Laboratory 5
Variant 3
Group 3

By Krzysztof Kotowski and Juan Manuel Aristizabal Henao

Introduction

- Aim of the exercise is classifying 28x28 pixel images of clothes from Zalando using multi-layered perceptron
    - This is a neural network with multiple layers of neurons (1 input layer, N hidden layers and 1 output layer), with an image input and output choosing the clothing type
- Training of neural network is done using mini-batch gradient descent method
    - This method takes relatively small batches of training data and updates the neural network's neuron parameters based on the results of those small batches
- Our task will make multiple "versions" of the perceptron by trying out multiple parameter combinations. These parameters are:
    - Learning rate (gradient descent's step size)
    - Mini-batch size (number of test samples used in each iteration of training)
    - Number of hidden layers (number of neuron layers between input and output layers)
    - Width (number of neurons in each hidden layer)
    - Activation function (function determining the activation of neurons, based on their input values)

Implementation

The solution to this task was implemented using Pytorch and ran mostly on Google Colab. The basic neural network implementation was based mostly on the LeNet convolutional network.

The following are the static parameters for all the experiments:

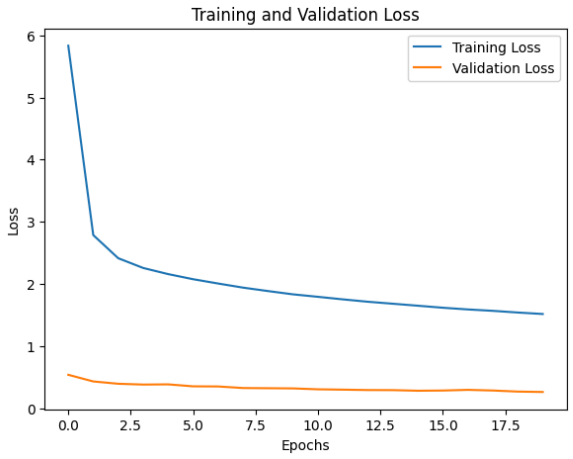
- Seed: 2
- Epochs: 20
- Criterion: Cross Entropy
- Optimizer: Stochastic Gradient Descent

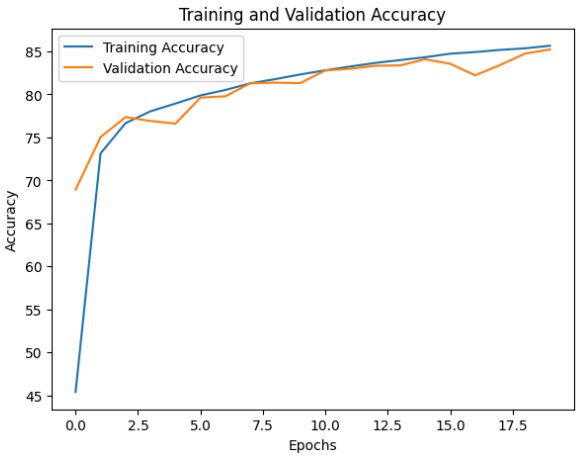
Experiments

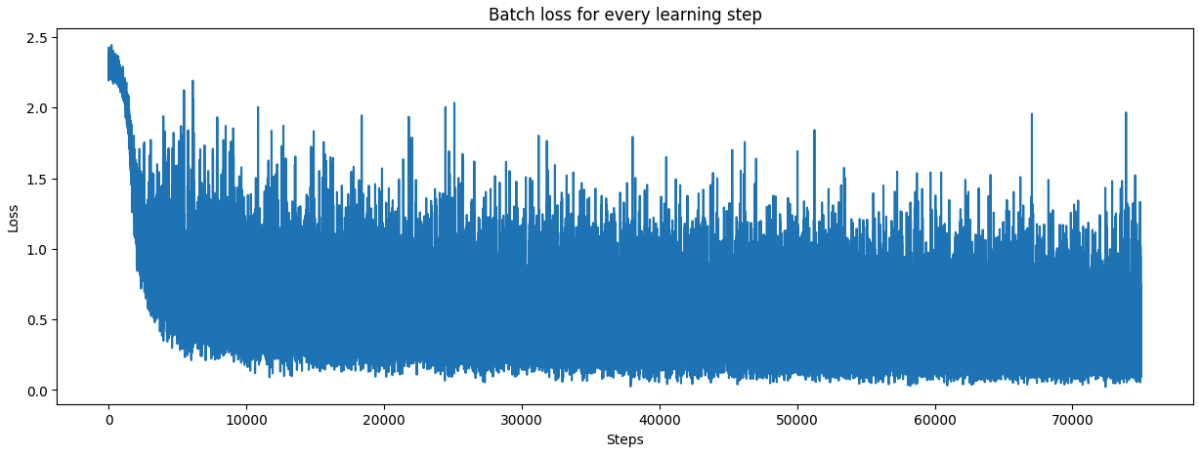
Experiment 1 (base parameters):

This experiment shows the basic behaviour of the neural network implementation. The parameters mentioned here will be used in the subsequent experiments unless their change is specified.
Parameters:

- Learning rate: 0.001
- Mini-batch size: 16
- Number of hidden layers: 7
- Width: 28
- Activation function: ReLU





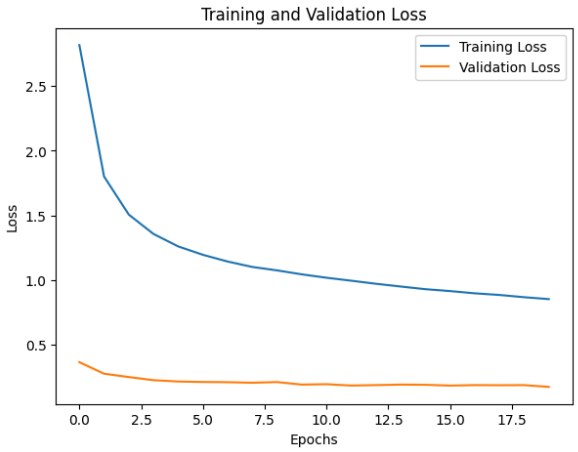


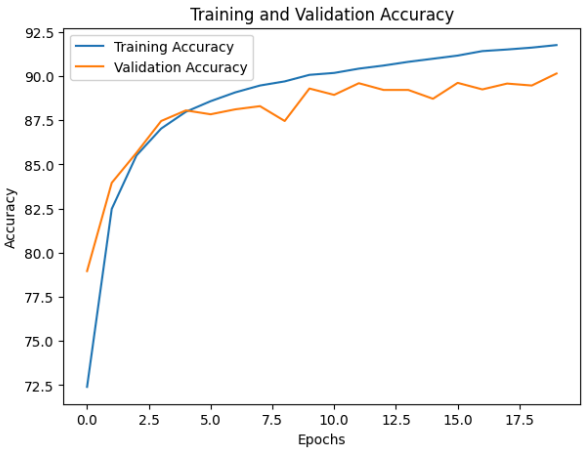
From the above pictures, it can be seen that the training and validation loss shows that the model is trending not to Underfit or Overfit the data, but also that more Epochs might be needed to achieve more accurate predictions. Regarding the batch loss, it can be seen that at the first 2 epochs, the reduction of loss of each batch after every learning step is more significant than in the following ones; at the following epochs, the losses also decrease, but at a much lower rate.

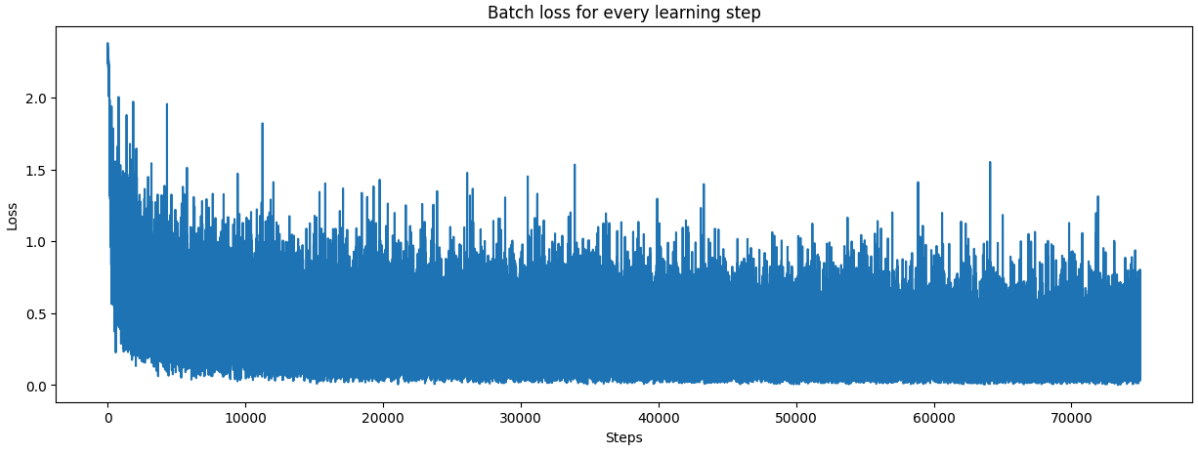
Experiment 2 (change of Learning rate):

Parameters:

- Learning rate: 0.01
- Mini-batch size: 16
- Number of hidden layers: 7
- Width: 28
- Activation function: ReLU





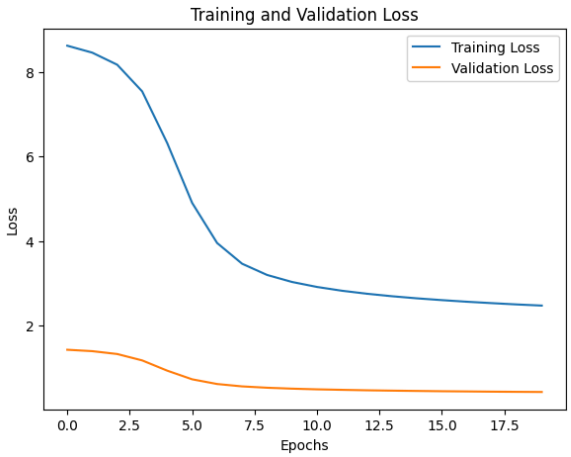


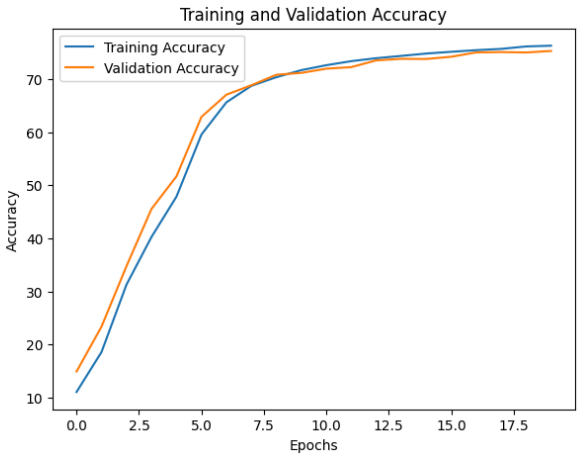
As shown in the above graphs, increasing the learning rate to 0.01 increases the rate at which the model learns; hence, the losses after each epoch decrease rapidly for the first 4 epochs. This steep decrease in the losses can also be seen in the losses of the batches after each learning step, in which a low loss is reached faster than in Experiment 1. In the case of accuracy, a higher level is reached at the end of the epochs than in Experiment 1.

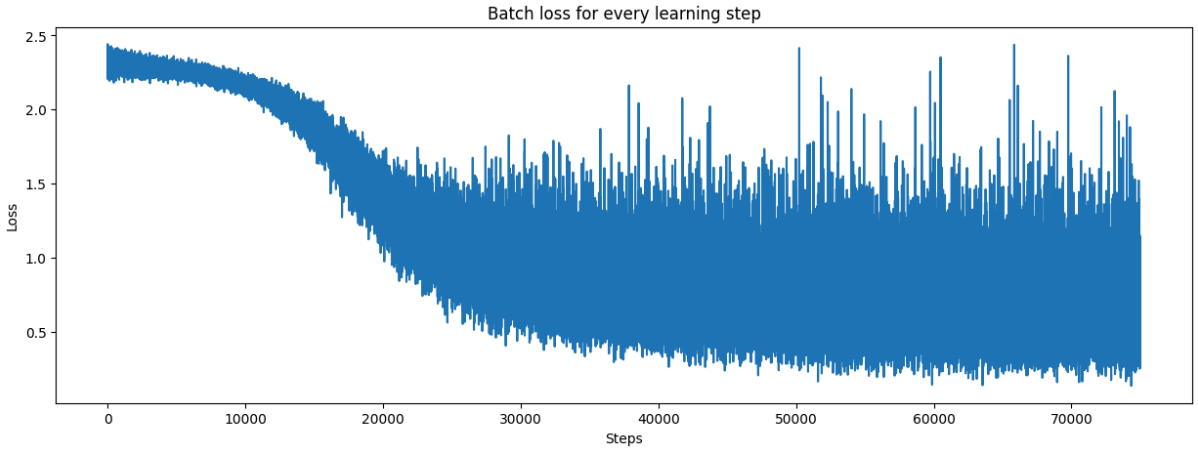
Experiment 3 (change of Learning rate):

Parameters:

- Learning rate: 0.0001
- Mini-batch size: 16
- Number of hidden layers: 7
- Width: 28
- Activation function: ReLU





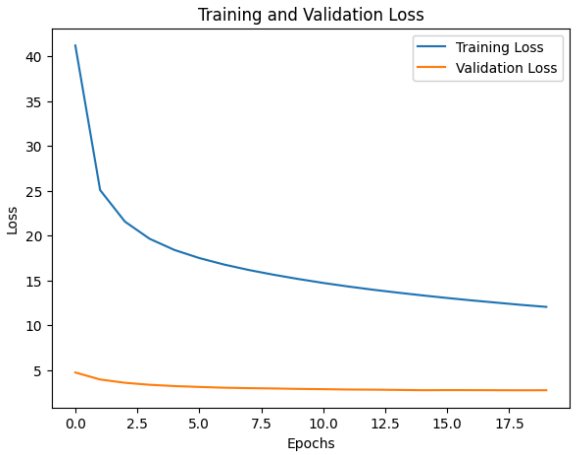


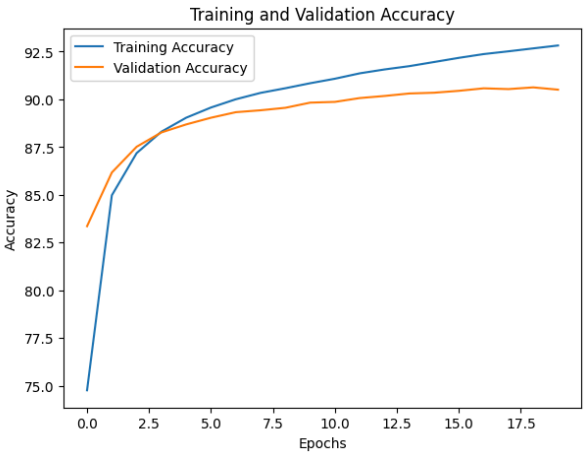
For this experiment, the decreased learning rate shows a proportional reduction in the rate at which the losses decrease, almost reaching the initial training loss for the two previous experiments on epoch 20. This reduction also shows a decrease in the overall accuracy at each epoch of training and validation. On the other hand, training and validation accuracy were quite similar at each epoch, which shows an overall high accuracy tendency that given more epochs, would have been able to train a model that may generalize better than the previous two.

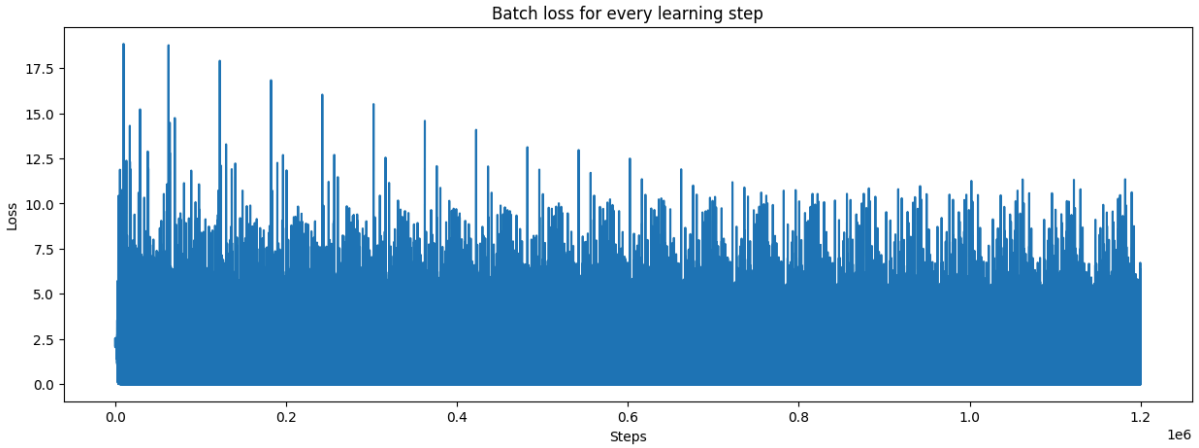
Experiment 4 (change of Mini-batch size):

Parameters:

- Learning rate: 0.001
- Mini-batch size: 1
- Number of hidden layers: 7
- Width: 28
- Activation function: ReLU





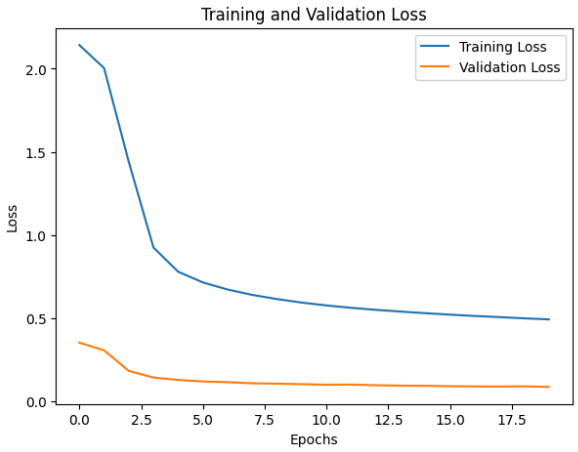


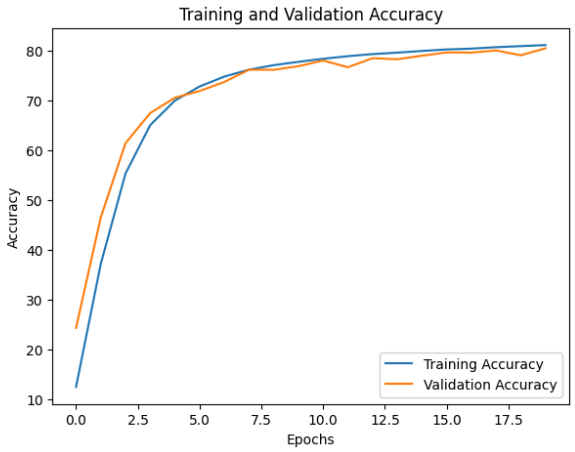
As seen in this experiment, decreasing the batch size to 1 element highly increases the losses for the batches, training, and validation. In the case of the batch losses, the losses show either a correct prediction (loss equal to 0) of the loss of a specific image to its prediction. It is worth noticing how, in the batch loss graph, the maximum loss of a batch is decreasing after each step, and that the number of steps increased significantly, which in turn made the training process much slower.

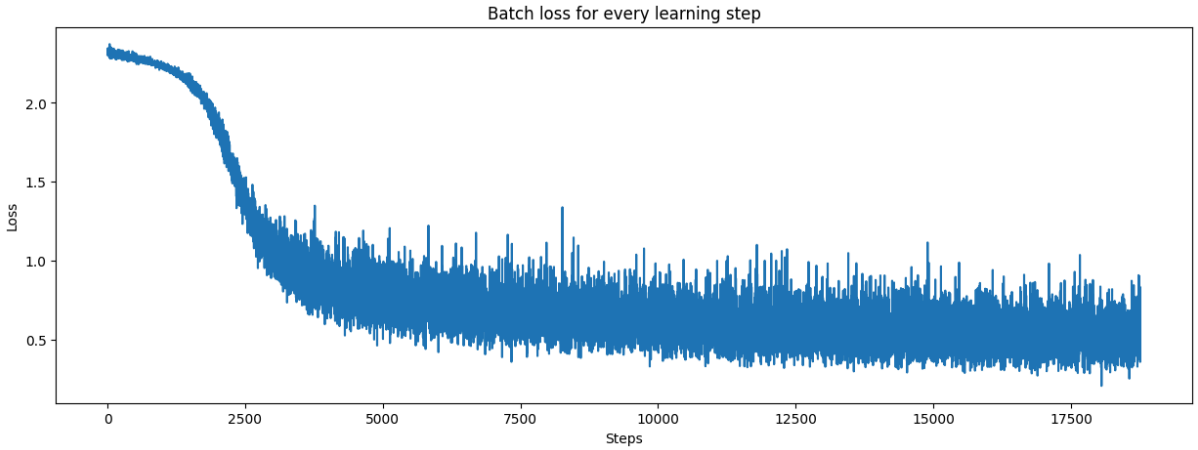
Experiment 5 (change of Mini-batch size):

Parameters:

- Learning rate: 0.001
- Mini-batch size: 64
- Number of hidden layers: 7
- Width: 28
- Activation function: ReLU





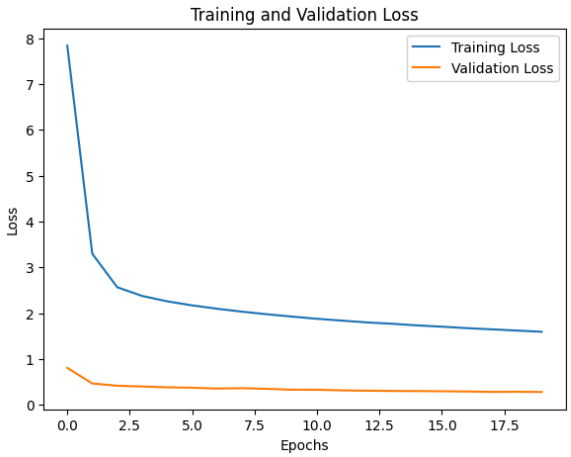


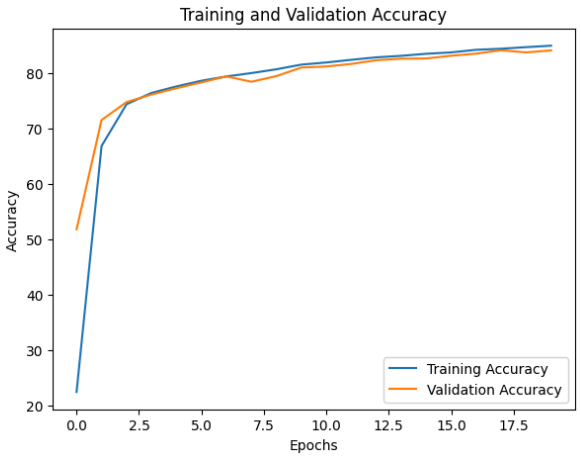
In this experiment, the increase of the batch size caused a much faster reduction of the losses on training, validation, and batches. In addition, the number of steps done by the overall training reduced significantly, speeding up the training loop. On the other hand, the accuracy of the model was lower than in most of the previous experiments, including Experiment 1.

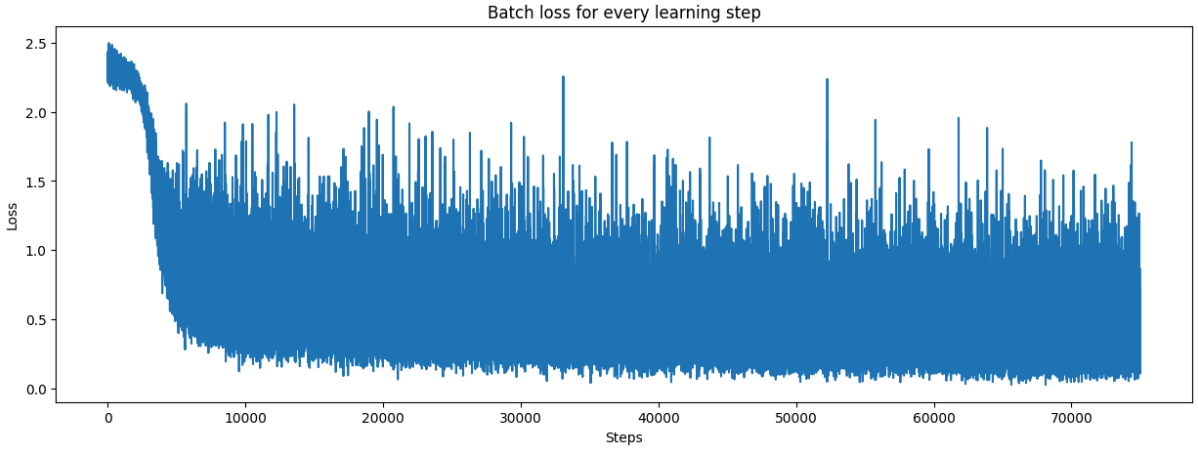
Experiment 6 (change of Width):

Parameters:

- Learning rate: 0.001
- Mini-batch size: 16
- Number of hidden layers: 7
- Width: 14
- Activation function: ReLU





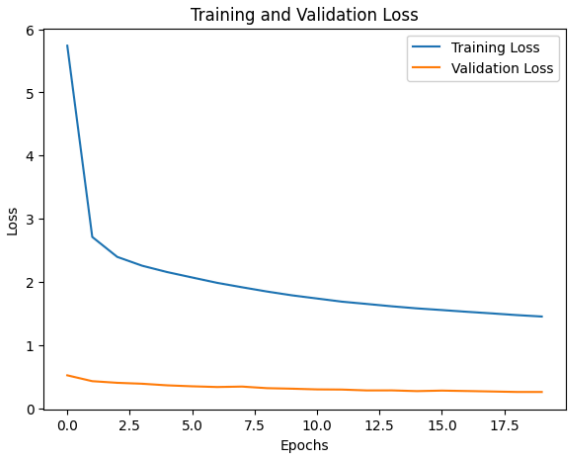


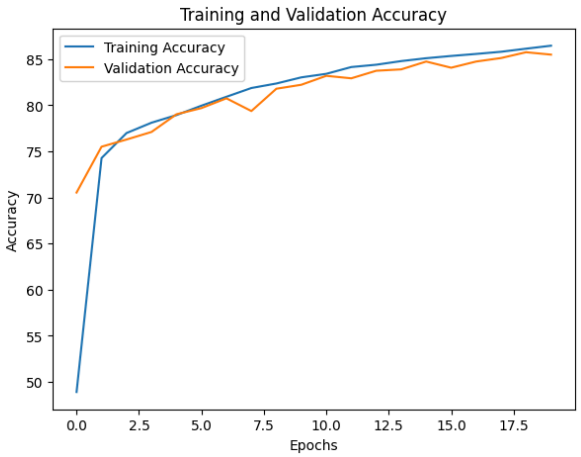
By decreasing the width of the model, the initial losses are higher with respect to the Experiment 1, but quickly decrease to the values of losses for Experiment 1. In terms of the accuracy, it starts much lower but quickly increases to values similar to the Experiment 1, reaching a very similar value at the last epoch. This behaviour shows that by decreasing the width, the initial losses increase, but over the training, it will quickly reach acceptable values of accuracy and losses.

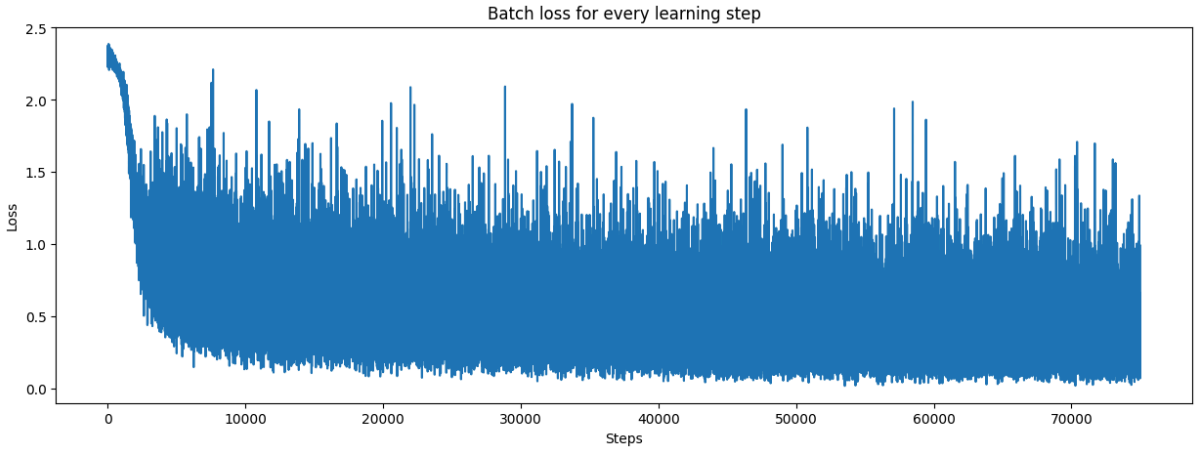
Experiment 7 (change of Width):

Parameters:

- Learning rate: 0.001
- Mini-batch size: 16
- Number of hidden layers: 7
- Width: 56
- Activation function: ReLU





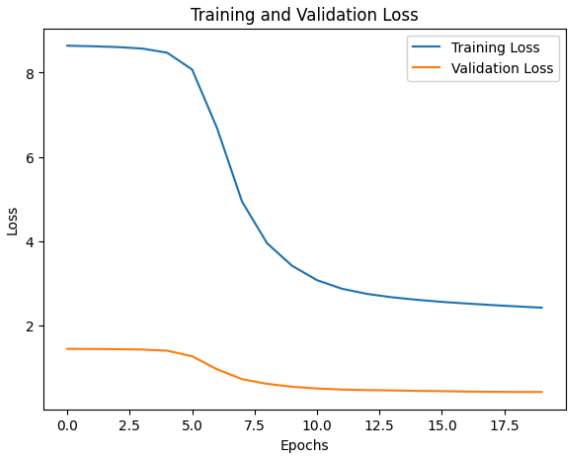


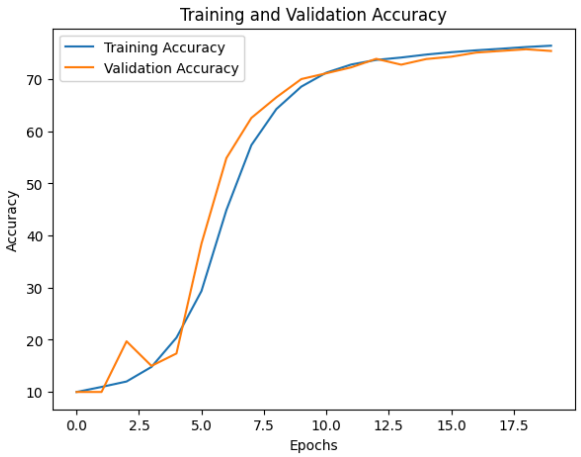
In this experiment, the increase in the width did not significantly change the training of the model with respect to the results of Experiment 1. The losses for training, validation, and batch remained in very similar values. In the case of the accuracy, it may have initially not been as good as in Experiment 1, but it quickly increased and reached not only its level but also a very similar trend to Experiment 1. Hence, it can be concluded that having a wider network does not guarantee that the accuracy will increase and the loss will decrease.

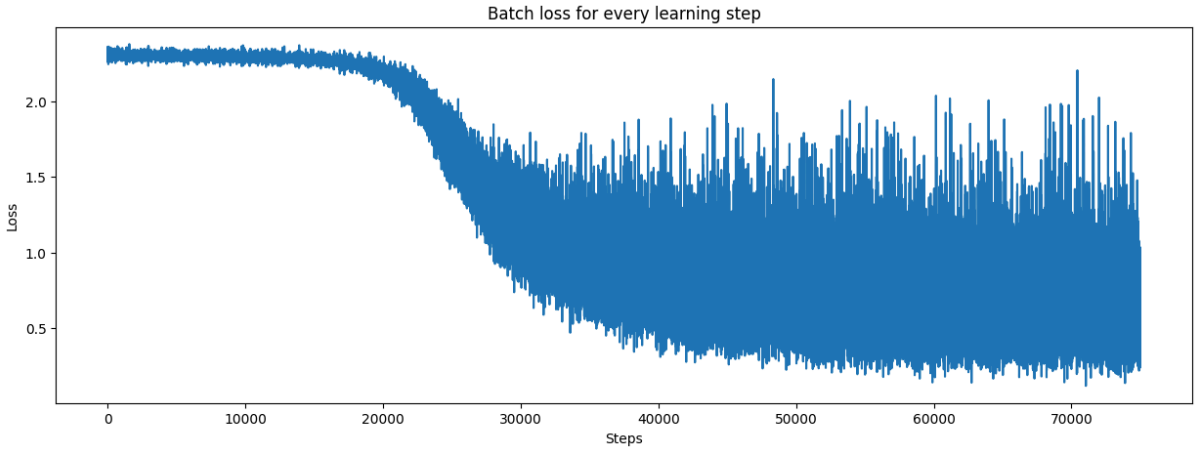
Experiment 8 (change activation function):

Parameters:

- Learning rate: 0.001
- Mini-batch size: 16
- Number of hidden layers: 7
- Width: 28
- Activation function: Sigmoid





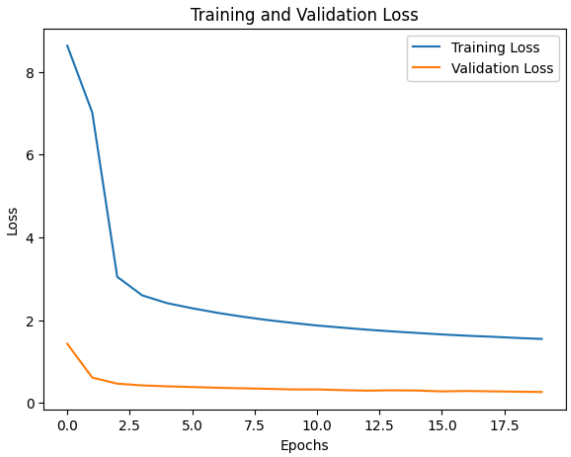


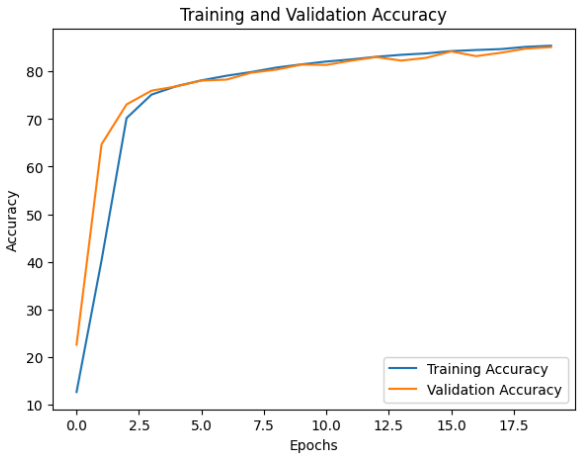
From the above graphs can be seen that the usage of the Sigmoid function as an activation function for this kind of network yields suboptimal performance on the training. This is because the loss on the initial steps and epochs changes more slowly than in experiments with an activation function as ReLU. This is seen on the first 5 epochs, where the losses do not decrease much, even though they are high. In the last epochs, it started to reach values closer to the previous experiments, but still below them. Hence, it can be concluded that the usage of the Sigmoid function for this type of network makes the training go slower than when using the ReLU function.

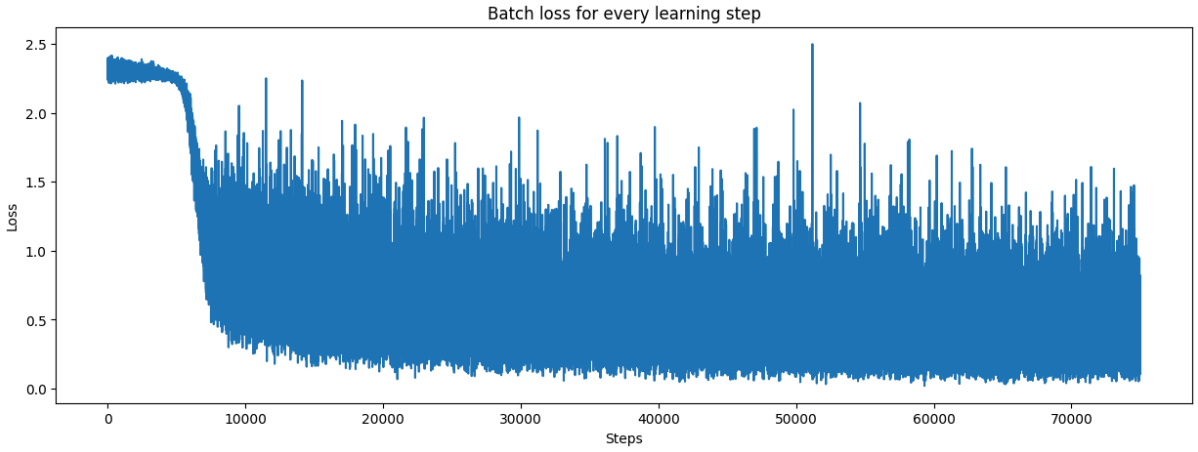
Experiment 9 (change activation function):

Parameters:

- Learning rate: 0.001
- Mini-batch size: 16
- Number of hidden layers: 7
- Width: 28
- Activation function: GELU





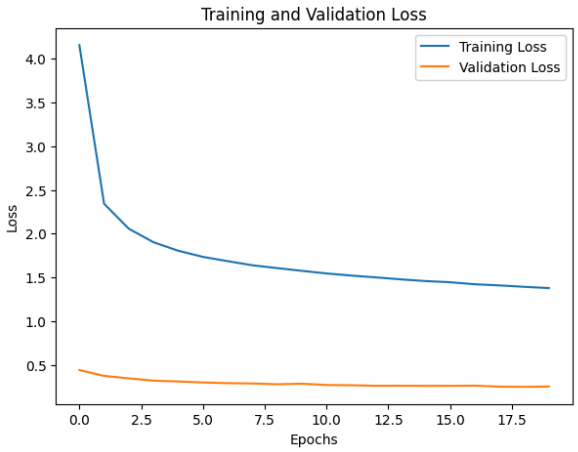


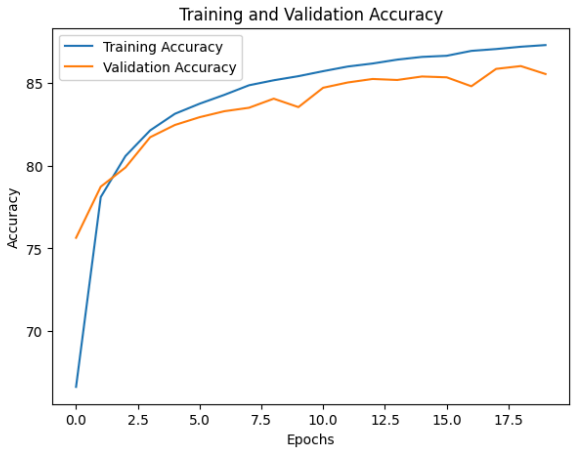
By using GELU as an activation function in this experiment, the initial losses were similar to the Sigmoid function, but contrary to the experiment with Sigmoid function, the losses quickly decreased in the first 3 epochs and reached, at the end, very similar values to those in Experiment 1. Something similar happened on the accuracy, where starting in much lower values, it was quickly to reach accuracy values on par with Experiment 1. Hence, this activation function can work as a middle point between Sigmoid and ReLU, because it can make the model accuracy start from low values, as when using Sigmoid, and it can reach high values quickly, as when using ReLU.

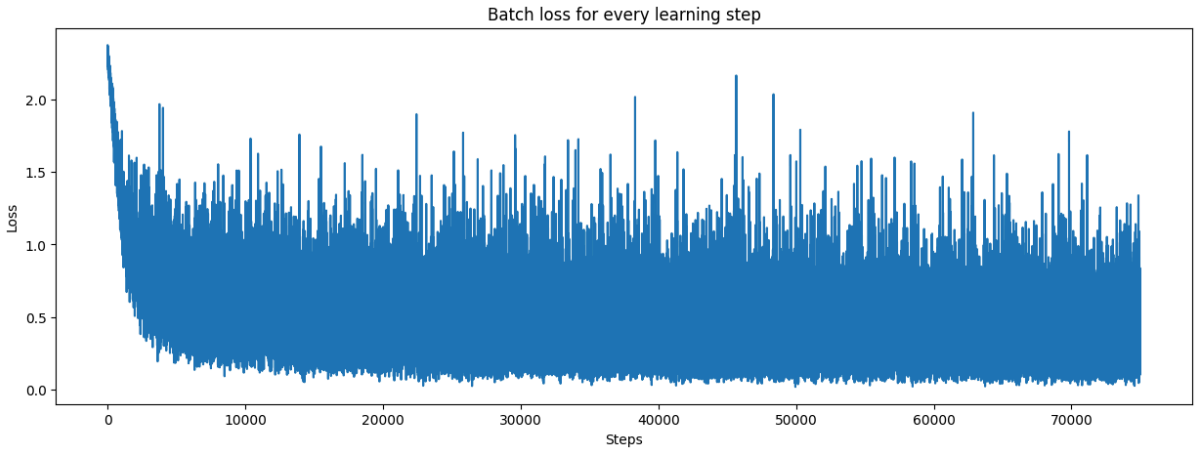
Experiment 10 (change of number of hidden layers):

For this network, a convolutional layer was removed from the original.
Parameters:

- Learning rate: 0.001
- Mini-batch size: 16
- Number of hidden layers: 5
- Width: 28
- Activation function: ReLU





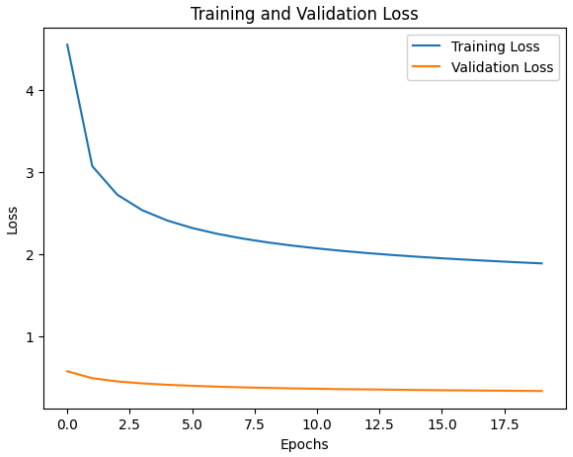


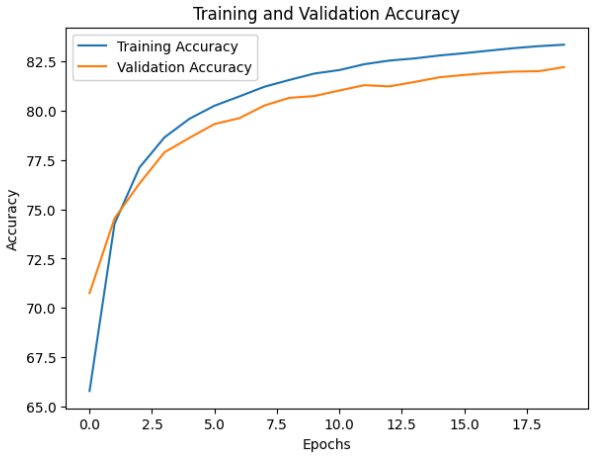
By removing one layer, the loss decreased, and the value obtained after all the epochs was lower than in Experiment 1. The accuracy of the model increased a bit more than in Experiment 1. In conclusion, by removing this convolutional layer, the overall training performance of the model increased.

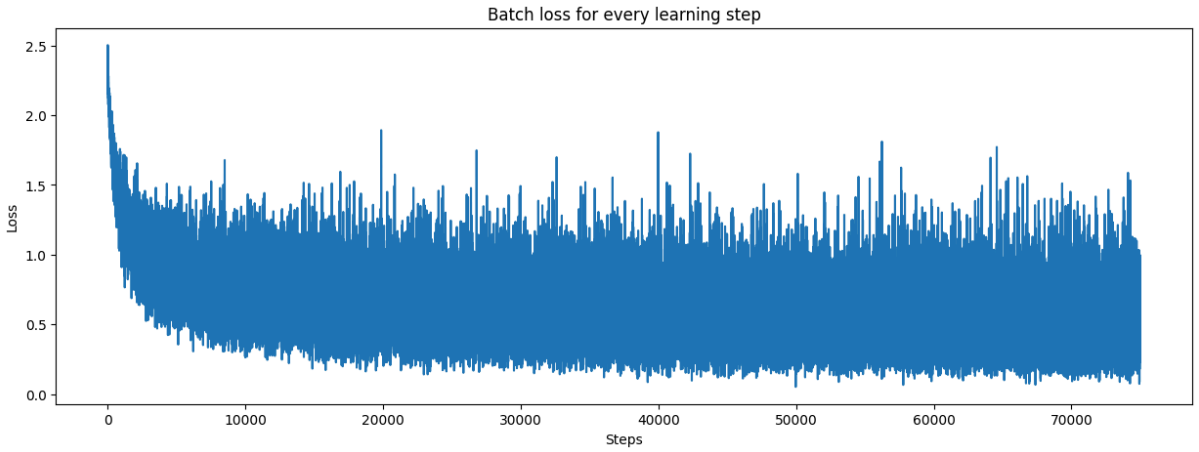
Experiment 11 (change of number of hidden layers):

For this experiment, a linear model was implemented.
Parameters:

- Learning rate: 0.001
- Mini-batch size: 16
- Number of hidden layers: 0
- Width: 0
- Activation function: None







As seen in the above graphs, training a linear model to fit the images, gave very similar results as the previous 2 network implementations. In this case, the losses were at the beginning a bit higher than in Experiment 10, but lower than on Experiment 1. At the end of the epochs the model had very similar losses and accuracy to the one fitted in Experiment 1. Hence, it can be concluded that the training of the network used in Experiment 1 yields very similar results to the just a linear model.