COMP3105 A4 Report
Ryan Lo (101117765)

Report on Learning Algorithm Design

The task at hand involves image classification with a dataset represented by an n × d matrix, where n is the number of samples and d is the number of features representing a vectorized image. This report details the design and rationale behind the learning algorithm employed for this task.

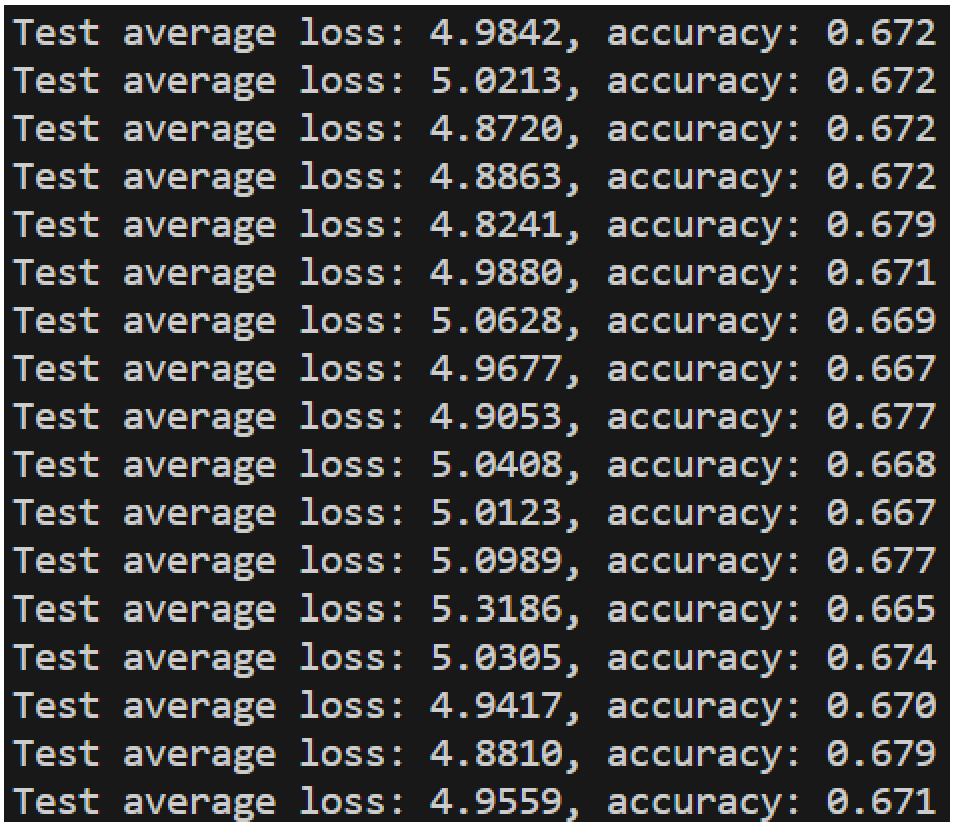
The input data, consisting of images, was preprocessed to ensure compatibility with the chosen neural network architecture. Each pixel value in the images was normalized to the range [0, 1] by dividing by 255. Additionally, the labels were encoded as integers between 0 and 9, inclusive.

The neural network model employed is a simple feedforward network with three fully connected layers. The choice of this architecture is motivated by its proven effectiveness in image classification tasks. The first layer takes in the flattened input image with 2352 features, followed by two hidden layers with 512 and 256 neurons, respectively. The final layer outputs a log-softmax activation for multi-class classification. The inclusion of dropout layers with a dropout probability of 0.25 after each fully connected layer is a regularization technique. This helps prevent overfitting during training and improves the model's generalization to unseen data.

Stochastic Gradient Descent (SGD) was chosen as the optimization algorithm. SGD is a widely used optimization algorithm suitable for large datasets, and its hyperparameter, the learning rate, was set to 0.1 based on initial experiments. Trying out different learning rates and ended up choosing 0.1 as the best option. When I lowered the learning rate, the accuracy went down a lot, setting it too high made it unstable so I kept it at 0.1. I tried some different optimization algorithms and most of the other ones did not perform as well as SGD, that was the reason why I stuck with this optimization algorithm. Adam was an example of another optimization algorithm that I ended up trying out and it gave painfully bad results.

The model was trained for 100 epochs on batches of 32 samples each. The choice of 100 epochs strikes a balance between allowing the model to converge and preventing overfitting. Training in batches helps optimize the computational efficiency of the training process. I let the model train for 100 epochs but could've set it to more for better accuracy in the long run after training for a long time. This was just to get a general idea of what the accuracy would hover around after 100 epochs.

The model's performance was evaluated on a separate validation dataset. The testing loop calculates the average loss and accuracy over the entire validation set. Overall the screenshot that I have included below shows the accuracy of the training model over time. The best accuracy that I was able to achieve was around 68%.



```
Test average loss: 4.9842, accuracy: 0.672
Test average loss: 5.0213, accuracy: 0.672
Test average loss: 4.8720, accuracy: 0.672
Test average loss: 4.8863, accuracy: 0.672
Test average loss: 4.8241, accuracy: 0.679
Test average loss: 4.9880, accuracy: 0.671
Test average loss: 5.0628, accuracy: 0.669
Test average loss: 4.9677, accuracy: 0.667
Test average loss: 4.9053, accuracy: 0.677
Test average loss: 5.0408, accuracy: 0.668
Test average loss: 5.0123, accuracy: 0.667
Test average loss: 5.0989, accuracy: 0.677
Test average loss: 5.3186, accuracy: 0.665
Test average loss: 5.0305, accuracy: 0.674
Test average loss: 4.9417, accuracy: 0.670
Test average loss: 4.8810, accuracy: 0.679
Test average loss: 4.9559, accuracy: 0.671
```

In conclusion, the chosen design for the learning algorithm is based on established practices in image classification tasks. The model architecture, preprocessing steps, and training strategy are all motivated by a desire for simplicity, efficiency, and generalization to unseen data.