Machine Learning – Deep Neural Network – Report

Shlomi Ben-Shushan

Summary

In this report I'll describe the models of fully connected neural networks implemented via PyTorch, and for each model I'll explain its settings and their effects in terms of loss and accuracy. Afterwards. I will explain my implementation of a custom model that classifies examples with accuracy of 90%.

Hyper Parameters

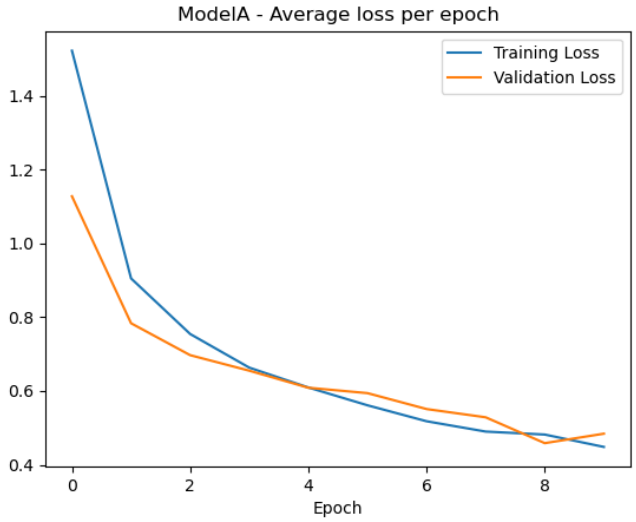
The only hyper-parameter in this exercise is the models' learning-rate that given by η. The optimized η for each model was found by running the NN for any η from 0.001 to 1 with increments of 0.001, and choosing the lowest η that gives the highest accuracy.

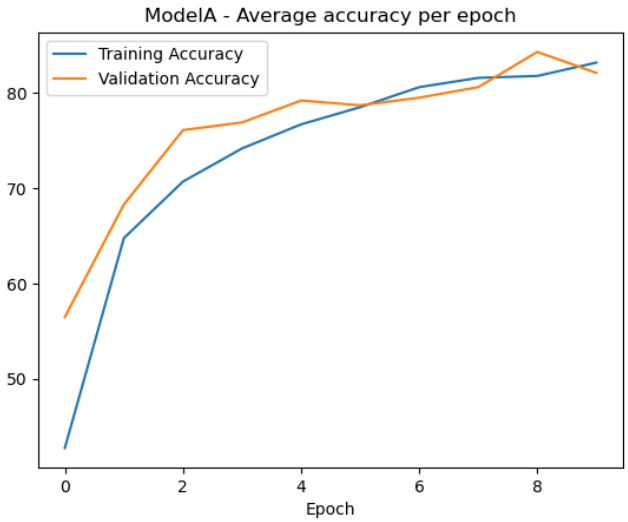
Model A

Description: A Neural Network with two hidden layers, the first layer is at size of 100 and the second layer is at size of 50, both layers followed by ReLU activation function. This model is trained by SGD optimizer.

Hyper-Parameter: learning-rate $\eta = 0.216$.

Effects:





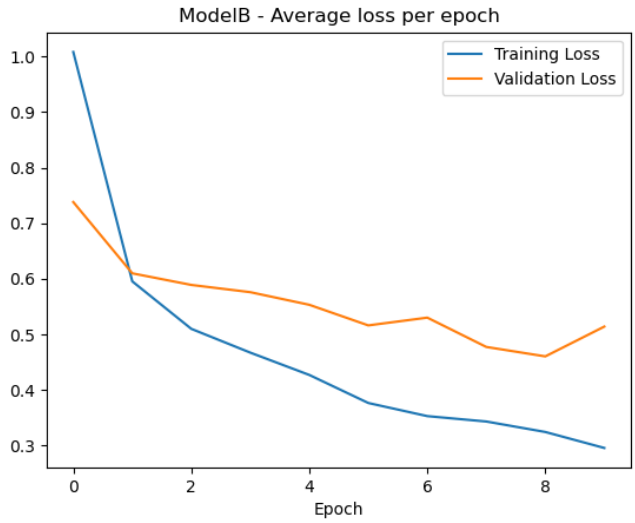
Achieved average validation accuracy of **76.22%**

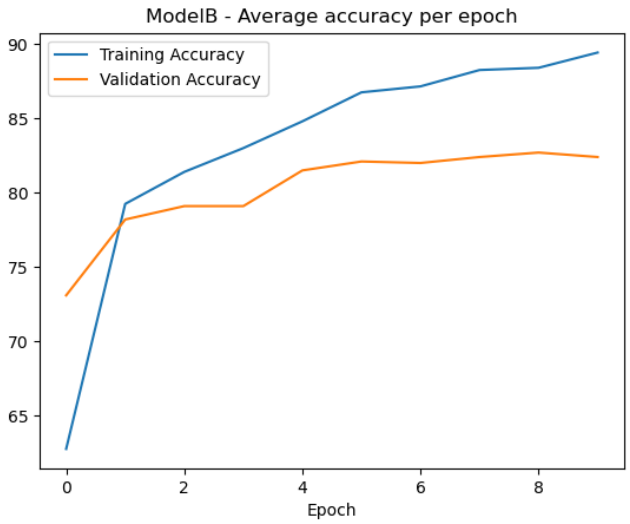
Model B

Description: A Neural Network with two hidden layers, the first layer is at size of 100 and the second layer is at size of 50, both layers followed by ReLU activation function. This model is trained by Adam optimizer.

Hyper-Parameter: learning-rate $\eta = 0.005$.

Effects:





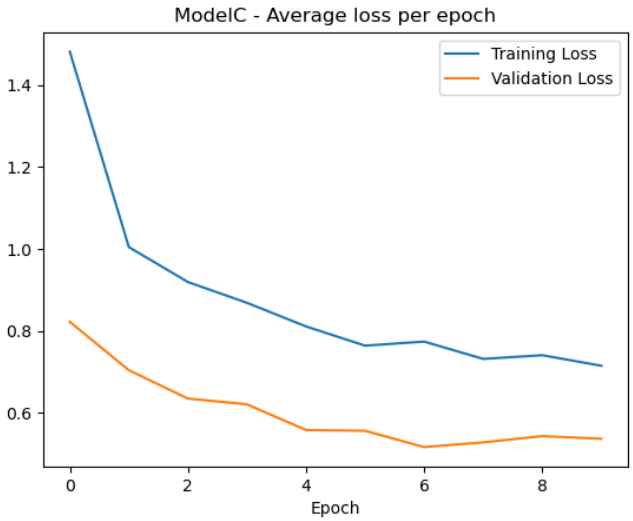
Achieved average validation accuracy of **80.26%**

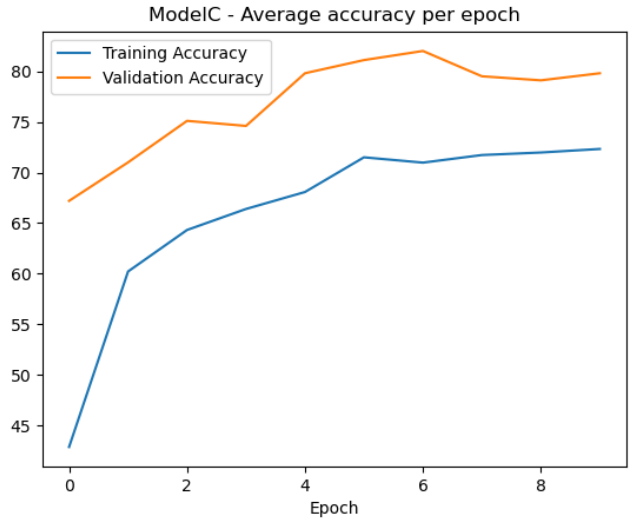
Model C

Description: A Neural Network with two hidden layers, the first layer is at size of 100 and the second layer is at size of 50, both layers followed by ReLU activation function, and dropout is activated on the output of each hidden layers. This model is trained by Adam optimizer.

Hyper-Parameter: learning-rate $\eta = 0.005$.

Effects:





Achieved average validation accuracy of **76.92%**

Model D

Description: A Neural Network with two hidden layers, the first layer is at size of 100 and the second layer is at size of 50, both layers followed by ReLU activation function. In this mode, Batch Normalization is activated before each activation function. This model is trained by Adam optimizer.

Normalization Placing: the decision for placing the normalization layer before the each activation function was made after testing placing it before the activation functions and after them, and the results were as follows:

- Placing before:
  - Code:

```
out = F.relu(self.bn0(self.fc0(out)))
out = F.relu(self.bn1(self.fc1(out)))
```

  - Average training accuracy in the last epoch: **88.72%**
  - Average validation accuracy in the last epoch: **82.61%**
- Placing after:
  - Code:

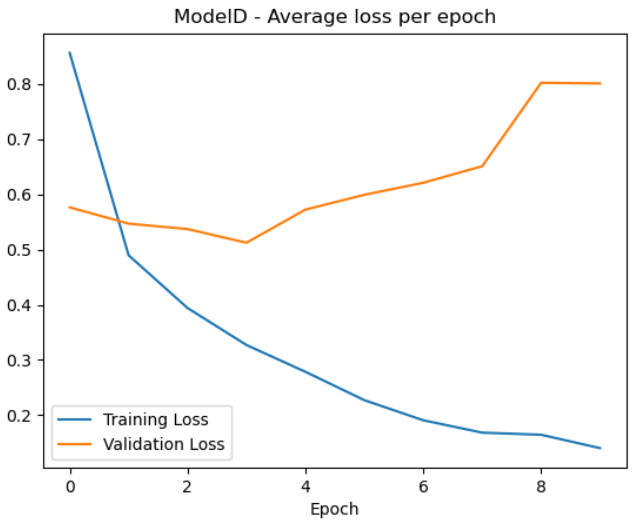
```
out = self.bn0(F.relu(self.fc0(out)))
out = self.bn1(F.relu(self.fc1(out)))
```

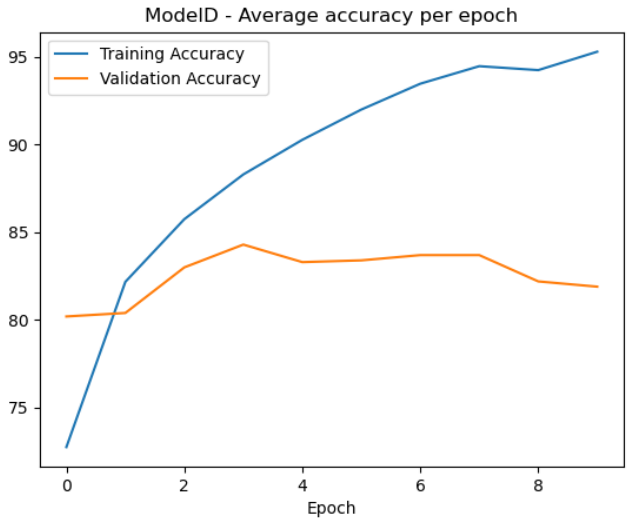
  - Average training accuracy in the last epoch: **85.04%**
  - Average validation accuracy in the last epoch: **76.48%**

Clearly the first option is better.

Hyper-Parameter: learning-rate $\eta = 0.005$.

Effects:





Achieved average validation accuracy of **82.61%**

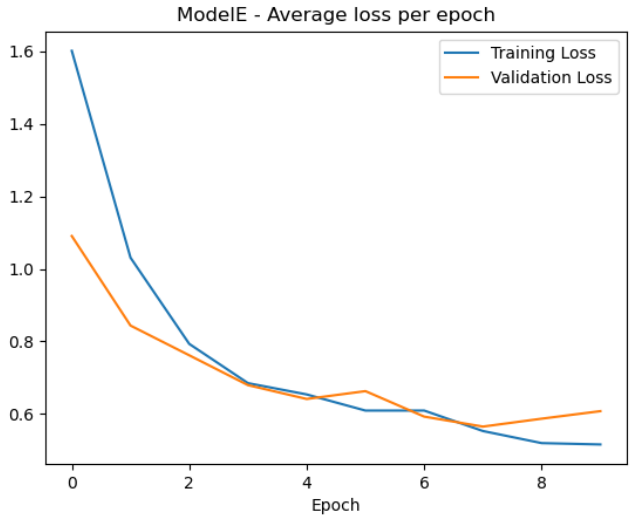
Model E

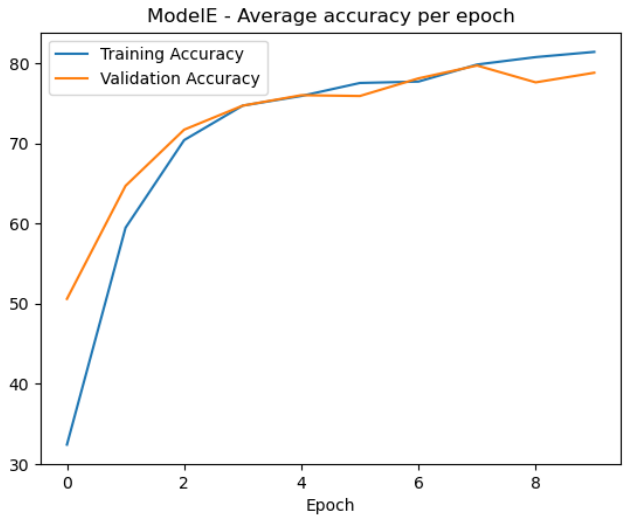
Description: A Neural Network with five hidden layers at sizes 128, 64, 10, 10, 10 respectively using ReLU activation function. This model is trained by Adam optimizer.

Note: Four different optimizers have been tested – SGD, Adam, AdaDelta and RMSprop. The best performance were achieved with Adam optimizer.

Hyper-Parameter: learning-rate $\eta = 0.013$.

Effects:





Achieved average validation accuracy of **72.78%**

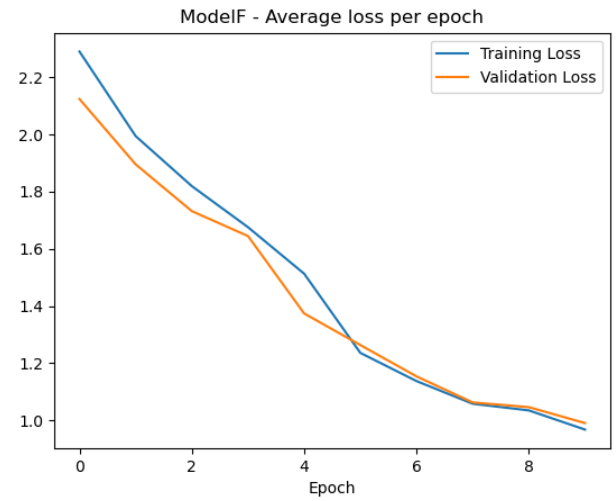
Model F

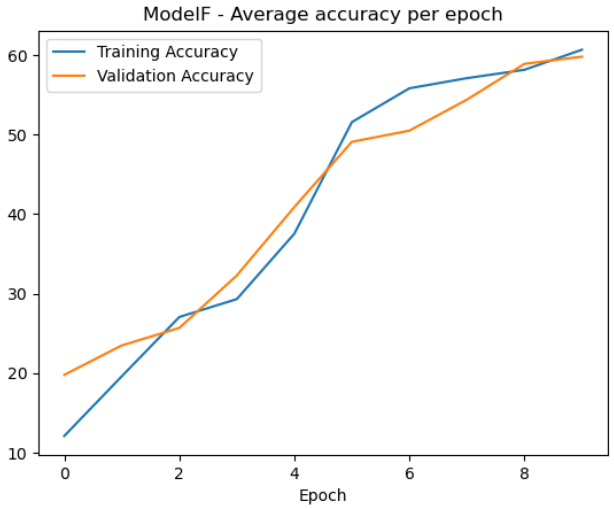
Description: A Neural Network with five hidden layers at sizes 128, 64, 10, 10, 10 respectively using Sigmoid activation function. This model is trained by Adam optimizer.

Note: Four different optimizers have been tested – SGD, Adam, AdaDelta and RMSprop. The best performance were achieved with Adam optimizer.

Hyper-Parameter: learning-rate $\eta = 0.023$.

Effects:





Achieved average validation accuracy of **41.49%**

Model S

Description: This is a custom NN I have implemented to achieve accuracy over 90%. it consists of two hidden layers, the first is half the size of the input, and the second is quarter the size of the input. Each layer is followed by ReLU activation function and Batch Normalization, and drop-out is activated after it. In the end, the NN returns output through log_softmax function. This model is trained by Adam optimizer as long as the validation accuracy does not converge, and by SGD as long as it converges.

Foraward Method:

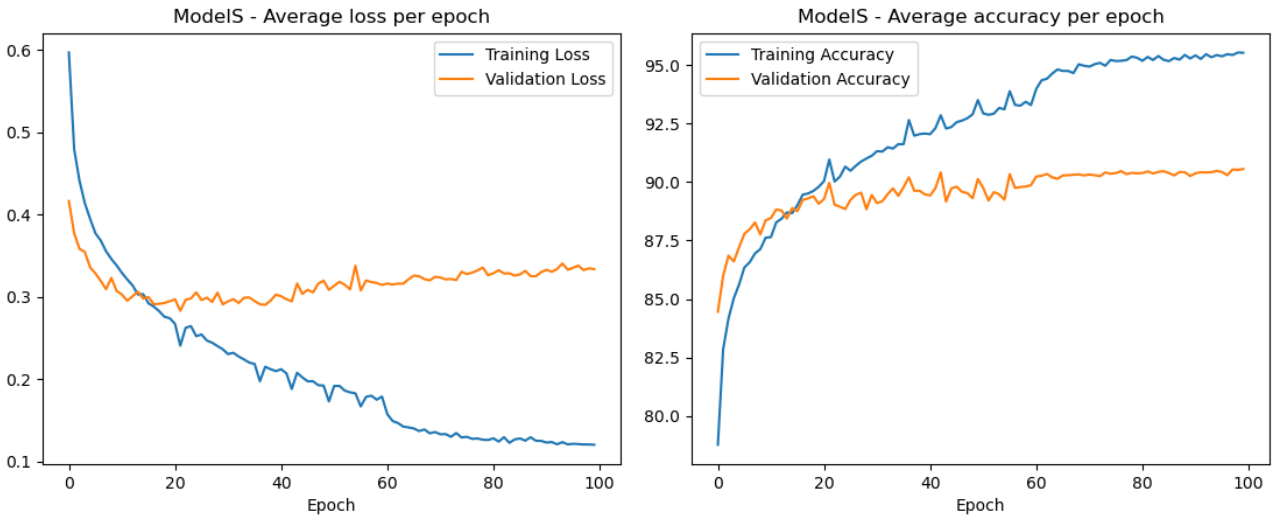
```
out = x.view(-1, self.image_size)
out = F.relu(self.bn0(self.fc0(out)))
out = F.dropout(out, training=self.training)
out = F.relu(self.bn1(self.fc1(out)))
out = F.dropout(out, training=self.training)
out = F.log_softmax(self.fc2(out), dim=1)
return out
```

Hyper-Parameters: learning-rates $\eta_{Adam} = 0.005,\ \eta_{SGD} = 0.2, n_{epochs} = 100$.

Notes:

- Although Model D yielded very good performance (best among models A to F), I decided to implement my own model, that uses the advantages of previous models, to ensure accuracy over 90%.
- This model provided an accuracy greater than 90% when trained for **100 epochs**.

Effects:



The diagram shows the switching moments between the Adam and SGD optimizers until the validation accuracy is fully converged around the 90%.

Achieved average validation accuracy of **89.57%**, and the last accuracies were 95.52% on the training set and **90.06%** on the validation set.