2018

Siamese Network

IFN680 ASSIGNMENT 2

TRACEY WRIGHT – N9131302

Siamese Network

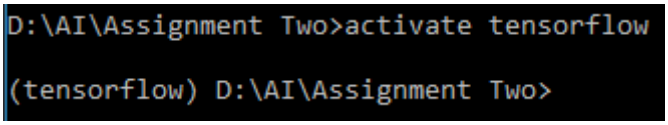
The purpose of this implementation of a Siamese Network is to differentiate a pair of images. Contrast to a regular CNN, instead of learning to classify images, the Siamese Network learns the similarity between two (or more) images. This is useful for automated recognition.

Technically, the Siamese Network learns the characteristics of a set of training data to produce a model that is representative of the features of the training data. The model is comprised of a polynomial set of weights, which when applied to test data, will predict whether a pair of images are the same or different.

A Siamese Network architecture consists of two identical subnetworks which share the same model. This is followed by a distance calculation layer, which evaluates the Euclidean distance between image pairs. The final layers of the two subnetworks are then fed to a contrastive loss function, which evaluates the similarity between the two images.

Implementation

The code requires local installation of tensorflow and activation in a command prompt as shown below.



The MNIST dataset is loaded from Keras, pooled to one set to make use of all available data, and saved to file in the current working directory. This step may be switched off by comment once the initial load has been done.

The data is basically comprised of X, Y, where X is image data (images of digits 0 to 9) and Y is labels where the index of Y corresponds to the index of X.

The data is loaded from the local directory and the X values are typed as float32, scaled between 0 and 1, and reshaped to a format that the model compiler will recognise.

Indices for Y values are grouped based on whether their values (the class labels) are [2,3,4,5,6,7], [0,1,8,9], or [0,1,2,3,4,5,6,7]. Indices group [2,3,4,5,6,7] is then randomised and split (80:20) into training and test sets, for the test set [2,3,4,5,6,7] to be used during training validation. [0,1,8,9] is not used during training.

X data and Y digit indices are passed to a function for creating pairs. The create_pairs function alternates between creating positive and negative pairs, which are pairs with class labels that match, and pairs with class labels that don't match, respectively. This will result in an equal amount of positive and negative pairs for training. The create_pairs function returns two sets of pairs with labels.

A base network is created via the create_base_network function, which will be shared across the two branches. The base network is comprised of layers from the keras library. The layers configuration from IFN680 CNN tutorial was used for initial model training attempts. The layers configuration was experimented with and optimised during subsequent model training attempts.

Distance is measured via a Euclidean distance function, which is passed to a Keras Lambda object. Lambda combines the distance function along with the two branch networks to produce a custom layer for the model.

The model is compiled with the contrastive loss function as a parameter. The contrastive loss function allows the model to optimise and learn by providing feedback regarding the difference between true values and predicted values.

A custom callback is used to collect validation errors over time, which writes collected data to csv files. Custom functions have also been created to collect precision and recall metrics which are also output to csv.

Training and Optimisation

Neural network training processes generally require optimisation. Factors which may be tuned include the selection of layers to use for the neural net and the number of epochs to run. The goal is to create a model that generalises well to all test cases, in addition to providing a combination of invariance and discriminability.

The first implementation attempt was an arbitrary utilisation of the Keras layers used in the IFN680 CNN tutorial (Conv2D, Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Dropout, Dense) with 10 epochs. Model training and validation error was recorded for each epoch over time.

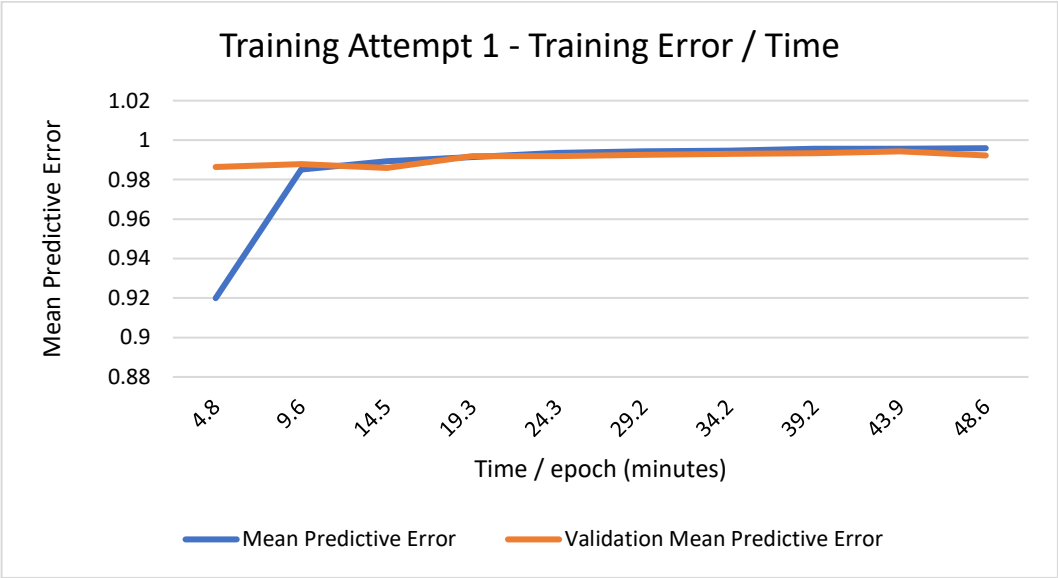


Figure 1. First Training Attempt. Training and validation error over time. Times recorded at the end of each epoch.

The first thing to notice from these results is that the error values are small, which is a positive indication that the model is converging, and that the training will produce a useful model.

The results also demonstrate that 10 epochs are far too many. This is clear from the plateau (and even slight increase) in training and validation error over time. The model took an unnecessarily long time to train for this run, primarily because of the excessive number of epochs, so this will be decreased for the next training attempt. It's also worth considering whether a variant model architecture will produce accurate results in less time per epoch, which will be done in a later training attempt.

The slight increase in validation error seen in the graph for training attempt one is an indication of a small amount of overfitting, which although subtle and minor in this case, is something that should be prevented where possible. Removing the excessive epochs should help with preventing overfitting.

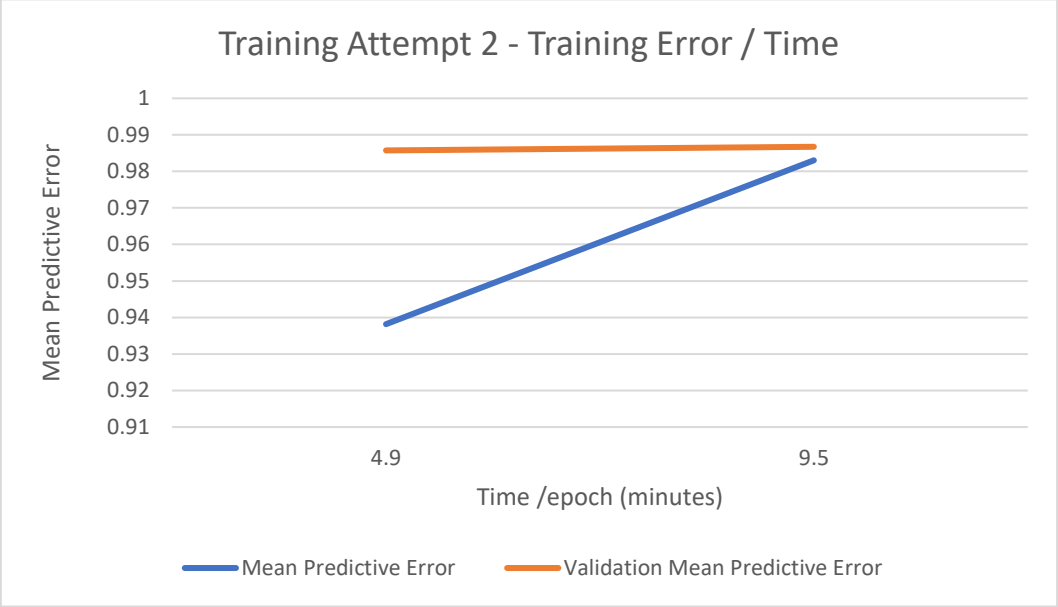


Figure 2. Second Training Attempt. Training and validation error over time. Times recorded at the end of each epoch.

Two epochs for the second training attempt appears to be plenty, and the total training time has improved from almost an hour to less than 10 minutes. The slight increase in validation error indicates that one epoch is better than two, but I'll continue to run at least two epochs to gather test metrics throughout the model optimisation process for transparency purposes.

Precision and recall metrics were collected for all test cases ([2,3,4,5,6,7], [0,1,2,3,4,5,6,7,8,9], [0,1,8,9]) during the second training attempt to evaluate the capacity of the model. Precision is a

ratio of true positives to true positives plus false positives, which is a measure of the ability of the classifier not to label a positive a sample that is negative. Recall is a ratio of true positives to true positives plus false negatives, which is a measure of the ability of the classifier to find all the positive samples. These metrics will provide feedback about the model performance.

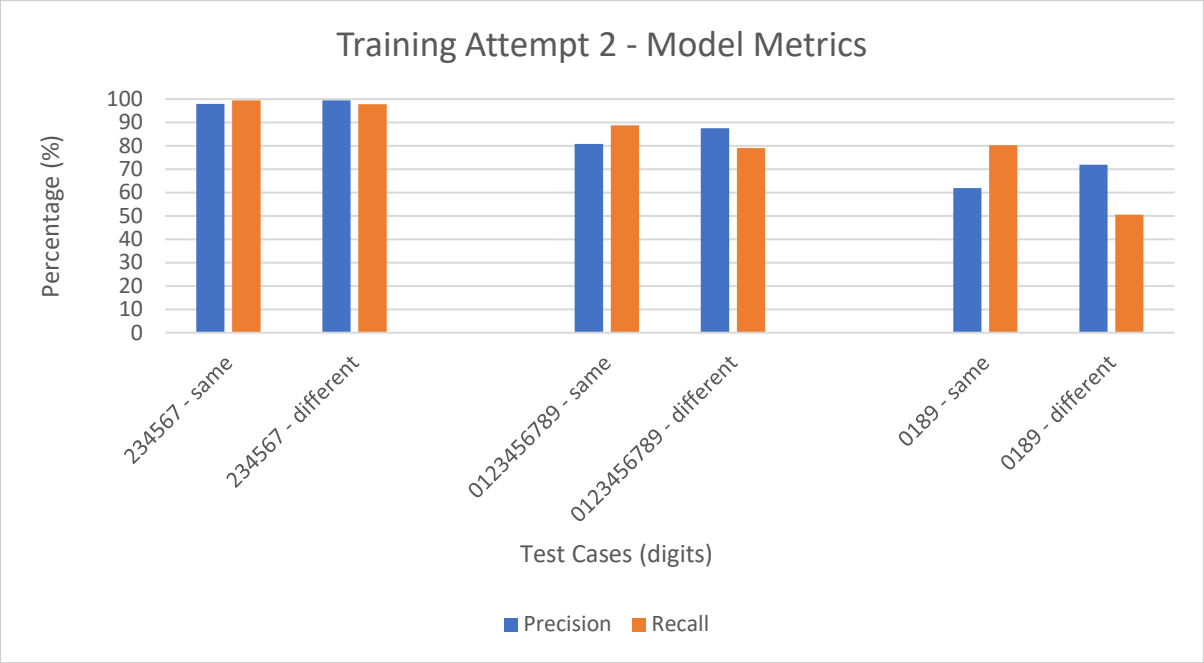


Figure 3. Second Training Attempt - Precision and recall metrics for each test case [2,3,4,5,6,7], [0,1,2,3,4,5,6,7,8,9], [0,1,8,9]. Positive matches (same) and negative matches (different) are shown individually to demonstrate the model capacity for discriminating between digits that are the same and digits that are different.

The model performance for the test cases of second training attempt is generally what I would expect to see, that is:

- A high performance for test case [2,3,4,5,6,7], given that the training was performed on [2,3,4,5,6,7] digits;
- A significant reduction in performance for [0,1,2,3,4,5,6,7,8,9];
- Another significant reduction in performance for test case [0,1,8,9].

These results are consistent with what's expected for a model which is testing digits that it hasn't seen examples of before. The apparent linear reduction in performance from [2,3,4,5,6,7] to [0,1,2,3,4,5,6,7,8,9] to [0,1,8,9] is attributable to the proportion of unseen digits in each of the test cases.

In terms of precision and recall, the following trends have been observed:

- Across all test cases, positive pairs (same) were more sensitive (higher recall than precision) and less specific.

- Across all test cases, negative pairs (different) were more specific (higher precision than recall) and less sensitive.

In general, the model has less false negatives for digits that are the same, and more false negatives for digits that are different. Conversely, there are generally more false positives for digits that are the same, and less false positives for digits that are different.

As the base network was created with the same layers used for the tutorial as a starting point, further optimisation could be done by testing variations of the model layer configurations in attempt to produce a model with higher precision and recall. Further, it's worth experimenting with layer configurations which may allow the code to run faster, although improved precision and recall is the priority.

Given that the data set is relatively small and simplistic already and that dimensionality reduction is not necessarily required, and with knowledge that the general purpose of the initial layers is to reduce data complexity whilst maintaining the characteristics of the data, there's opportunity with this data set to try subverting the dimensionality reduction layers and head straight into implementing core keras layers.

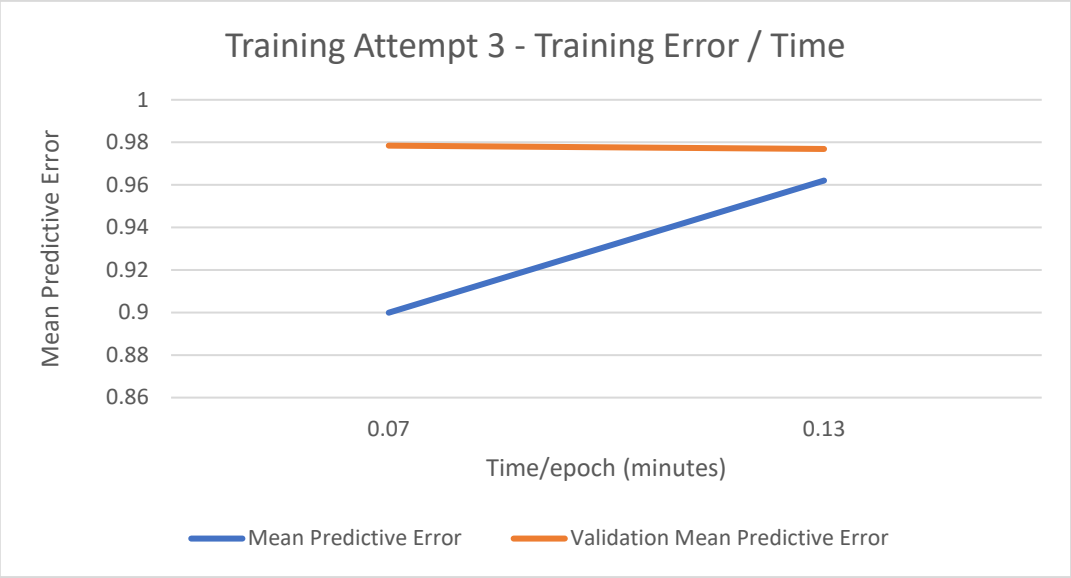


Figure 4. Third Training Attempt. Training and validation error over time. Times recorded at the end of each epoch.

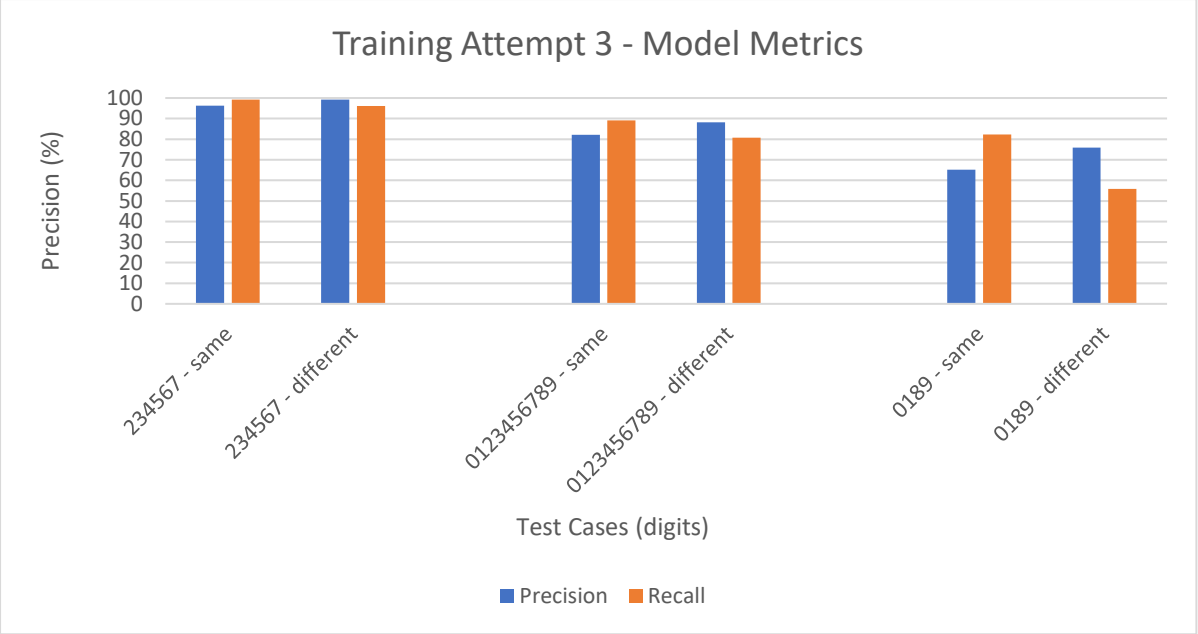


Figure 5. Third Training Attempt - Precision and recall metrics for each test case [2,3,4,5,6,7], [0,1,2,3,4,5,6,7,8,9], [0,1,8,9]. Positive matches (same) and negative matches (different) are shown individually to demonstrate the model capacity for discriminating between digits that are the same and digits that are different.

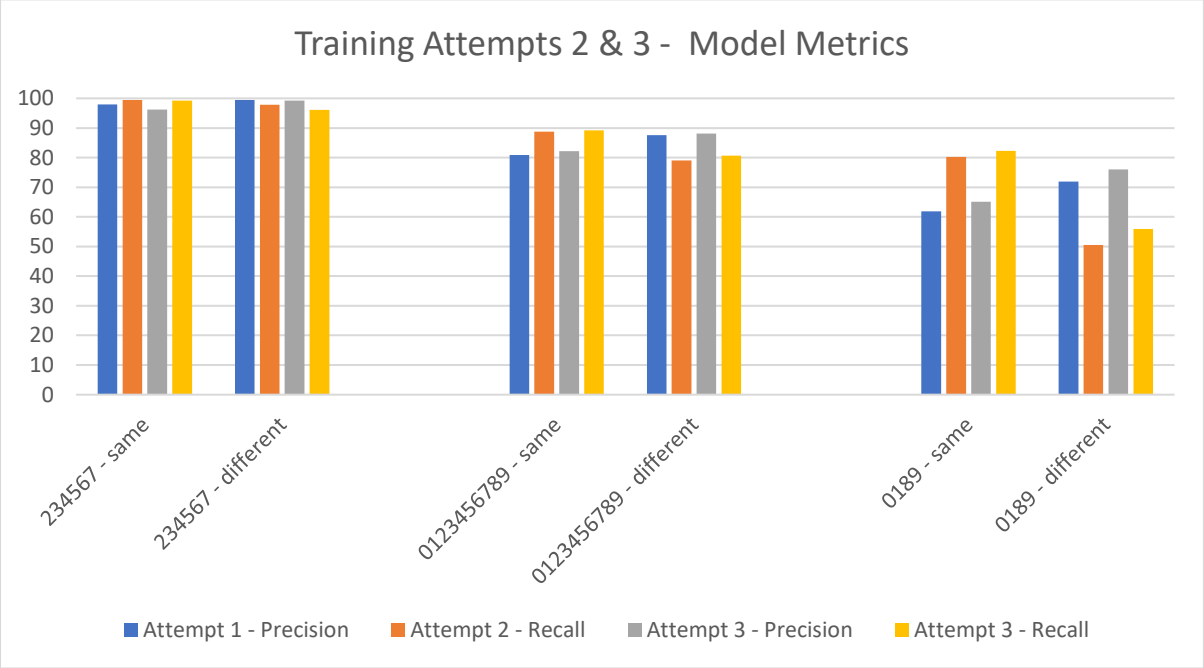


Figure 6. Second and Third Training Attempt - Precision and recall metrics for each test case [2,3,4,5,6,7], [0,1,2,3,4,5,6,7,8,9], [0,1,8,9]. Positive matches (same) and negative matches (different) are shown individually to demonstrate the model capacity for discriminating between digits that are the same and digits that are different.

While the precision and accuracy metrics for test case [2,3,4,5,6,7] are much the same as for previous attempts, the metrics for test cases [0,1,2,3,4,5,6,7,8,9] and [0,1,8,9] are marginally increased. The third training attempt has also taken dramatically less time than previous attempts, with epochs completing in a matter of seconds, rather than minutes as seen with previous training attempts.

Network Robustness

To demonstrate robustness of this Siamese Network 20 individual training runs were performed and the accuracy of each training and test case was recorded. The Network was found to be produce consistent results over many training instances and is therefore robust.

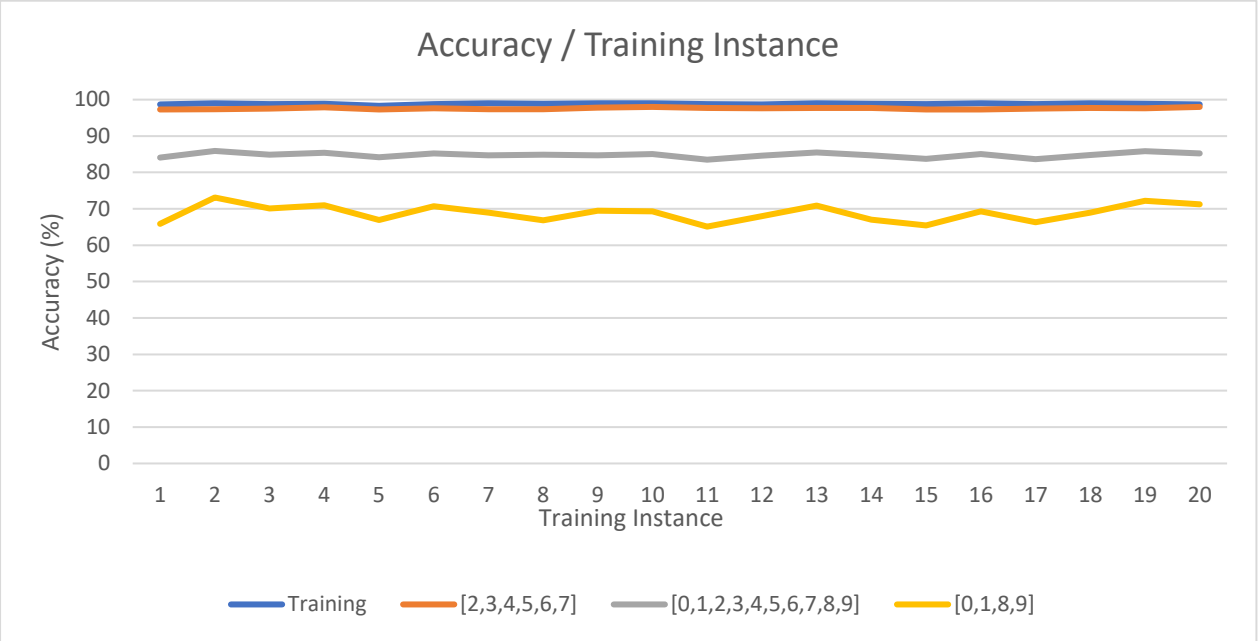


Figure 7. Training and Test Accuracy for 20 training instances. Accuracy for training and each of the 3 test cases is shown.

Training	[2,3,4,5,6,7]	[0,1,2,3,4,5,6,7,8,9]	[0,1,8,9]
98.65	97.6	84.8	68.8

Figure 8. Average Training and Test Accuracy over 20 training instances.

Conclusion

This assignment has demonstrated an effective implementation of a Siamese Network for the MNIST data set. The model produced by the Siamese Network recognises whether digits are the same or different with approximately 99% accuracy. Further, the model extends its capacity beyond predicting digits seen during training to digits that it hasn't seen before, with an accuracy significantly greater than random.