**Question 1** (Building the Mode)**.**

**Question 2** (Initialization)**.** The model architecture for this question maintain the same mentioned in solution.py file:

- hidden layers =(784, 256)
- epsilon=1e-6
- learning rate=0.01
- batch size=64
- activation function = relu

In fact, here we consider different initialization schemes for the weight parameters: Normal and Glorot.
The results of training the model on CIFAR 10 for 20 epochs using are shown on the plots below :
The left plot record the loss measured on the training data at the end of each epoch usin normal distribution as a method of weights initialization and the right plots using Glorot method .

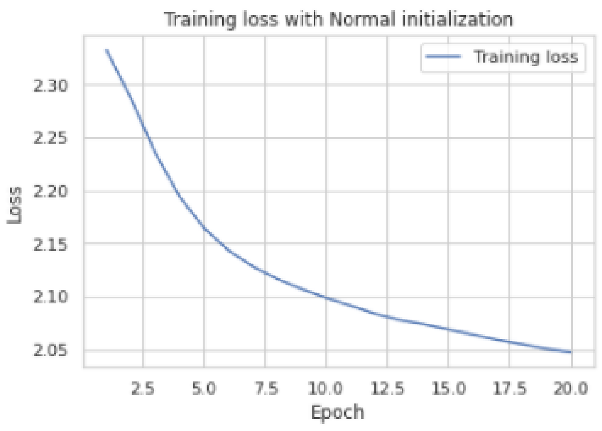


Figure 1: Training loss with Normal initialization

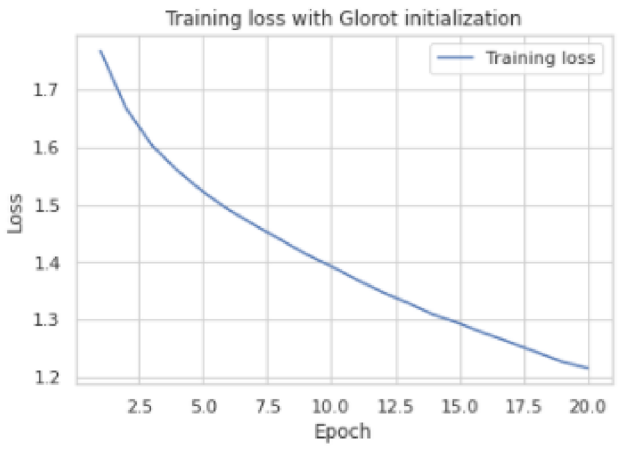


Figure 2: Training loss with Glorot initialization

Should be noticed that the plots have different scales and it is clear that we get different results for the two methods. To visualize the difeerence better we can put them in a single plots as shown below:

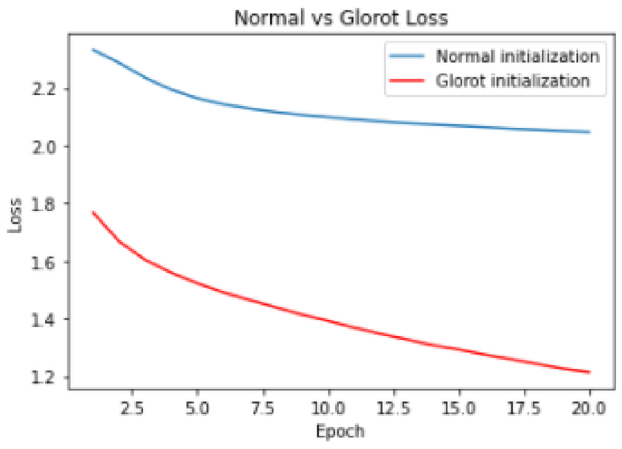


Figure 3: Loss Comparison

We notice that the loss, using Normal method (the blue one) starts withe a high value ($\geq$2.30), it does improve over time, but the rate of convergence is very low compared to other and at the end of the 20 epoch, the loss is close to 2.05.
In the other hand, the loss , using Glorot method (the red one) starts withe a less value ($\approx 1.8$) and gets 1.2 at the end of the training.
The MLP achieved better results using Glorot weight initialization technique, less loss and better accuracy(shown in the plot below).

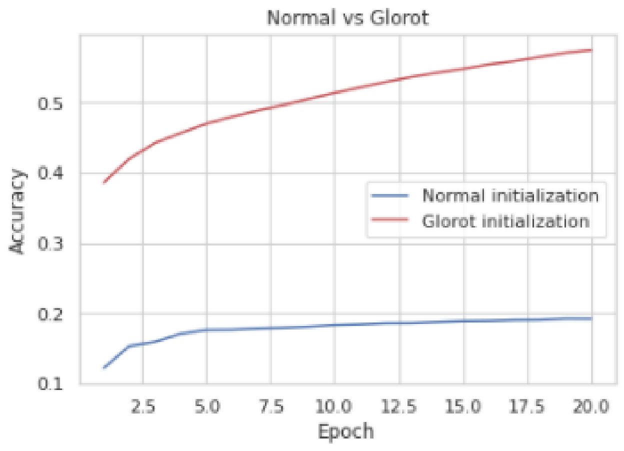


Figure 4: Accuracy Comparison

This experiement proves that weight initialization can actually have a profound impact on the final results of a network, since the only difference between the two experiment is the weights initialization techniques and the difference is very clear. So, choosing the right method for initialization is necessary for the model's ultimate performance.

**Question 3** (Hyperparameter Search). (10)
for this part of the report, we kept some of the default MLP parameters (learning rate=0.01,batch size=64) then we tried different combinations of model architectures(2 hidden layers (784, 256) or 4 hidden layers (1024, 512, 64, 64)) and activation functions(eLU or tanh or sigmoid).

The following table resume the validation accuracy for each combination at the end of 20 epochs:

| | (784, 256) | (1024, 512, 64, 64) |
|---|---|---|
| ReLU | 50.05% | 51.21% |
| tanh | 48.6% | 49.31% |
| sigmoid | 35.16% | 17.39% |

Table 1: The accuracy on the validation set for each combination

The plot below report the accuracy on the validation set for each combination training the model for 20 epochs:

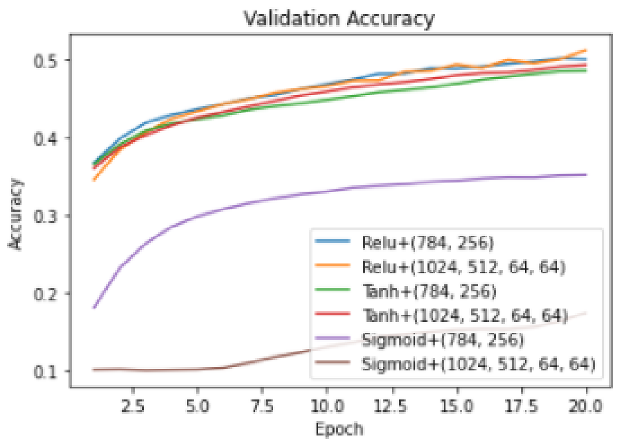


Figure 5: Accuracy Comparison

It is evident that the best accuracy rate on the validation set, at the end of 20 epochs, corresponds to the combination of **Relu activation function with 4 hidden layers (1024, 512, 64, 64)**.

**Question 4** (Validate Gradients using Finite Difference)**.** (15)

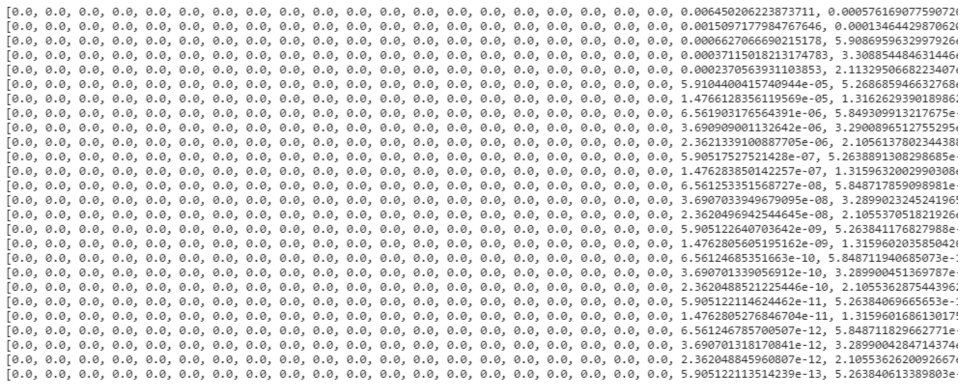
4.1 Evaluate the finite difference gradients.

Here we want to check if our implementation of back propagation by computing numerical gradients and compare it with gradients from back propagation.

Using the pre trained MLP model and test it using one single sample: we take the already trained weights and update p parameters each time, by computing the numerical gradients using the formula below:

$$\nabla_i^N = \frac{L(\theta_1, \ldots, \theta_{i-1}, \theta_i + \epsilon, \theta_{i+1}, \ldots, \theta_p) - L(\theta_1, \ldots, \theta_{i-1}, \theta_i - \epsilon, \theta_{i+1}, \ldots, \theta_p)}{2\epsilon}$$

Here we set 30 values of N and p=100 using the sample number 130. The figure below show a screenshot of the results:



```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.006450206223873711, 0.00057616907759072
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0015097177984767646, 0.00013464429870620
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0006627066690215178, 5.908695963299792
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.00037115018213174783, 3.308854484631446
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0002370563931103853, 2.113295066822340
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5.9104400415740944e-05, 5.268685946632768
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.4766128356119569e-05, 1.31626293901898
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 6.561903176564391e-06, 5.849309913217675e
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.690909001132642e-06, 3.290089651275529
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.3621339100887705e-06, 2.10561378023443
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5.90517527521428e-07, 5.263889130829868 5e
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.476283850142257e-07, 1.315963200299030
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 6.561253351568727e-08, 5.848717859098981e
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.6907033949679095e-08, 3.28990232452419
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.3620496942544645e-08, 2.105537051821926
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5.905122640703642e-09, 5.263841176827988e
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.4762805605195162e-09, 1.31596020358504
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 6.56124685351663e-10, 5.848711940685073e-
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.690701339056912e-10, 3.289900451369787e
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.3620488521225446e-10, 2.10553628754439
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5.905122114624462e-11, 5.26384069665653e-
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.4762805276846704e-11, 1.31596016861301
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 6.561246785700507e-12, 5.848711829662771e
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.690701318170841e-12, 3.289900428471437
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.362048845960807e-12, 2.105536262009266
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5.905122113514239e-13, 5.263840613389803e
```

Figure 6: Finite difference gradients

and this plot show finite difference gradients values :

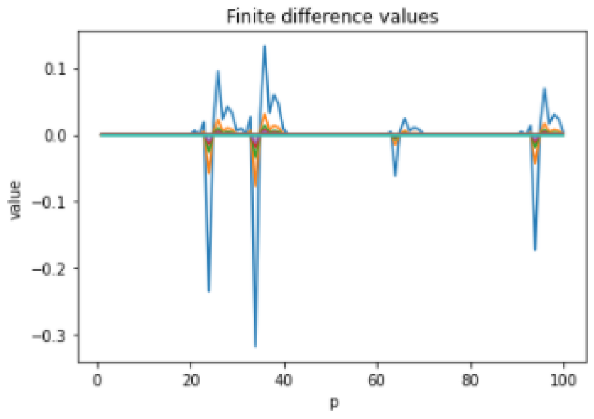


Figure 7: Finite difference gradients values

4.2 Plot the maximum difference between the true gradient and the finite difference gradient ($\max_{1 \leq i \leq p} |\nabla_i^N - \frac{\partial L}{\partial \theta_i}|$) as a function of $N$. Comment on the result.
The following plot record the maximum difference between the true gradient and the finite difference gradient:

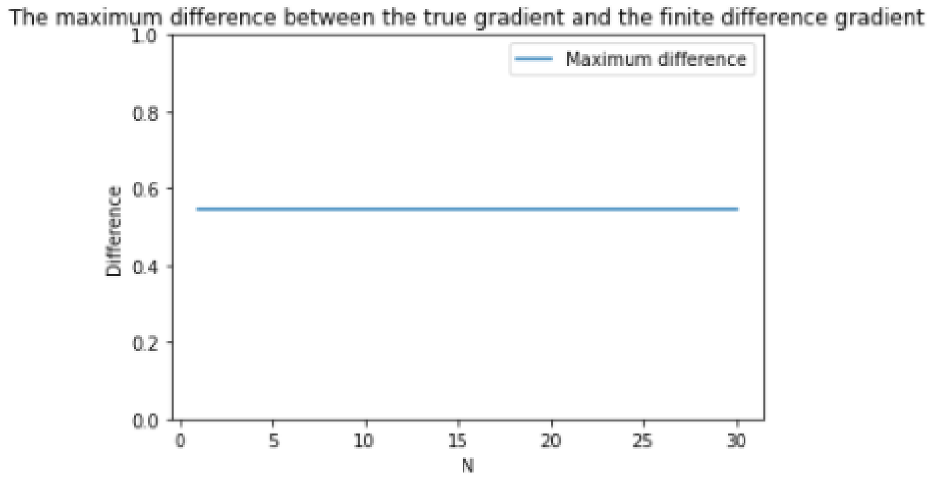


Figure 8: The maximum difference between the true gradient and the finite difference gradient

We can notice that there is a difference between he maximum difference between the true gradient and the finite difference gradient, and we can see the two quantities separately:

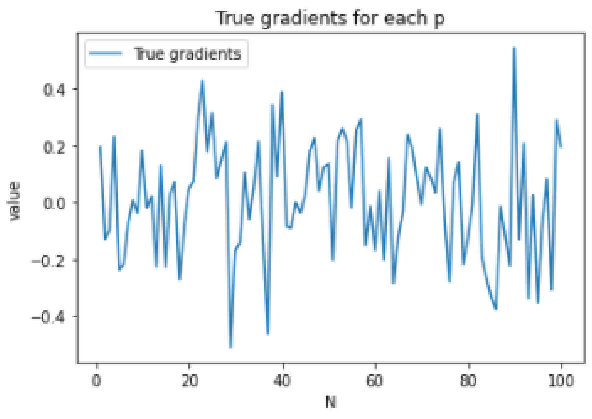


Figure 9: True gradients

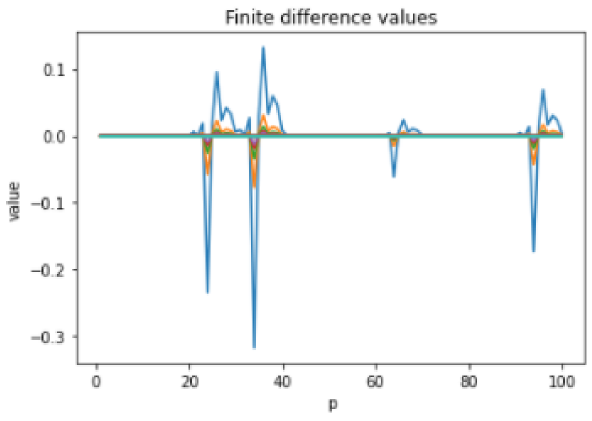


Figure 10: Finite difference gradients values

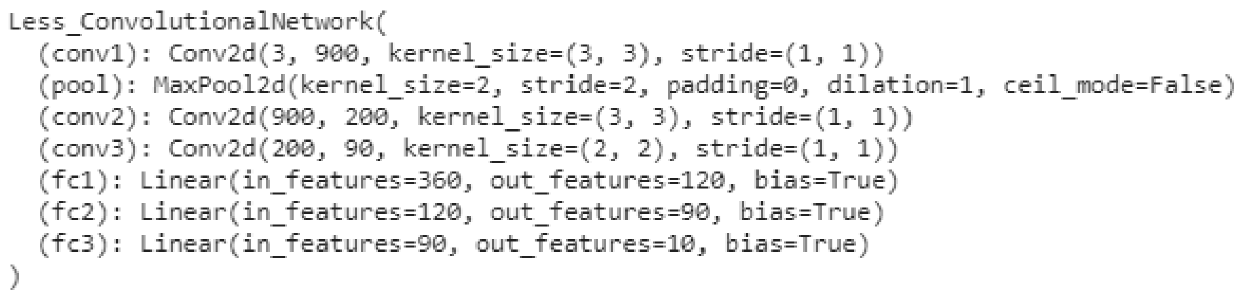
From this, we can deduce that our model can't perform well even with small changes which can clarify the accuracy of the model in the previous question.

**Question 5** (Convolutional Neural Networks)**.** (25) In this question, you will implement and train a convolutional network on CIFAR10 using PyTorch. To have a reasonable training time, you should train your CNN on GPU.

Learning rate=0.001

5.1 Come up with a CNN architecture with more or less similar number of parameters as the best MLP architecture found in Question 3.

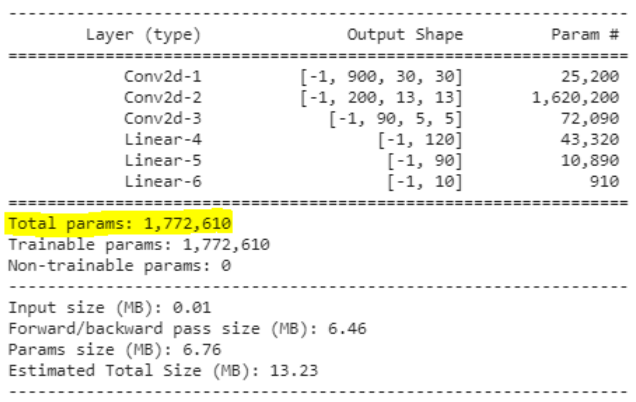
As mentioned earlier, the best MLP architecture is the one with 4 hidden layers (1024, 512, 64, 64) and relu activation function. Compute the number of parameters of this architecture we found:
$NumberParametresinMLP = 3072 * 1024 + 1024 + 1024 * 512 + 512 + 512 * 64 + 64 + 64 * 64 + 64 + 64 * 10 + 10 = 3709194$
There are tons of different CNN architectures with less number of parameters as the MLP architecture. The proposed architecture is presented in the image below:



```
Less_ConvolutionalNetwork(
  (conv1): Conv2d(3, 900, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(900, 200, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(200, 90, kernel_size=(2, 2), stride=(1, 1))
  (fc1): Linear(in_features=360, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=90, bias=True)
  (fc3): Linear(in_features=90, out_features=10, bias=True)
)
```

Figure 11: The proposed architecture

We can check that it has a less number of parameters using the function 'summary'



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 900, 30, 30]          25,200
            Conv2d-2          [-1, 200, 13, 13]       1,620,200
            Conv2d-3            [-1, 90, 5, 5]           72,090
            Linear-4                  [-1, 120]          43,320
            Linear-5                   [-1, 90]          10,890
            Linear-6                   [-1, 10]             910
================================================================
Total params: 1,772,610
Trainable params: 1,772,610
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 6.46
Params size (MB): 6.76
Estimated Total Size (MB): 13.23
----------------------------------------------------------------
```

Figure 12: Model Summary

5.2 Train your CNN and compare its performance to your MLP after 20 epochs. Plot the training and validation *loss* curves for both models (on a single plot). Plot the training and validation *accuracy* curves for both models (on a single plot).

Here, we compare our best MLP model with CNN model proposed in the previous question. Despite the fact that the CNN has less parameters than the MLP, it performed better results. In fact, the gap difference between the accuracy, with the training or validation set, of each model is very clear, shown in the plot below:

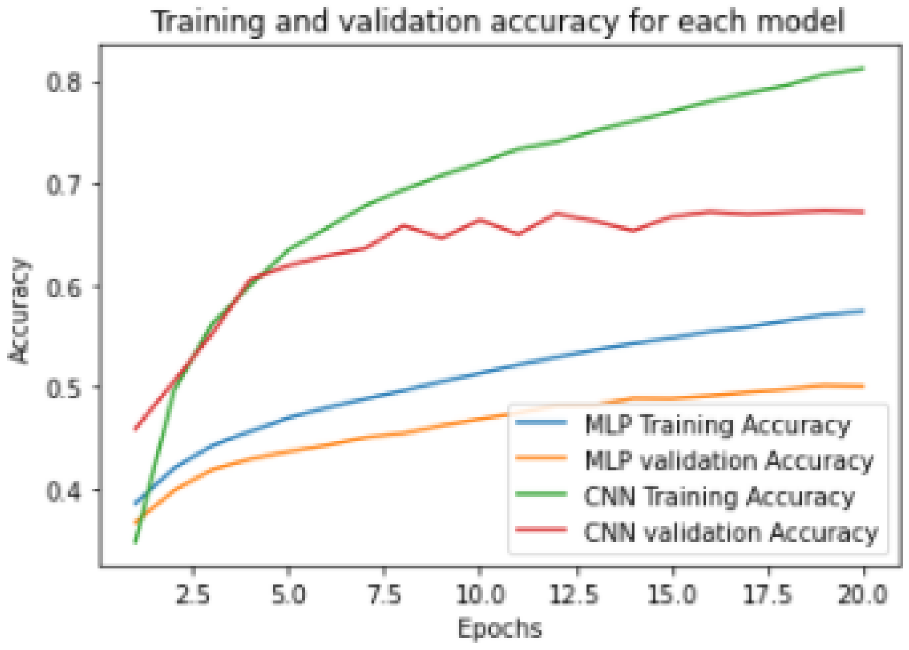


Figure 13: Accuracy comparison

The CNN model reach more than 80% accuracy in the training set, whereas the MLP get only less than 60% accuracy , at the end of 20 epochs.We notice the same for the validation set where the accuracy decrease for the two models but the validation set accuracy for CNN still higher than the MLP.

The figure below shows the loss curves for both models:

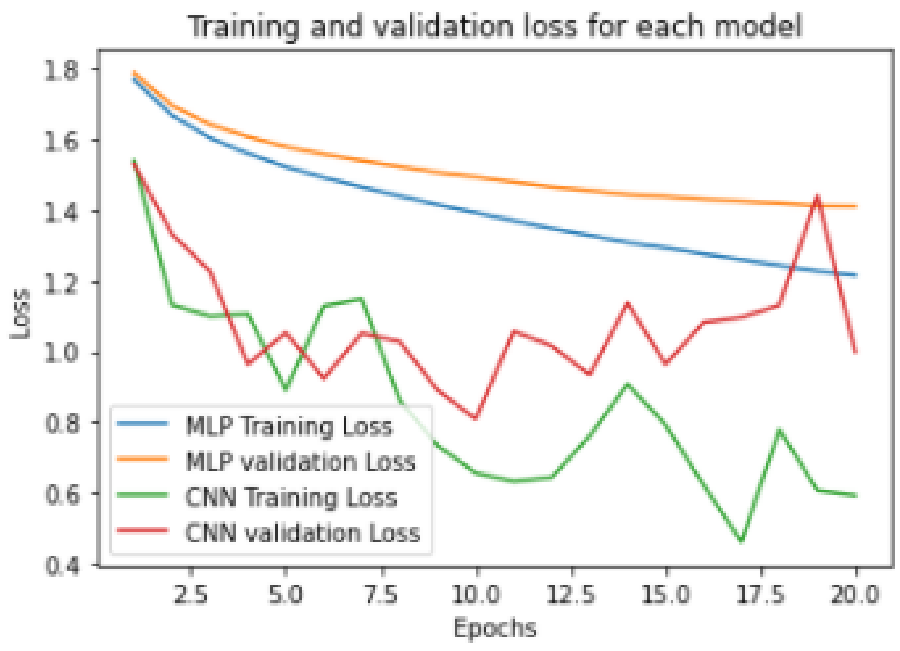


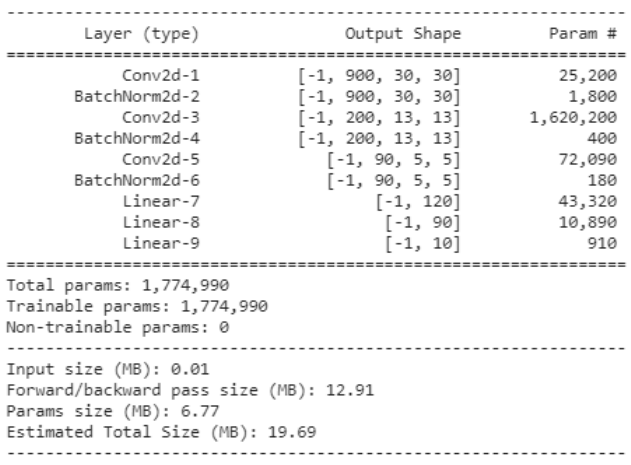
Figure 14: Loss comparison

Even with the unstable loss, the CNN model has lower loss than the MLP model.

5.3 Explore *one* regularization technique

Here **batch normalization** is used as a regularization technique. It has the effect of stabilizing the learning process because the main idea of batch normalization is to normalize the inputs of each layer in such a way that, they have a mean activation output zero and a unit standard deviation. ( Dropout technique didn't give us a good results).

Our CNN model with regularization is presented in the figure below:



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 900, 30, 30]          25,200
       BatchNorm2d-2         [-1, 900, 30, 30]           1,800
            Conv2d-3         [-1, 200, 13, 13]       1,620,200
       BatchNorm2d-4         [-1, 200, 13, 13]             400
            Conv2d-5            [-1, 90, 5, 5]          72,090
       BatchNorm2d-6            [-1, 90, 5, 5]             180
            Linear-7                 [-1, 120]          43,320
            Linear-8                  [-1, 90]          10,890
            Linear-9                  [-1, 10]             910
================================================================
Total params: 1,774,990
Trainable params: 1,774,990
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 12.91
Params size (MB): 6.77
Estimated Total Size (MB): 19.69
----------------------------------------------------------------
```

Figure 15: Model Summary with regularization

The results of training the regularized CNN with the vanilla CNN after 80 epochs are shown in the plots below.

The first one is dedicated to the accuracy, for training and validation sets, for both models. It is clear that the regularization make the difference since we can notice that the regularized CNN perform better than the vanilla model. Especially when it comes to the validation set, the regularized CNN has not the best accuracy only but also the gap between the training and the validation accuracy is is more less than the gap for the vanilla model.

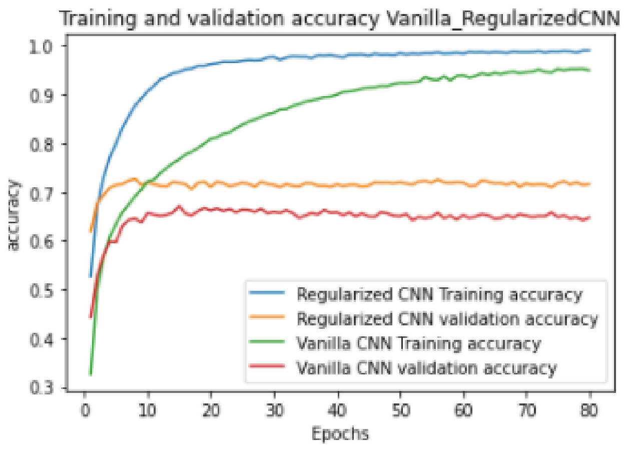


Figure 16: Accuracy comparison

When it comes to the loss, the models didn't work very well. The loss curves, for training and validation sets for both models are unstable. The plot of training loss continues to decrease, however, the plot of validation loss decreases to a point and begins increasing again. Therefore, we can deduce that the models are overfitting.

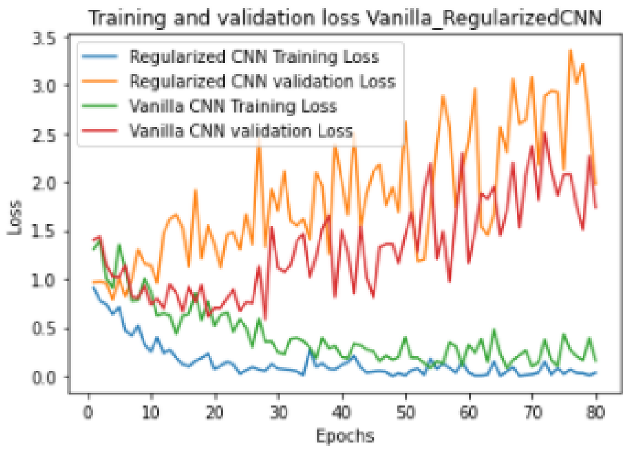


Figure 17: Loss comparison