E0-270: Machine Learning

Assignment-2

Akash Maji
CSA, 24212

April 18, 2025

Problem 1: Image Classification

Answer 0:

We use the code at: `https://github.com/parag1604/Basic-Image-Classification`. Upon running the *version9.py* file as *python3 main.py 9*, we observe the following trends as shown below in Figure-1. We see that without any changes, the **Net** model is giving the following

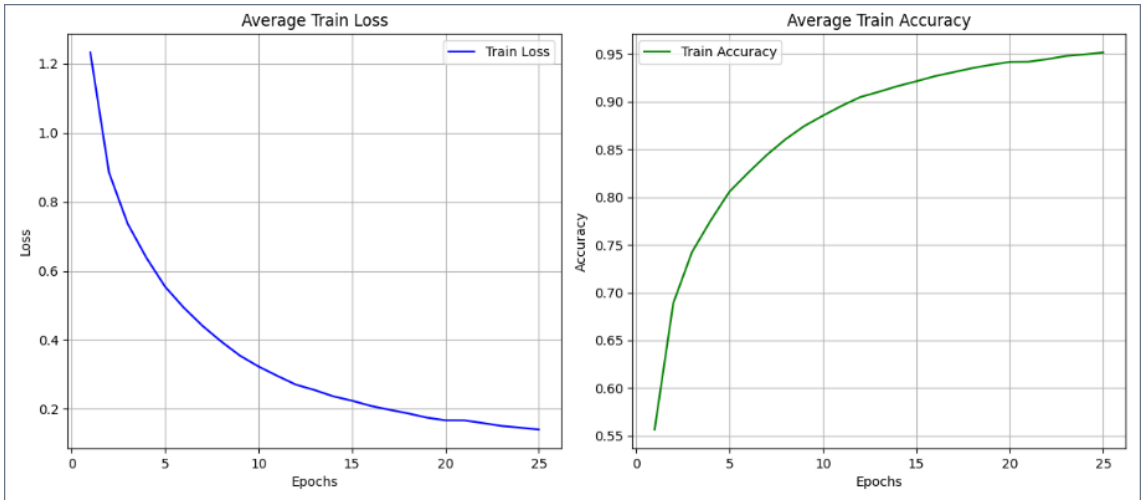

Figure 1: Average test loss and accuracy on CIFAR-10

particulars as shown in Table-1. The training was done with the following details:

Metric	Value
Average Test Loss	0.8494
Average Test Accuracy	0.7970

Table 1: Evaluation Metrics on Test Set

This is how the training accuracy and train error changes over 25 epochs as shown in Figure-2:

Parameter	Value
Dataset	CIFAR-10
Model Parameter Count	655,882
Epochs	25
Batch Size	64
Optimizer	Adam
Learning Rate	0.001
Weight Decay	0

Table 2: Training Configuration

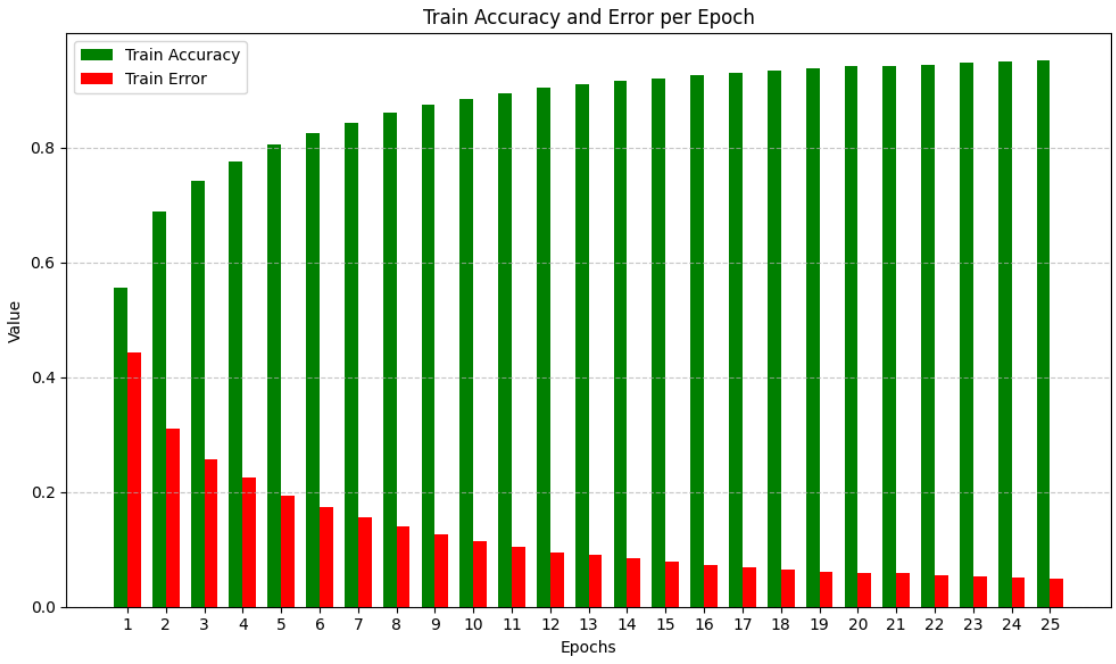

Figure 2: Train loss and accuracy on CIFAR-10

We now change the various hyperparameters to reach higher level of test accuracy. We try out various configurations with varying batch sizes, learning rates, number of epochs to see which configuration is giving best values. We show for learning rates 0.01 and 0.001 respectively.

Batch	25	50	75	100
32	0.7653	0.7634	0.7574	0.7621
64	0.7727	0.7751	0.7869	0.7842
128	0.7691	0.7819	0.7907	0.7805
256	0.7506	0.7799	0.7798	0.7810
512	0.7542	0.7696	0.7807	0.7898
1024	0.7779	0.7567	0.7918	0.7662

Batch	25	50	75	100
32	0.7960	0.8098	0.8159	0.8220
64	0.7902	0.7909	0.7916	0.7923
128	0.7297	0.8046	0.8395	0.8543
256	0.7570	0.7831	0.8092	0.8353
512	0.7833	0.7734	0.7635	0.7536
1024	0.7416	0.7768	0.8119	0.8370

The highest accuracy was **0.8543** for batch-size=**128** and epochs=**100** at learning rate=**0.001**. Clearly, we improved from our earlier accuracy of **0.7970**.

Listing 1: improve test accuracy

```
from torchvision import transforms
transform_train = transforms.Compose([
    # crop the image
    transforms.RandomCrop(32, padding=4),
    # flip the image horizontally
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    # noramlize the image
    transforms.Normalize((0.4914, 0.4822, 0.4465),
                         (0.2023, 0.1994, 0.2010))
])
```

Next, we apply data augmentation and data normalizations like random cropping, random flipping, mean and standard deviation normalization which helps in faster convergence, and better accuracy. Finally, we reach a test accuracy of **0.8719**.

Metric	Value
Average Test Loss	0.8719
Average Test Accuracy	0.3844

Answer 1:

We use the code at: `https://github.com/techxzen/pytorch-residual-networks`
In the given generic implementations, we want to have depths of 20, 56, 110 for **ResNet** models. From the formula given: depth = 6n + 2, we get n = 3, 9, 18 respectively for ResNet-20, ResNet-56 and ResNet-110.

We observe the following trends when we train the 3 models for **100** epochs with learning rate = **0.001** and batch size = **32**, using **Adam** optimizer.

We appropriately save the model weights as:

- *weights-new/v9-cifar10-resnet-20.pth*

Table 3: Train and Test Loss/Accuracy for ResNet Models

Model	Train Loss	Train Accuracy	Test Loss	Test Accuracy
ResNet-20	0.0010	0.9999	0.9151	0.8482
ResNet-56	0.0001	1.0000	0.8591	0.8735
ResNet-110	0.0001	1.0000	0.8637	0.8800

- *weights-new/v9-cifar10-resnet-56.pth*
- *weights-new/v9-cifar10-resnet-110.pth*

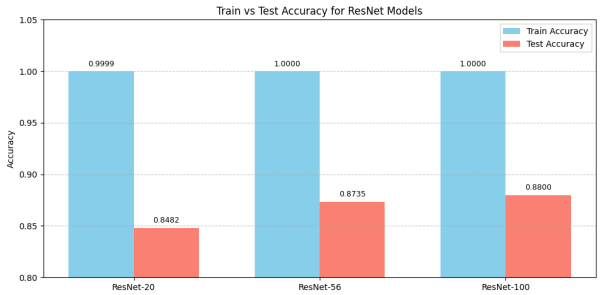

Figure 3: Train and Test Accuracy

Figure 4: Train and Test Loss

Answer 2:

We use the code at: `https://github.com/techxzen/pytorch-residual-networks`
In the given generic implementations, we want to have depths of 20, 56, 110 for **PlainNet** models. From the formula given: depth = 6n + 2, we get n = 3, 9, 18 respectively for PlainNet-20, PlainNet-56 and PlainNet-110.

We observe the following trends when we train the 3 models for **100** epochs with learning rate = **0.001** and batch size = **32**, using **Adam** optimizer.

Table 4: Train and Test Loss/Accuracy for PalinNet Models

Model	Train Loss	Train Accuracy	Test Loss	Test Accuracy
PlainNet-20	0.0051	0.9989	0.9543	0.8299
PlainNet-56	0.3165	0.8882	0.8537	0.7465
PlainNet-110	1.6808	0.3479	1.6730	0.3511

We appropriately save the model weights as:

- *weights-new/v9-cifar10-plainnet-20.pth*
- *weights-new/v9-cifar10-plainnet-56.pth*
- *weights-new/v9-cifar10-plainnet-110.pth*

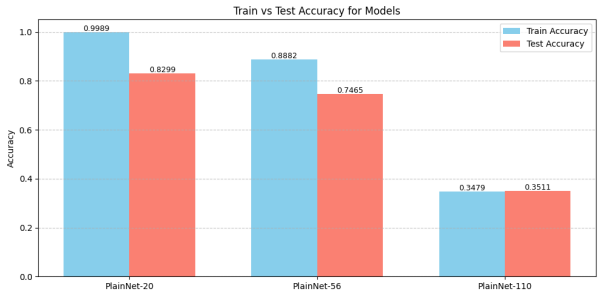

Figure 5: Train and Test Accuracy

Figure 6: Train and Test Loss

Answer 3:

Upon performing the experiments, we observe that as we increase the depth of the model, it performs better and better. Also, **ResNet** model performs better than its counterpart **PlainNet** model. **ResNet** models make use of residual connections that help in reducing vanishing gradient problem during weight update at back-propagation by allowing gradients flow through skip-connections. We also see that in **PlainNet**, the accuracy falls with increasing depth, making model ineffective, however **ResNet** improves its performance.

Table 5: Test Loss and Accuracy Comparison of NET, PlainNet, and ResNet Models

Model	**Test Loss**	**Test Accuracy**
NET	0.3844	0.8719
PLAINNET-20	0.9543	0.8299
PLAINNET-56	0.8537	0.7465
PLAINNET-110	1.6730	0.3511
RESNET-20	0.9151	0.8482
RESNET-56	0.8591	0.8735
RESNET-110	0.8637	0.8800

Problem 2: Visualizing Loss Landscape

Listing 2: running code to plot surfaces

```
python plot_loss_surface.py --model_path='net/v9-cifar10-net.pth' --model_type='net'

python plot_loss_surface.py --model_path='plainnet-20/v9-cifar10-plainnet-20.pth' --model_type='plainnet20'

python plot_loss_surface.py --model_path='plainnet-56/v9-cifar10-plainnet-56.pth' --model_type='plainnet56'

python plot_loss_surface.py --model_path='plainnet-100/v9-cifar10-plainnet-100.pth' --model_type='plainnet100'

python plot_loss_surface.py --model_path='resnet-20/v9-cifar10-resnet-20.pth' --model_type='resnet20'

python plot_loss_surface.py --model_path='resnet-56/v9-cifar10-resnet-56.pth' --model_type='resnet56'

python plot_loss_surface.py --model_path='resnet-100/v9-cifar10-resnet-100.pth' --model_type='resnet100'
```

ResNet models are better than **PlainNet** models as they have higher accuracy and lower loss. With increasing depth, **PlainNet** performance decreases, while that of **ResNet** increases.

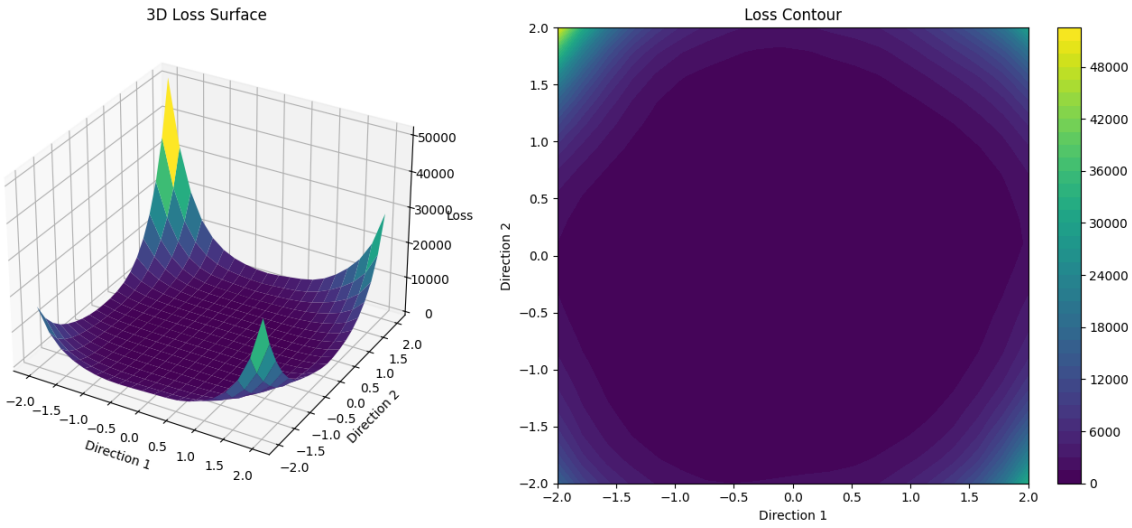

Figure 7: Loss Landscape for Net Model

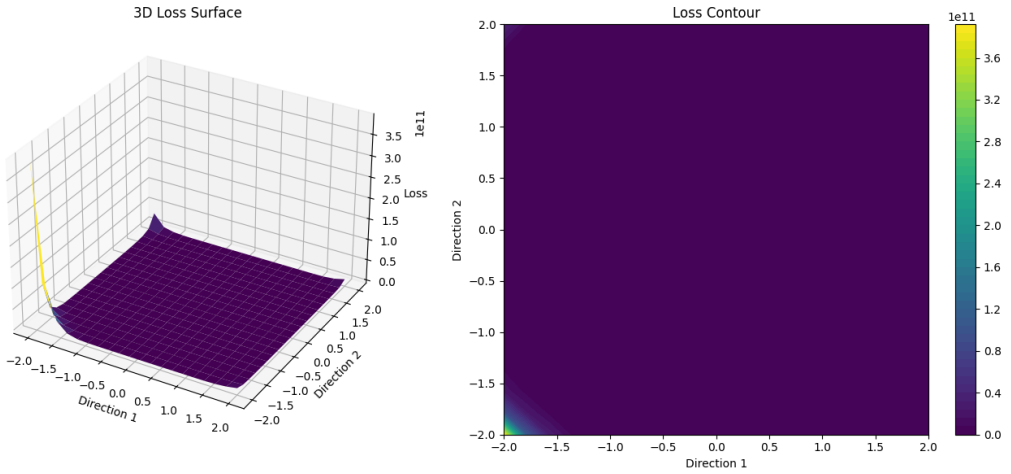

Figure 8: Loss Landscape for resNet-20 Model

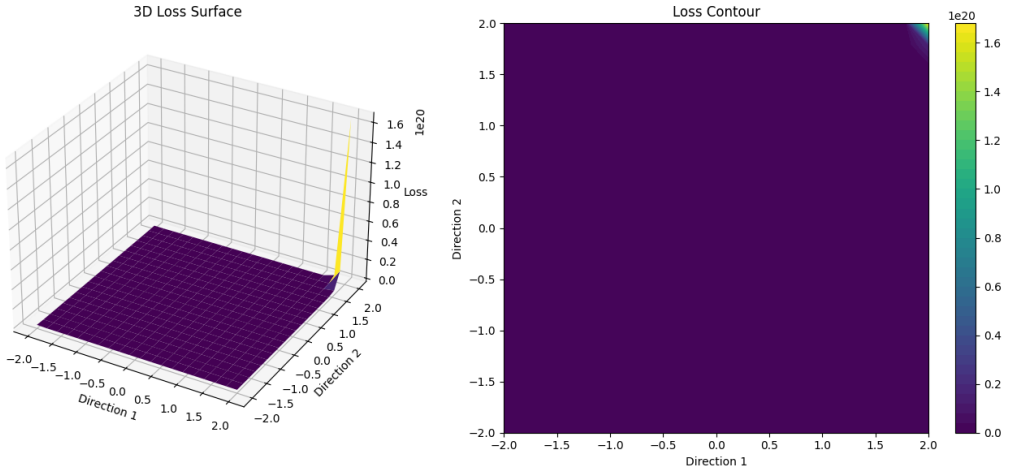

Figure 9: Loss Landscape for resNet-56 Model

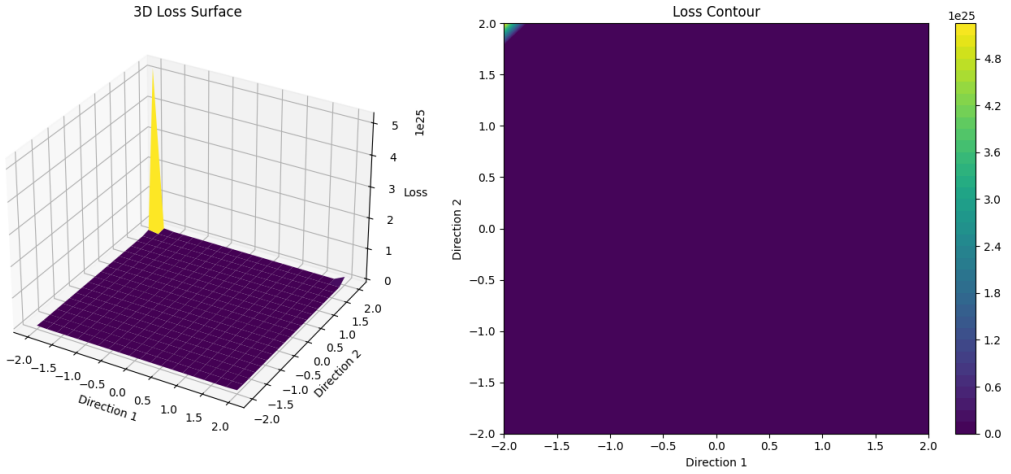

Figure 10: Loss Landscape for resNet-56 Model

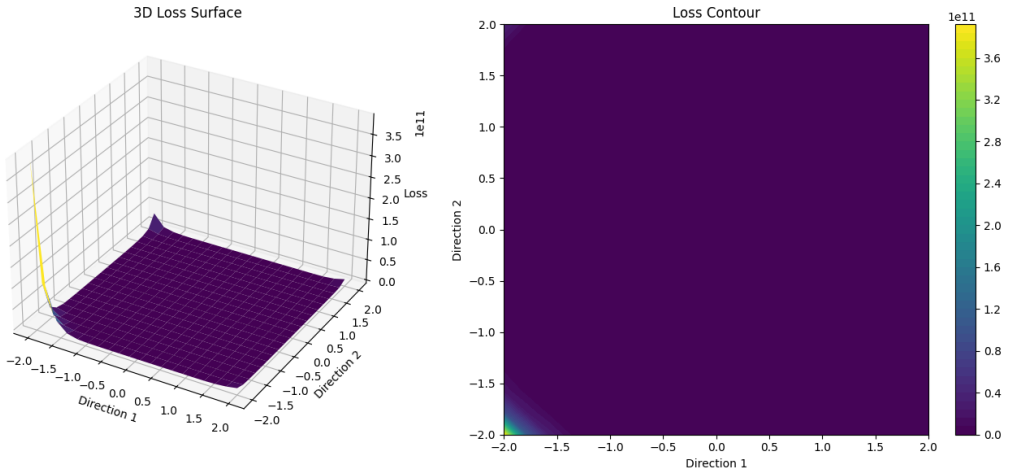

Figure 11: Loss Landscape for plainNet-20 Model

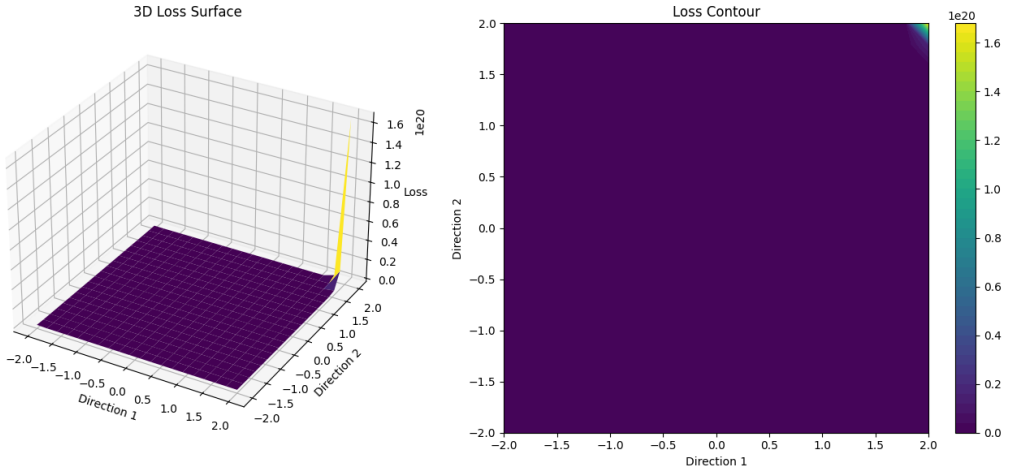

Figure 12: Loss Landscape for plainNet-56 Model

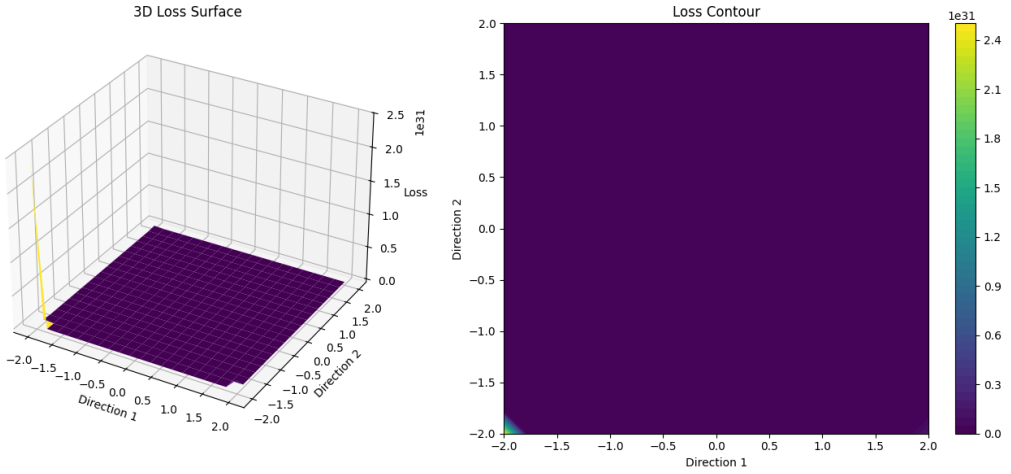

Figure 13: Loss Landscape for plainNet-110 Model

References:

https://arxiv.org/abs/1512.03385
https://github.com/tomgoldstein/loss-landscape

System Used:

The following system was used:

Component	Specification
System	$Ryzen7 - 5800H$
Cores	8
Threads	16
Base Clock	$3.2 - GHz$
RAM	$24 - GB$