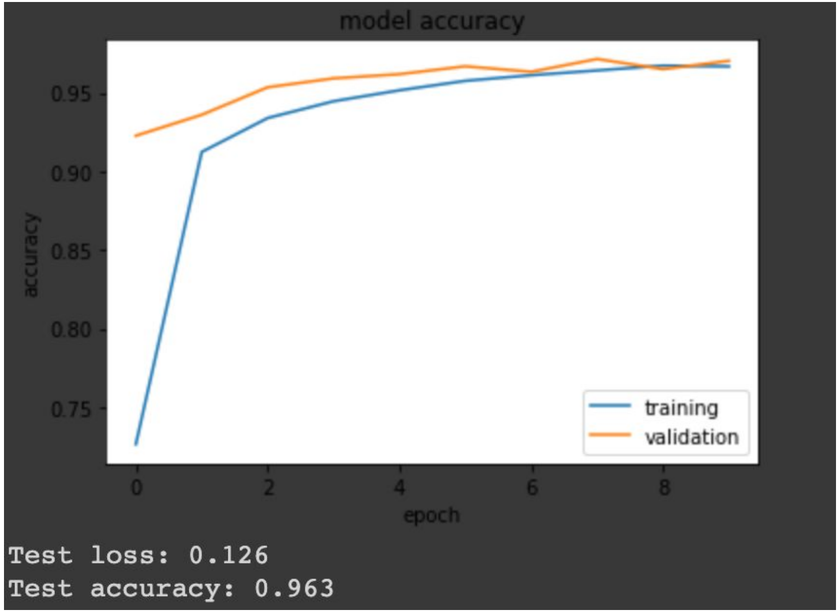


4 hidden layers

Accuracy: 0.963
Overall Rank: 7th out of 8

Neural Technique: Deep Neural Network

Width	Depth	Batch Size	Epochs	Training time	Testing Time	Accuracy
2048	3	16	10	145s	0.5s	0.963



- Adding layers leads to overfitting
- Decreasing batch size and increasing epochs increases training time a lot

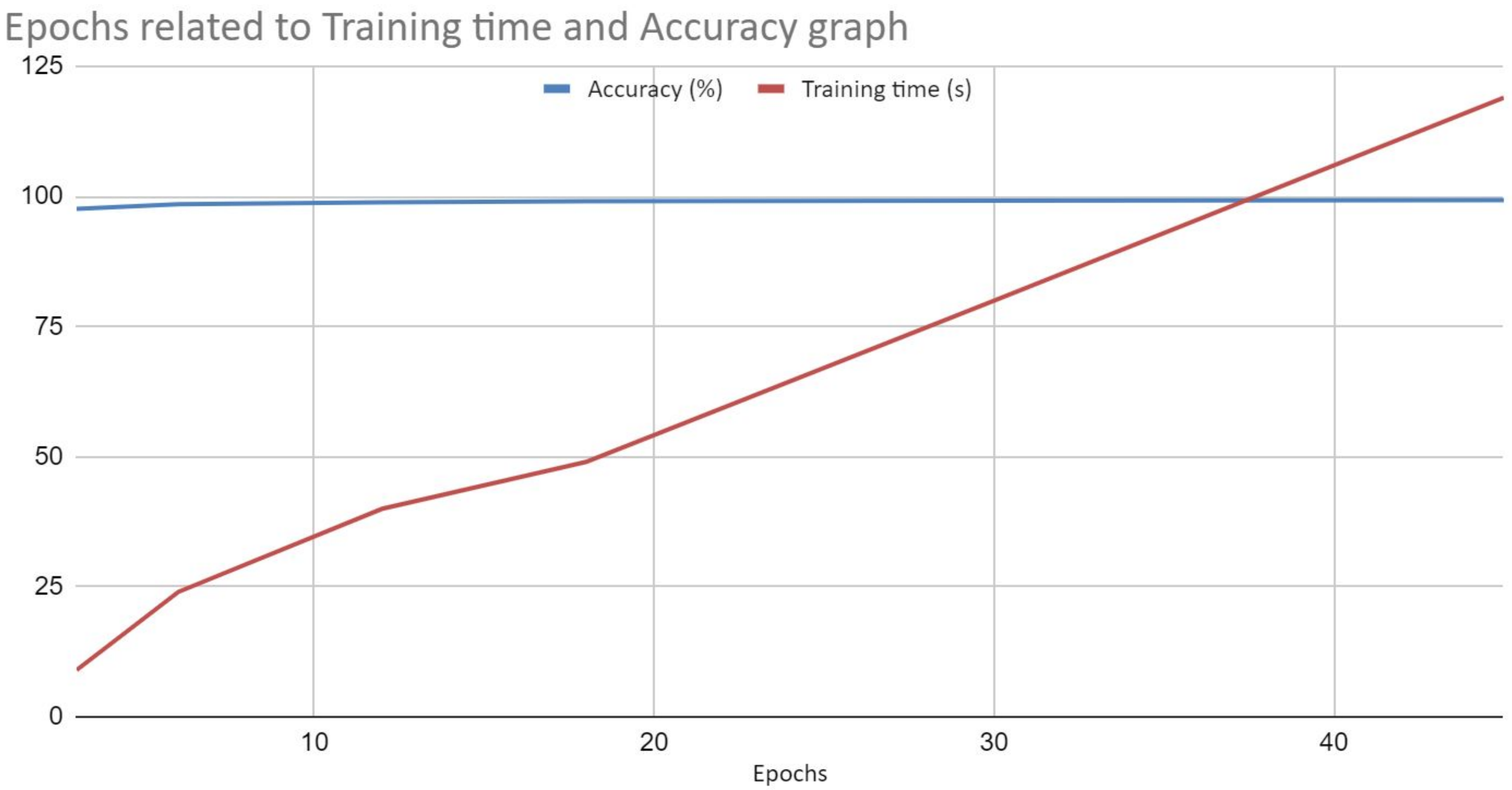
+ GPU works well in reducing training time

Accuracy: 0.9890
Overall Rank: 1st out of 8

Neural Technique: Convolutional Neural Network

CNN uses multilayer perceptrons to do computational works and use relatively little pre-processing compared to other image classification algorithms, the network learns through filters that in traditional algorithms are hand engineered. These facts lend to Convolutional Neural Networks being best suited for image processing tasks such as processing the MNIST database. As found by my tests.

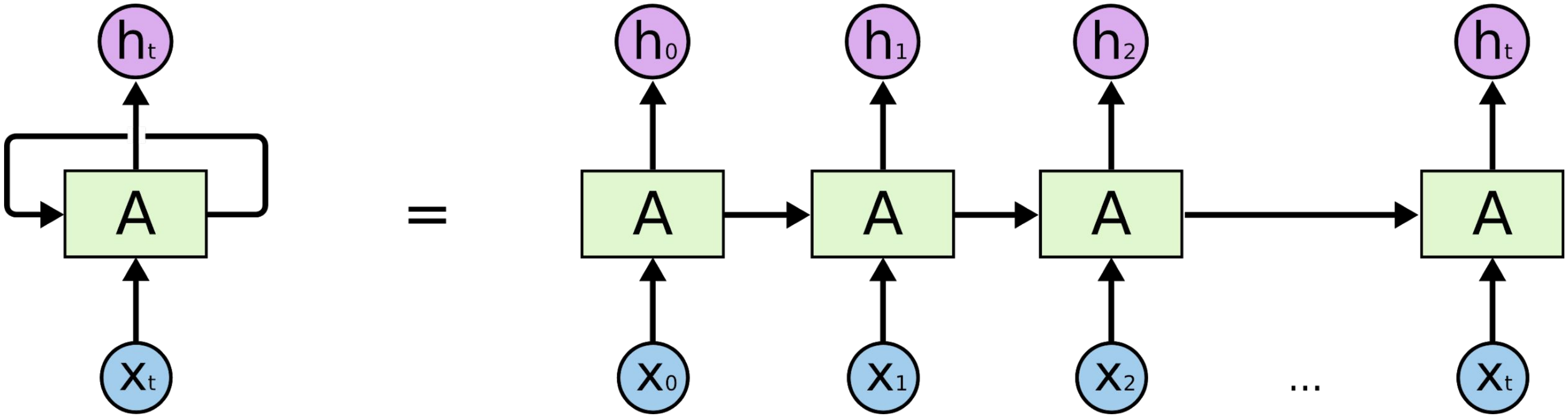
Epochs	Training times (s)	Testing times (s)	Accuracy (%)
12	40	1.7	98.9



Epochs related to Training time and Accuracy graph
Accuracy (%)
Training time (s)
125
100
75
50
25
0
10
20
30
40
Epochs

Accuracy: 0.9831
Overall Rank: 2nd out of 8

Neural Technique: Recurrent Neural Network



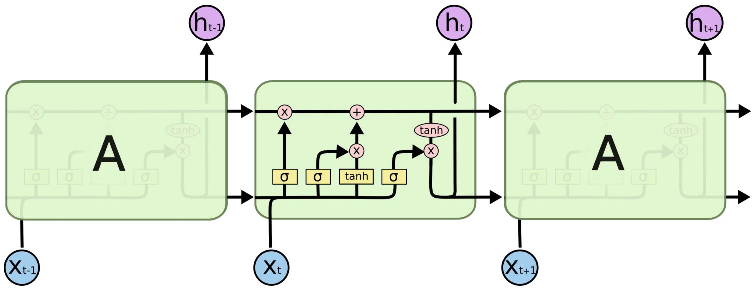
- These networks consider time and sequence and ‘remember’ previous time-steps
- Outputs are fed back into the node with new inputs in a feedback loop
- The network can pick up patterns in the sequence of inputs this way

Accuracy: 0.9831
Overall Rank: 2nd out of 8

Two types of recurrent layers explored in this project

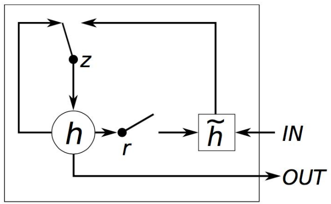
LSTM

- ‘Long Short-Term Memory Units’ are designed to have a longer ‘memory’ or dependency than simple RNNs, aka capable of remembering previous information from many time-steps back
- Each cell is quite complicated, containing layers inside itself, and other functions and operations



GRU

- ‘Gated Recurrent Units’ are simpler versions of LSTMs, containing fewer parameters and internal gates
- The performance of GRUs is similar to that of LSTM on many tasks such as music modelling, speech and natural language processing but LSTM is still considered to be stronger for language translation and other more complicated data



Accuracy: 0.9831
Overall Rank: 2nd out of 8

RNN Model Testing Process

- Built a model in Python using the Keras tensorflow library
- Model consists of multiple layer, with either a main GRU or LSTM layer
- Other layers added - dense layers, simple RNN layer, further GRU/LSTM layers, and dropout layers
- Vary the number of layers
- Vary the activation function on these layers (ReLU or Softmax)
- Vary the value passed to the Dropout layers (value between 0 and 1 - the fraction of the input units to drop)

Why?

Dropout helps to prevent **overfitting** - when models train too well to a specific dataset and then perform badly on unseen data.
Experimenting with different model architectures and recording their accuracies allows us to identify successful approaches.

Accuracy: 0.9831
Overall Rank: 2nd out of 8

RNN Conclusions

Experiment Findings

- Majority of models returned a >0.95 accuracy
- Models that had a poor accuracy had fewer layers, no dropout layers/ too little dropout
- LSTM models worked best with about two LSTM layers, followed by two dense layers, with dropout layers as padding between each.
- Dropout was most effective with a parameter value of 0.1-0.3 (10-30% of nodes dropped)

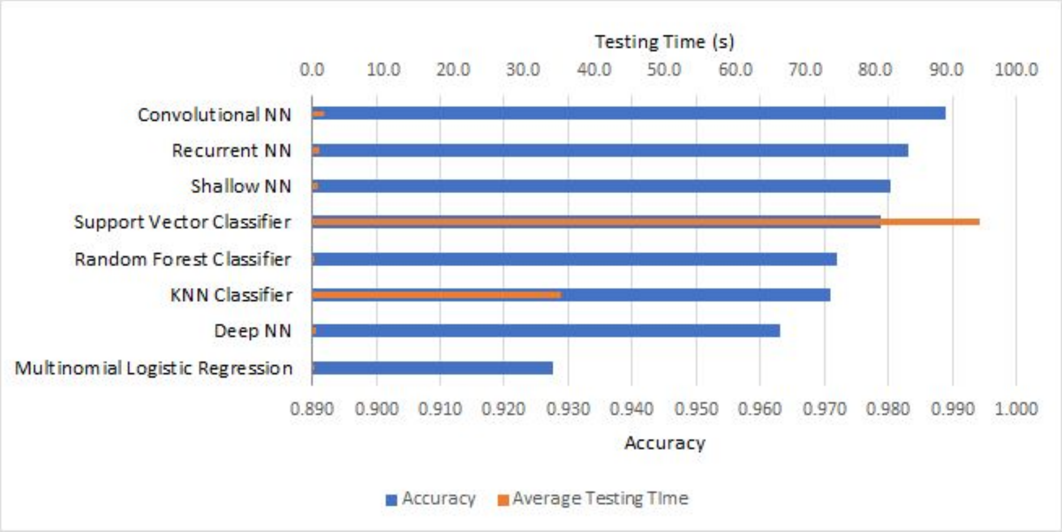
The best models:

Model description	time	accuracy
GRU layer, 0.2 dropout, dense layer (relu), dense layer (softmax)	57s	0.9831
GRU layer, 0.2 dropout, dense layer (relu), 0.1 dropout, dense layer (softmax)	57s	0.9819
LSTM layer (relu), 0.1 dropout, LSTM layer (relu), 0.3 dropout, dense layer (relu), 0.3 dropout, dense layer (softmax)	113s	0.9808
LSTM layer (relu), 0.2 dropout, LSTM layer (relu), 0.2 dropout, dense layer (relu), 0.2 dropout, dense layer (softmax)	82s	0.9805

Conclusion

Rank	Technique	Average Training Time (seconds)	Average Testing Time (seconds)	Accuracy
1	Convolutional NN	40.3	1.7	0.989
2	Recurrent NN	57.27	1.04	0.9831
3	Shallow NN	91.131	0.8889	0.9804
4	Support Vector Classifier	354.971	94.801	0.9788
5	Random Forest Classifier	47.4538	0.3530	0.9720
6	KNN Classifier	0.576	35.238	0.9709
7	Deep NN	145	0.5	0.963
8	Multinomial Logistic Regression	58.9474	0.0221	0.9276


Training Time (s)
0.0
50.0
100.0
150.0
200.0
250.0
300.0
350.0
400.0
Convolutional NN
Recurrent NN
Shallow NN
Support Vector Classifier
Random Forest Classifier
KNN Classifier
Deep NN
Multinomial Logistic Regression
0.890
0.900
0.910
0.920
0.930
0.940
0.950
0.960
0.970
0.980
0.990
1.000
Accuracy
Accuracy
Average Train Time


Testing Time (s)
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
100.0
Convolutional NN
Recurrent NN
Shallow NN
Support Vector Classifier
Random Forest Classifier
KNN Classifier
Deep NN
Multinomial Logistic Regression
0.890
0.900
0.910
0.920
0.930
0.940
0.950
0.960
0.970
0.980
0.990
1.000
Accuracy
Accuracy
Average Testing Time

Teamwork

Every project needs a team behind it that is passionate for the project and most importantly of all a team that has chemistry to work well with each other.

By tracking how the team is working and by using methods from Tuckman's model on team development we could avoid conflicts.

Teamwork Overview

- During our first meeting we focussed on breaking the ice between all team members and helped by certain members of the team taking leadership and pushing the drive and from this splitting up the work effectively and to get started fast.
- We continued to have meetings every 3/4 days to check up on everyone's work and share results and how it is all going, keeping team morale and drive up by using simple strategies such as recognizing individual and group efforts and constantly monitoring the ‘energy’ of the group.
- Reaching the end of the project where our focus was entirely to share results and prepare a presentation so this is when teamwork was needed most

Questions?