DLCV-Lab 4

Kush Gupta

1 Steps used to Train CNN

- Import the required libraries, and load the data using the data loader, define batch size.
- Define the number of convolutional layers, kernel size, and the number and size of the fully connected layers in the network.
- Define the loss function and the optimizer.
- When training the model put the model in training mode, and clear the gradients using zero_grad().
- Pass the data to the model, calculate the predictions and compute the loss between the predictions and actual targets.
- Backpropagate the loss and take the optimizer step. Repeat the process for the whole batch.

2 Steps used to Test CNN

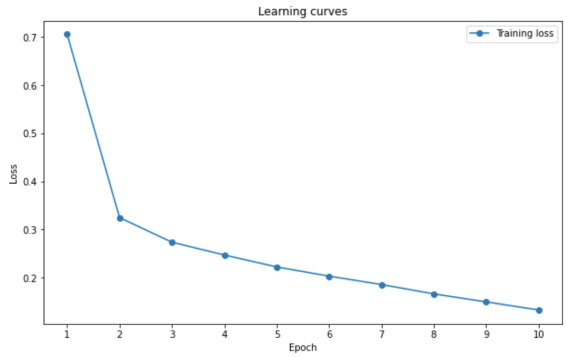
- For evaluating the model put the model in evaluation mode.
- Since we do not need to keep track of the gradients for testing, use torch.no_grad() method.
- Loop the test data to the model, calculate the predictions and pass them to the softmax function to calculate the probability of belonging to a certain class.
- For the calculated probabilities take the highest probability as the class label.

3 Exercise 1: CNN for MNIST Dataset

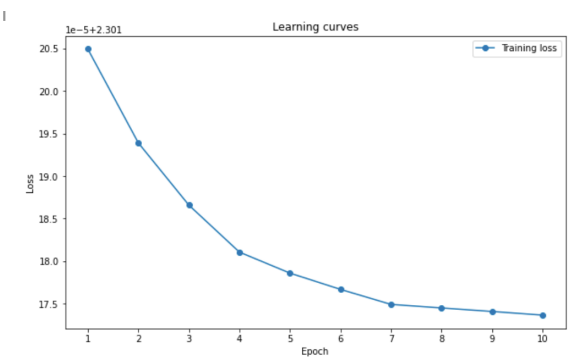
- Model 1:- The layers of the first type of the neural network used to train on the MNIST dataset is shown in figure 1. This network contained two convolutional layers, of kernel size 3x3. A max-pooling layer with kernel size 2x2 has a stride of 2 and two drop-out layers to avoid overfitting. It has two fully connected layers, the size of each layer is shown in 1.
- For the first 3 runs, I used this model with different hyperparameters. The loss curves for the first three runs can be observed in figure 2. As shown in table 1, the highest accuracy achieved with this model was 99.12% with SGD as the optimizer, learning rate as 0.01, and cross-entropy as the loss function.
- Model 2:- The second type of CNN used to train on the MNIST dataset is shown in figure 3. Modifying the previous network, this network contained two convolutional layers, of kernel size 3x3 with 16 output channels. A max-pooling layer with kernel size 2x2 has a stride of 2 and one drop-out layer to avoid overfitting. It has two fully connected layers, the size of each layer is shown in 3.
- For the next 2 runs, I trained this model earlier for 10 epochs and when I observed good accuracy I further ran it for 20 epochs. The loss curves for these two runs can be observed in figure 4. As shown in table 1, the highest accuracy achieved with the second model was 99.12% with SGD as the optimizer, learning rate as 0.01, and cross-entropy as the loss function.

```
Neural_net(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dc1): Dropout(p=0.25, inplace=False)
  (dc2): Dropout(p=0.5, inplace=False)
  (mp1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=6272, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=10, bias=True)
)
```

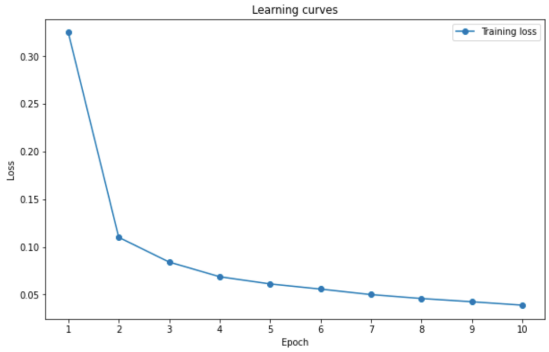
Figure 1: Model- 1 used for MNIST dataset



(a) Run 1



(b) Run 2



(c) Run 3

Figure 2: Training Loss curves for 1st Model

- For the different types of CNNs trained on the MNIST dataset for classifying 10 digits, the highest accuracy achieved by both models was **99.12**%.
- The top 10 misclassified digits by the model can be seen in the figure 5.
- With the best model from lab 3.3, we obtained an accuracy of 97.81% for the MNIST dataset.

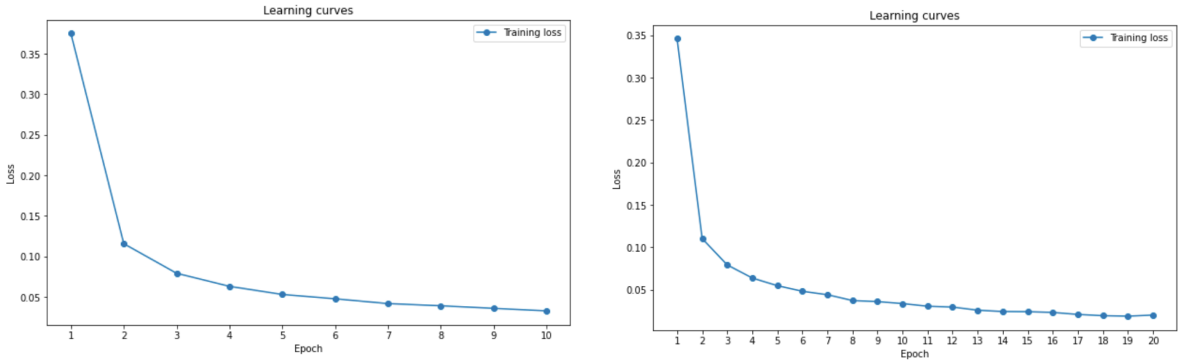
4 Exercise 2: CNN on CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 RGB images for 10 classes, with 6000 images per class. Some images from the dataset are shown in Figure 6. Before using the images preprocessing was performed.

- To begin with, I used model-2 from exercise 1 as the first neural network to train on the CIFAR-10 dataset as shown in figure 7. This network contained two convolutional layers, of kernel size 3x3. A max-pooling layer with kernel size 2x2 has a stride of 2 and two drop-out layers to avoid overfitting. It has two fully connected layers, the size of each layer is shown in 7.

```
Neural_net(
  (conv1): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dc2): Dropout(p=0.5, inplace=False)
  (mp1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=3136, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=10, bias=True)
)
```

Figure 3: Model- 2 used for MNIST dataset



(a) Run 1 (b) Run 2

Figure 4: Training Loss curves for 2nd Model

- The loss curve for the first run can be observed in figure 8. As shown in table 2, I trained this model for 20 epochs with SGD as the optimizer, learning rate as 0.01, and cross-entropy as the loss function. The accuracy achieved was 65.39%.

- To make the model deeper and in order to increase the accuracy of the CNN, I modified the previous model and added a new fully connected layer as shown in figure 9. This network contained two convolutional layers, of kernel size 3x3. A max-pooling layer with kernel size 2x2 has a stride of 2 and two drop-out layers to avoid overfitting. It has three fully connected layers, the size of each layer is shown in 9.

- Initially, I trained this modified network for 20 epochs with SGD as the optimizer, learning rate as 0.01, and cross-entropy as the loss function and could see that the accuracy increased to 70.5% which is almost an increase of 5% compared to the previous run. Hence, to further increase the accuracy I trained the model again for 30 epochs with the same parameters but this time the accuracy dropped by 3% as shown in table 2, because the network starts overfitting the data.

- The loss curves for the second and the third runs could be observed in the figure. 10.

- As the network starts to overfit the data, I further modified the network. This network contained two convolutional layers (of different sizes), kernel size 3x3. A max-pooling layer with kernel size 2x2 has a stride of 2 and two drop-out layers to avoid overfitting. I reduced the number of fully connected layers from three to two, the new network can be seen in figure 11.

- After reducing the trainable parameters, I trained this new light network for 50 epochs with SGD as the optimizer, learning rate as 0.01, and cross-entropy as the loss function and could see that the accuracy dropped to 64%. This reduction of 3% in accuracy w.r.t last run mainly resulted as the network overfitted the data. The results are shown in table 2.

- The loss curve for the 4th run could be observed in the figure. 12.

Run	Model	Num. Epochs	Optimizer	Learning rate	Momentum	Accuracy(%)
1	Model-1	10	SGD	0.01	-	97.96
2	Model-1	10	ASGD	0.001	-	11.35
3	Model-1	10	SGD	0.01	0.9	99.12
4	Model-2	10	SGD	0.01	0.9	99.09
5	Model-2	20	SGD	0.01	0.9	99.12

Table 1: Demonstrates the 2 different neural networks, their accuracies, and various hyperparameters used to train the model on the MNIST dataset. The Cross entropy loss function is used for all the runs.

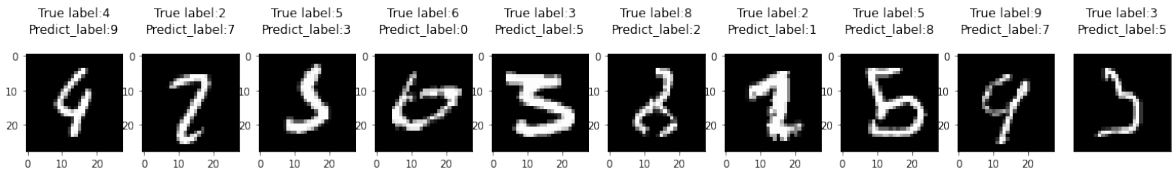


Figure 5: Top 10 worst classified results

- I trained the best model from lab 3.3 on the CIFAR-10 dataset and obtained an accuracy of 50.8% as mentioned in the table 2. The loss curve can be observed in figure 13.
- For the different types of CNNs trained on the CIFAR-10 dataset for classifying the images in 10 classes, the highest accuracy was achieved by model-2 **70.51**%.
- The top 10 misclassified images by the model can be seen in figure 14.

Run	Model	Num. Epochs	Optimizer	Momentum	Accuracy(%)
1	Model-1	20	SGD	0.9	65.39
2	Model-2	20	SGD	0.9	**70.51**
3	Model-2	30	SGD	0.9	67.18
4	Model-3	50	SGD	0.9	63.7
5	Best Model-3.3	20	SGD	-	50.8

Table 2: Demonstrates the 2 different neural networks, their accuracies, and various hyperparameters used to train the model on the MNIST dataset. The loss function used was cross entropy loss and the learning rate = 0.01, momentum =0.9 was used for all the runs.

5 Exercise 3: Data augmentation

As noticed earlier, the accuracy of our convolutional networks trained on the CIFAR 10 database did not demonstrate good results, because the images in the database are in poor resolution and for each class, we only have 6000 images which is not sufficient to train a good classifier. Due to this, we will use data augmentation techniques to increase the amount of training data for learning in order to achieve higher accuracy. After exploring the images in the CIFAR dataset for the different classes I choose to modify the parameters like brightness, saturation, contrast, and sharpness in the images and horizontal flip to perform data augmentation. Other augmentation techniques like vertical flipping would be unnecessary to apply due to data characteristics and since the resolution of the images is bad, cropping and zooming could be inadequate augmentation.

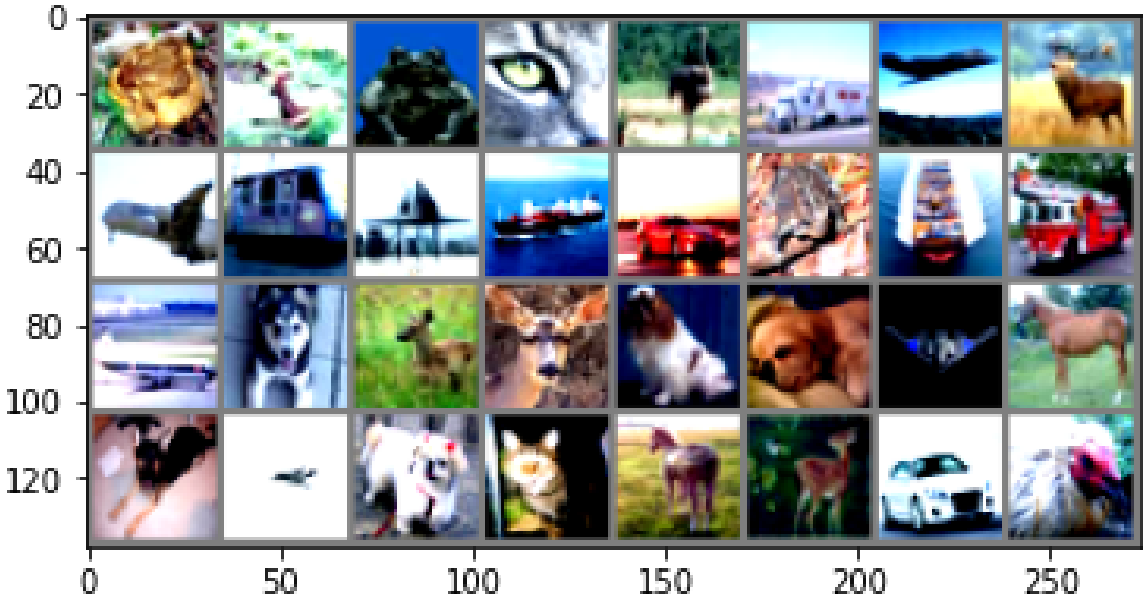


Figure 6: Images from CIFAR-10 dataset.

```
Neural_net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dc1): Dropout(p=0.25, inplace=False)
  (dc2): Dropout(p=0.5, inplace=False)
  (mp1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=4096, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=10, bias=True)
)
```

Figure 7: Model- 1 used for CIFAR-10 dataset

- For training the model with the augmented data (1st run), I used the model-2 from exercise 2 which gave the accuracy of 70% earlier. After training the model for 20 epochs with the same hyper-parameters as used in exercise 2, I observed that the accuracy increased by 2%. So, the model's new accuracy was 72.66%
- In order to obtain better accuracy, I tried to train the same model for 40 epochs. After training I could see that the accuracy increased very slightly, the new accuracy obtained was 72.78%.
- The loss curve for 1st and 2nd runs during the model training can be seen in figure 15

6 Exercise 4: Transfer learning / fine-tuning on CIFAR10 dataset

To achieve better results, I tried to fine-tune the resnet50 model pre-trained on imagenet databse. Imagenet is a huge database containing 14,197,122 annotated images. I choose resnet50 because it is a dense network, and even though it's not fast since it's quite precise it is used extensively.

```
model = models.resnet50(pretrained='imagenet')
input_features = model.fc.in_features
model.fc = nn.Linear(input_features,10)
```

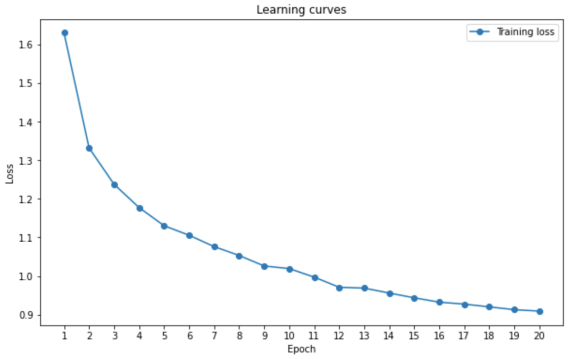
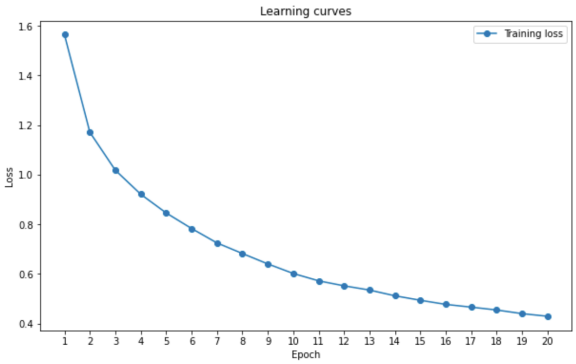


Figure 8: Loss curve for 1st run of the CNN on CIFAR-10.

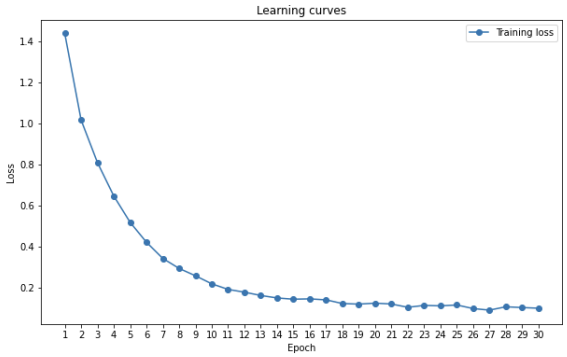
```
Neural_net(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dc1): Dropout(p=0.25, inplace=False)
  (dc2): Dropout(p=0.5, inplace=False)
  (mp1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=8192, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=10, bias=True)
)
```

Figure 9: Model- 2 used for CNN on CIFAR-10.

- To fine tune, trained the pretrained resnet 50 for 10 epochs with the CIFAR-10 augmented data and could observe that the accuracy of the new classifier is 85.91%.
- The loss curve for the model training can be seen in figure 16



(a) Loss curve for 2nd run



(b) Loss curve for 3rd run

Figure 10: Loss curve for 2nd and 3rd run of the CNN on CIFAR-10.

```
Neural_net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dc1): Dropout(p=0.25, inplace=False)
  (dc2): Dropout(p=0.5, inplace=False)
  (mp1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=8192, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=10, bias=True)
)
```

Figure 11: Model- 3 used for CNN on CIFAR-10.

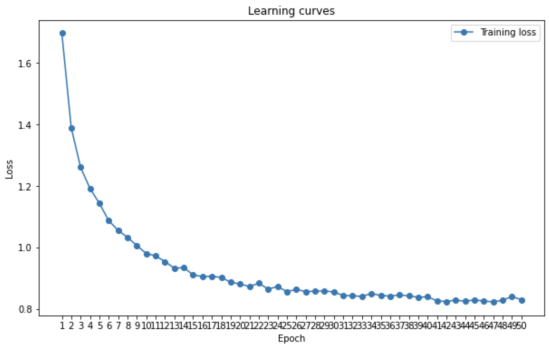


Figure 12: Loss curve for 4th run of the CNN on CIFAR-10.

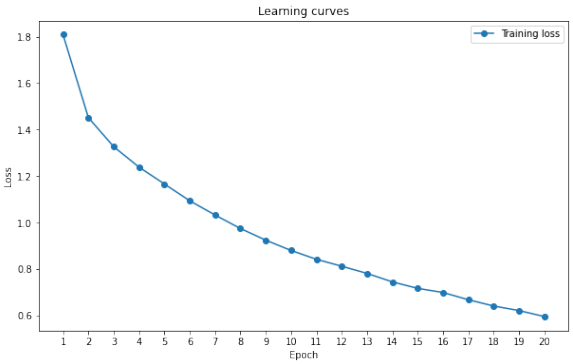


Figure 13: Loss curve for 5th run of the CNN on CIFAR-10.

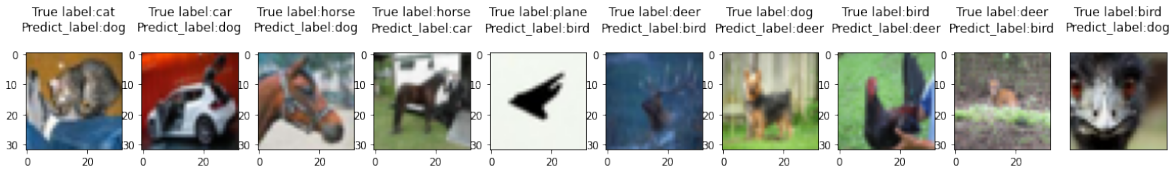
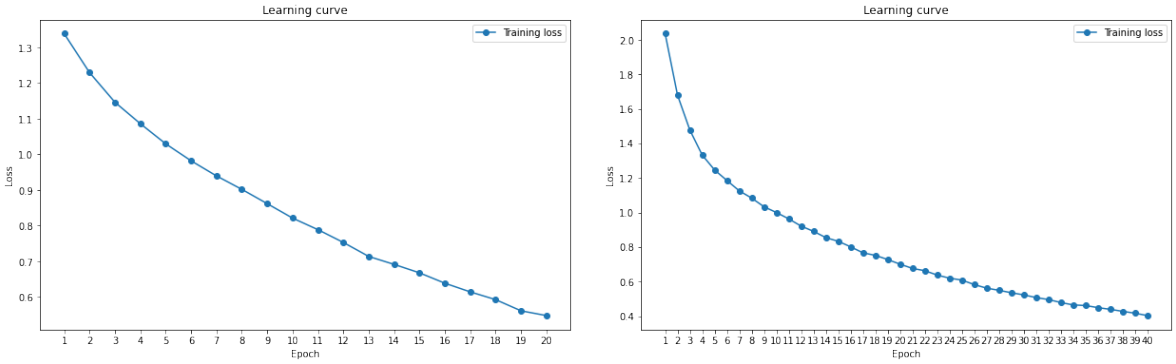


Figure 14: Top 10 worst classified results



(a) Loss curve for 1st run

(b) Loss curve for 2nd run

Figure 15: Loss curve for training the CNN on augmented data.

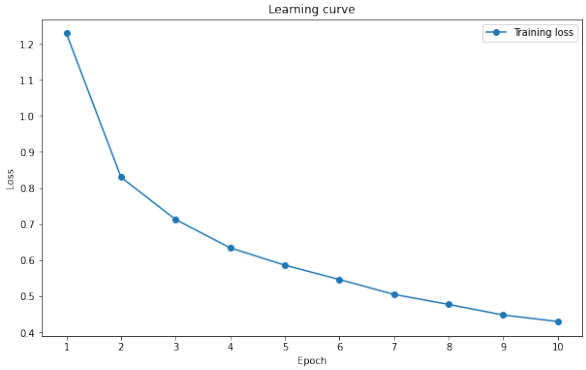


Figure 16: Loss curve for fine tuning resnet 50.