MultiClass Classification Using Neural Networks

Lan H. Le
Department of Mechanical and Aerospace Engineering
University at Buffalo
Buffalo, NY 14226
hoanglan@buffalo.edu

Abstract

This project's purpose is to use different types of classifier to classify clothing images. The three types being used are one-hidden-layer Neural Network, multi-layer Neural Network, and Convolutional Neural Network (CNN).

1 Introduction

Classifying images has been one of the essential topics of machine learning. Compared to humans, it is considerably more difficult for machines to be able to distinguish objects in images.

The task of this project is classifying Fashion-MNIST clothing images into ten classes. The classes are:

1. T-shirt/top
2. Trouser
3. Pullover
4. Dress
5. Coat
6. Sandal
7. Shirt
8. Sneaker
9. Bag
10. Ankle Boot

2 Dataset

The Fashion-MNIST is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and test data sets have 785 columns. The first column consists of the class labels (see above), and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.

3 Preprocessing

The dataset is loaded using provided fashion mnist reader notebook. Since the input values are integers between 0 and 255, we need to scale the inputs by dividing them by 255. This becomes helpful when we start training the classifiers. In addition, the output labels also needs to be one-hot encoded for training purposes.

4 Architecture

1. For Neural Network with One Hidden Layer

In Neural Network, we pass input data through the layer(s) in a process called feedforward propagation and then use the output to update the weights in the layer(s). The update is called back propagation.

Since we only have one hidden layer, the total number of layers in this neural network is 3, the input layer, the hidden layer, and the output layer.

With the weights initialized before the first pass (and the updated weight in consequent passes), we use the input array of pixel values (x) to find z using the following equation:

$z = w^T . x + b$

Next, z becomes inputs for the activation function in the hidden layer. For this project, we choose Tanh function as the activation function for the hidden layer and Softmax function for the output layer. The output of the hidden layer (output of Tanh activation function) becomes inputs for the output layer. At this point, we repeat the process to find z which is the input for the Softmax function and then calculate the final values.

In order to find the "best" w and b, we update them for a number of iterations (epochs) using gradient descent.

$b := b - adb$

$w := w - adw$

(a is the learning rate)

So as to keep track of the progress of the training, loss function is used. If loss decreases after each epoch then the training is going in the right direction. Loss is recorded for both training and validation data.

Loss function:

$L(a2,y) = -E(ylog(a2))$

where E is the symbol for sum and a2 is the output values of the output layer.

2. For Multi-layer Neural Network

For this part and the CNN part, we use Keras, a high level Neural Network library to train our model. In this model, instead of using Tanh function as activation function for hidden layers, we use ReLU function. The Softmax function is still used as the function for the output layer.

$R(z) = max(0,z)$

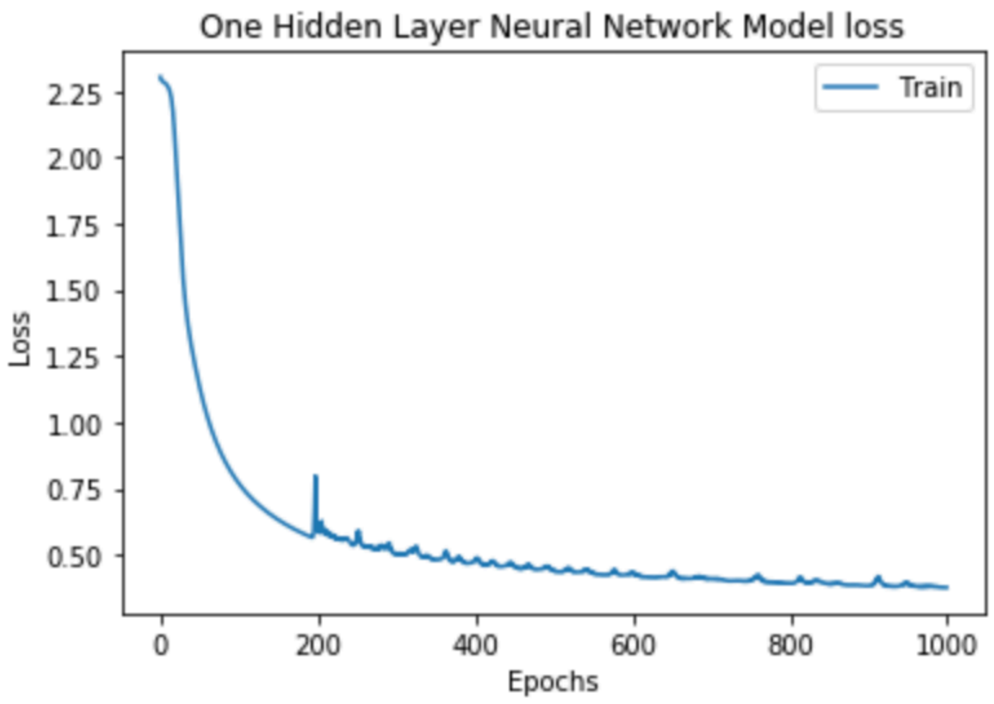
3. For Convolution Neural Network

We again use Keras to train the CNN model. We use 3 types of layers, Convolutional, Max Pooling, and Softmax. Instead of specifying number of nodes in each layer like in Multi-layer Neural Network, we now specify number of filters, filter size, and pool size.

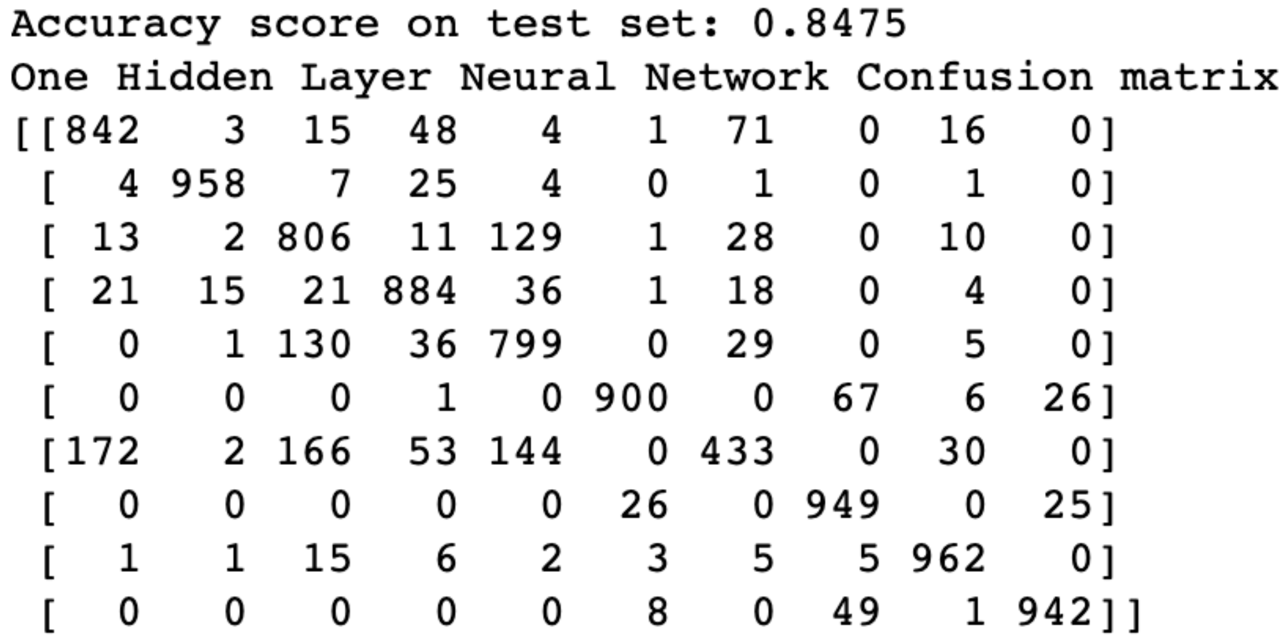
It is important that we tune the hyper-parameters for each of our model. For all three models, we need to monitor the number of epochs as well as the learning rate to improve the training accuracy and efficiency. In addition, for neural network with one or more layers, we also need to take into account the batch size, the number of nodes in each layer, and the number of layers.

5 Results

1. For Neural Network with One Hidden Layer

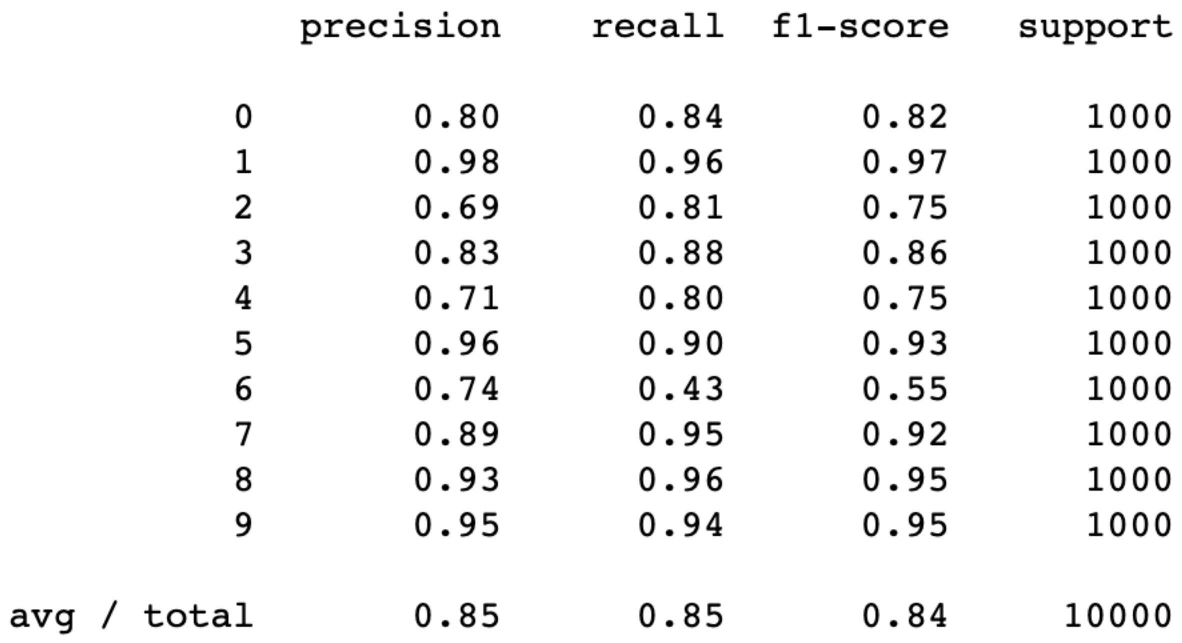


Loss vs Epochs graph



```
Accuracy score on test set: 0.8475
One Hidden Layer Neural Network Confusion matrix
[[842   3  15  48   4   1  71   0  16   0]
 [  4 958   7  25   4   0   1   0   1   0]
 [ 13   2 806  11 129   1  28   0  10   0]
 [ 21  15  21 884  36   1  18   0   4   0]
 [  0   1 130  36 799   0  29   0   5   0]
 [  0   0   0   1   0 900   0  67   6  26]
 [172   2 166  53 144   0 433   0  30   0]
 [  0   0   0   0   0  26   0 949   0  25]
 [  1   1  15   6   2   3   5   5 962   0]
 [  0   0   0   0   0   8   0  49   1 942]]
```

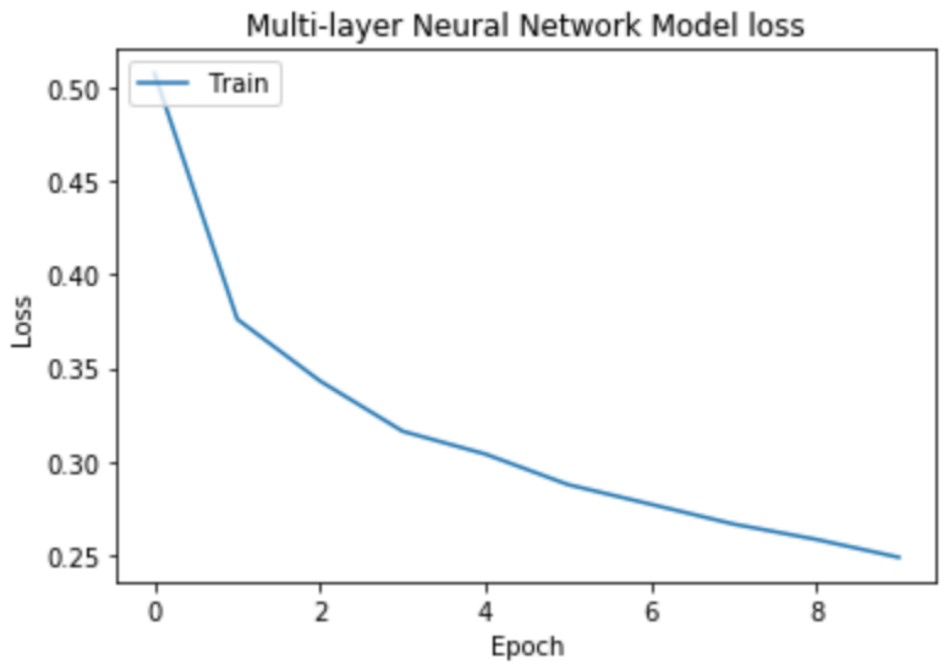
Confusion Matrix



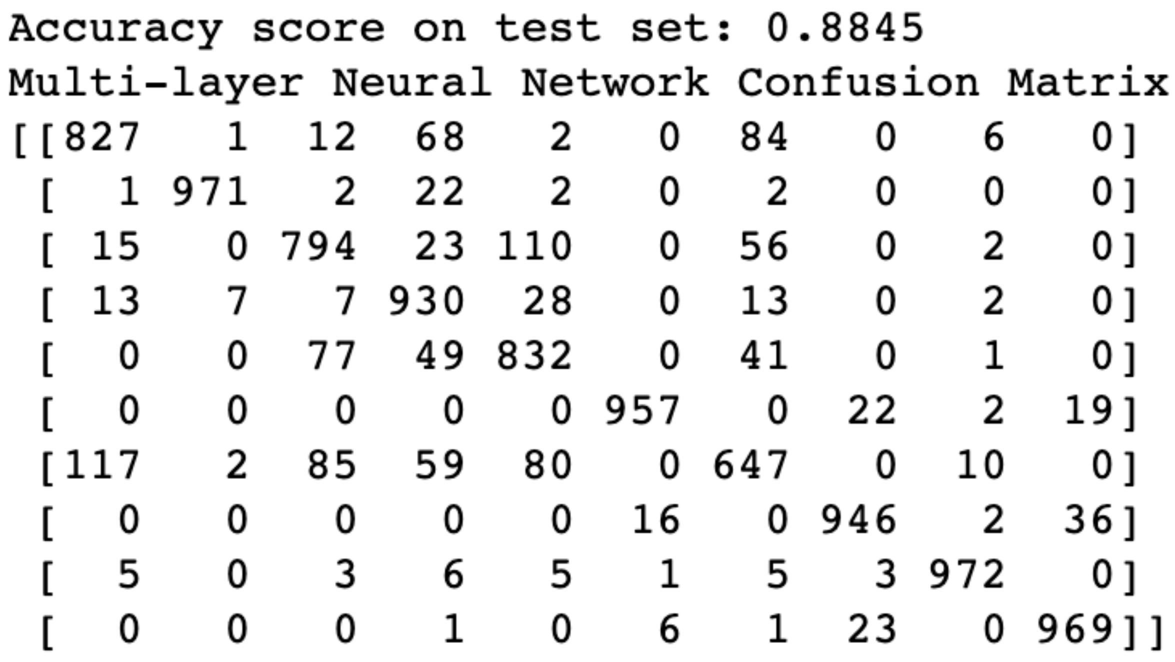
```
             precision    recall  f1-score   support

          0       0.80      0.84      0.82      1000
          1       0.98      0.96      0.97      1000
          2       0.69      0.81      0.75      1000
          3       0.83      0.88      0.86      1000
          4       0.71      0.80      0.75      1000
          5       0.96      0.90      0.93      1000
          6       0.74      0.43      0.55      1000
          7       0.89      0.95      0.92      1000
          8       0.93      0.96      0.95      1000
          9       0.95      0.94      0.95      1000

avg / total       0.85      0.85      0.84     10000
```

Classification Report

2. For Multi-layer Neural Network



Loss vs Epochs graph



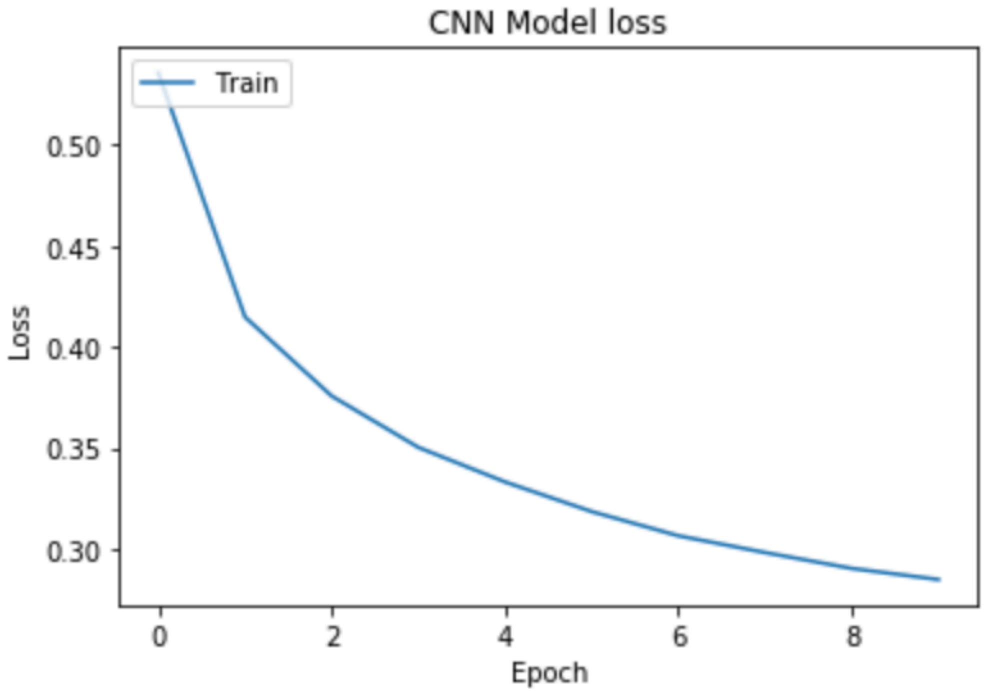
```
Accuracy score on test set: 0.8845
Multi-layer Neural Network Confusion Matrix
[[827   1  12  68   2   0  84   0   6   0]
 [  1 971   2  22   2   0   2   0   0   0]
 [ 15   0 794  23 110   0  56   0   2   0]
 [ 13   7   7 930  28   0  13   0   2   0]
 [  0   0  77  49 832   0  41   0   1   0]
 [  0   0   0   0   0 957   0  22   2  19]
 [117   2  85  59  80   0 647   0  10   0]
 [  0   0   0   0   0  16   0 946   2  36]
 [  5   0   3   6   5   1   5   3 972   0]
 [  0   0   0   1   0   6   1  23   0 969]]
```

Confusion Matrix

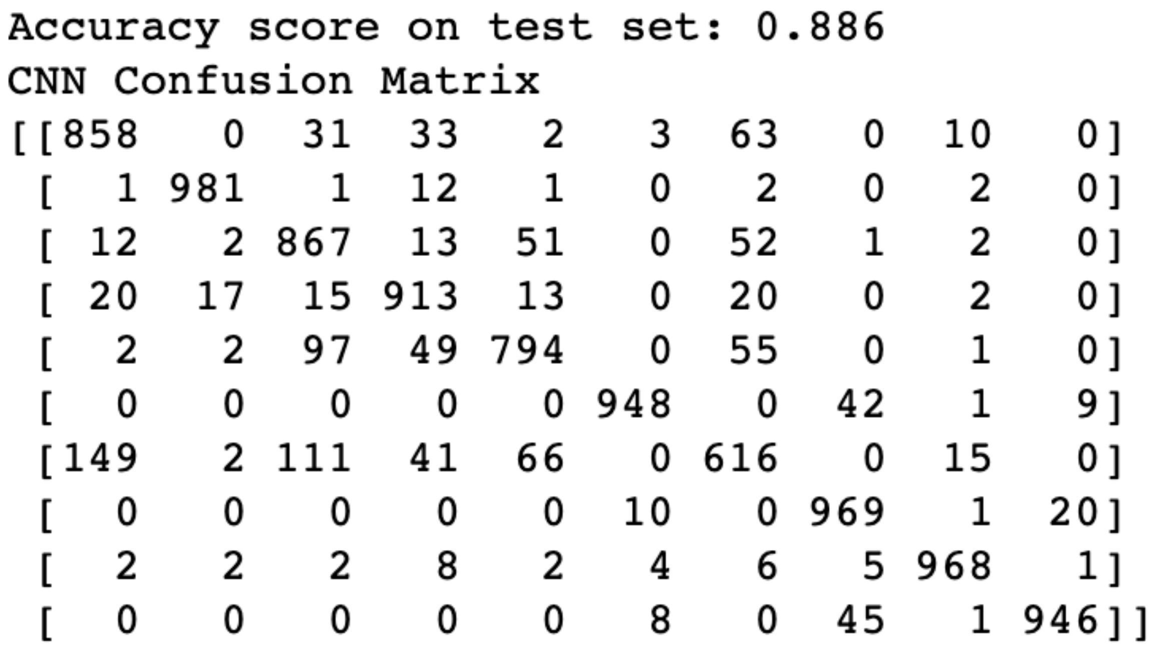
```
             precision    recall  f1-score   support

          0       0.85      0.83      0.84      1000
          1       0.99      0.97      0.98      1000
          2       0.81      0.79      0.80      1000
          3       0.80      0.93      0.86      1000
          4       0.79      0.83      0.81      1000
          5       0.98      0.96      0.97      1000
          6       0.76      0.65      0.70      1000
          7       0.95      0.95      0.95      1000
          8       0.97      0.97      0.97      1000
          9       0.95      0.97      0.96      1000

avg / total       0.88      0.88      0.88     10000
```

Classification Report

3. For Convolution Neural Network

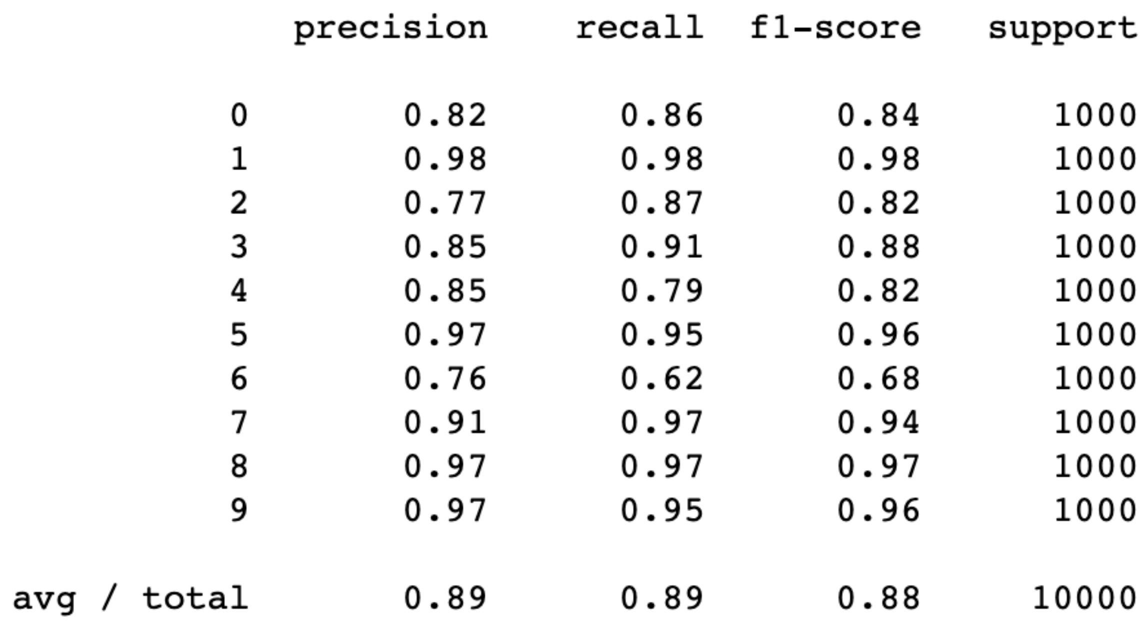


Loss vs Epochs graph



```
Accuracy score on test set: 0.886
CNN Confusion Matrix
[[858   0  31  33   2   3  63   0  10   0]
 [  1 981   1  12   1   0   2   0   2   0]
 [ 12   2 867  13  51   0  52   1   2   0]
 [ 20  17  15 913  13   0  20   0   2   0]
 [  2   2  97  49 794   0  55   0   1   0]
 [  0   0   0   0   0 948   0  42   1   9]
 [149   2 111  41  66   0 616   0  15   0]
 [  0   0   0   0   0  10   0 969   1  20]
 [  2   2   2   8   2   4   6   5 968   1]
 [  0   0   0   0   0   8   0  45   1 946]]
```

Confusion Matrix



```
             precision    recall  f1-score   support

          0       0.82      0.86      0.84      1000
          1       0.98      0.98      0.98      1000
          2       0.77      0.87      0.82      1000
          3       0.85      0.91      0.88      1000
          4       0.85      0.79      0.82      1000
          5       0.97      0.95      0.96      1000
          6       0.76      0.62      0.68      1000
          7       0.91      0.97      0.94      1000
          8       0.97      0.97      0.97      1000
          9       0.97      0.95      0.96      1000

avg / total       0.89      0.89      0.88     10000
```

Classification Report

6 Conclusion

We use 3 different classifiers to do one task: classifying clothing images into 10 classes of clothing. Though the implementation of each is slightly different, all 3 models prove to be very useful and produce considerably accurate results. Tuning the parameters is one of the most important process to ensure successful training. In the future, we would like to explore more about how other factors can influence and improve our models.

Acknowledgments

The description of the dataset is from the original project description. The equations for performance evaluation are also derived from class materials.

References