EEE: 515 - Machine Vision - Homework 2

Praveen Paidi
Student ID: 1225713099
Fall 2023

October 27, 2023

1 Problem status

1. **Problem 1 :**
   For 1.3: The results are furthur summarized and mentioned neatly. For 1.4: Completed fully.

2. **Problem 2 : Completed Convergence is with respect to computational capacity of my Local CPU/GPU. Used custom hyper paramaters and number of epochs as mentioned.**

2 Problem 1:

A. CIFAR 10 Dataset.

LeNet Style Architecture

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Listing 1: LeNet Architecture

Original dataset from CIFAR 10 is being shown in the figure 1

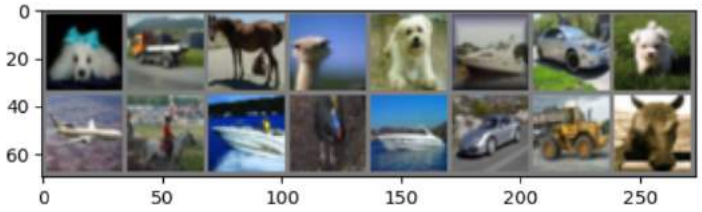

Figure 1: Original Dataset of CIFAR10

The following are the hyper parameters being used to get the optimal accuracy.
Batch size: 6
Optimizer : Adam
Learning rate : 0.001
Criterion : CrossEntropyLoss
Number of Epochs: 10

Accuracy percentage on Test dataset: 62

B.Dataset Augmentation;

Dataset augmentation has been performed for achieving better accuracy.
Rotation, translation, Flipping, Adding small amount of noise being added to augment the data.

LeNet Style Architecture, Augmentation with Transforms

```
transform_train = transforms.Compose([
transforms.ToTensor(),
transforms.RandomRotation(degrees=random.choice([0, 20,50, 0, 0, 0])),
transforms.RandomHorizontalFlip(p=random.choice([0, 0.2,0.5, 0, 0, 0])),
transforms.RandomVerticalFlip(p=random.choice([0, 0.2,0.5, 0, 0, 0])),
transforms.RandomAffine(degrees=(random.choice([0, 20,50, 0, 0, 0]))),
transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

Listing 2: Tranformation

Augmented dataset from CIFAR 10 is being shown in the figure 2 after performing transformation shown in listing2.

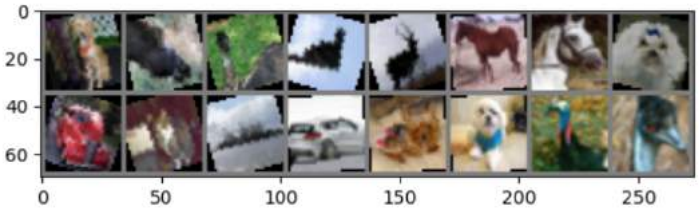

Figure 2: Augmented Dataset of CIFAR10

The following are the hyper parameters being used to get the optimal accuracy.
Batch size: 6
Optimizer : Adam
Learning rate : 0.001
Criterion : CrossEntropyLoss
Number of Epochs: 10

The accuracy of the different augmented datasets is 63.5 percentage.
There is a increase in accuracy by 1.5 percentage due to the dataset augmentation.

C.ResNet Architecture;

I implemented ResNet18. The following is the implemented architecture.

Listing 3 shows the resnet 18 architecture. I added original layer after making one convolution instead of classical renset appoach to make the accuracy better.

Skip connections are being shown in the listing 4.
I used Makelayer in listing 3 to generate the sequential layers and skip connections shown in listing 4.

Click Reference: to visit the website.

```
self.conv1 = nn.Conv2d(3, 64, kernel_size=3,stride=1, padding=1, bias=
    False)
self.bn1 = nn.BatchNorm2d(64)
self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
self.linear = nn.Linear(512*block.expansion, num_classes)
```

Listing 3: ResNet Architecture

```
(layer2): Sequential( BasicBlock
((conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
   1), bias=False)
(bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
   track_running_stats=True)
(conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
   1), bias=False)
(bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
   track_running_stats=True)
Sequential(
(0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
   track_running_stats=True))
```

Listing 4: Skip Connections of ResNet 18

Layer making for Resnet higher layers than 18 are followed by code in listing.

```
self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=1, bias=False)
self.bn1 = nn.BatchNorm2d(planes)
self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
                                stride=stride, padding=1, bias=False)
self.bn2 = nn.BatchNorm2d(planes)
self.conv3 = nn.Conv2d(planes, self.expansion *
                                planes, kernel_size=1, bias=False)
self.bn3 = nn.BatchNorm2d(self.expansion*planes)
self.shortcut = nn.Sequential()
if stride != 1 or in_planes != self.expansion*planes:
    self.shortcut = nn.Sequential(
        nn.Conv2d(in_planes, self.expansion*planes,
                  kernel_size=1, stride=stride, bias=False),
        nn.BatchNorm2d(self.expansion*planes)
    )
```

Listing 5: Skip Connections of ResNet higher layers

Hyper parameters: The following are the hyper parameters being used to get the optimal accuracy.
Batch size: 128
Optimizer : SGD
Learning rate : 0.01
Criterion : CrossEntropyLoss
Number of Epochs: 5

Accuracy percentage on Test dataset with Resnet 18: 81
Accuracy percentage on Test dataset with Resnet 34: 80
Accuracy percentage on Test dataset with Resnet 50: 77
Accuracy percentage on Test dataset with Resnet 101: 76
Accuracy percentage on Test dataset with Resnet 151: 76

Summary of results: It is outperforming the earlier LeNet Architecture.The ResNet 18 performing better in my case according to the hyperparamters, other higher layers might be overfitting on the data. Hyperparamter tuning may result in better performance of higher layers.
As of my results, ResNet 18 is getting highest accuarcy on test dataset with 81 percentage and preferable to use for the specified training dataset and testing dataset.

D. Dataset Bias and Greydata
Prepared the 2 datasets:
1. Concatenating the 5 classes of grey scale and 5 classes of color images.

2. Complete grey scale of 10 classes.

Preparation of dataset in listing 6. Testing set is two different sets:
1. Complete color dataset
2. Complete Grey dataset

Complete grey scale dataset in figure 3.

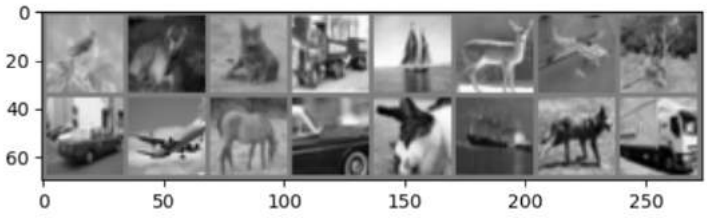

Figure 3: Grey data

Skewed dataset in figure 4.

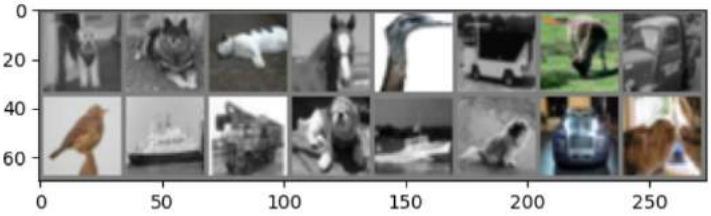

Figure 4: Skewed data

```
threshold = 4
condition_indices = [i for i, j in enumerate(label) if j > threshold]
non_condition_indices = [i for i, j in enumerate(label) if j <= threshold]
GREYSET = torch.utils.data.Subset(trainset_grey, condition_indices)
COLORSET = torch.utils.data.Subset(trainset, non_condition_indices)
FINAL = ConcatDataset([GREYSET, COLORSET])
```

Listing 6: DATASET PREPARATION

ResNet 18 applied on these datasets.
Batch size: 128
Optimizer : SGD
Learning rate : 0.01
Criterion : CrossEntropyLoss
Number of Epochs: 5

Summary of results:
Accuracy on Color Test dataset from 1st Method(Skewed): 37
Accuracy on Color Test dataset from 2nd Method(Grey): 77
Accuracy on Grey Test dataset from 1st Method(Skewed): 45
Accuracy on Grey Test dataset from 2nd Method(Grey): 81

The Accuracy of complete grey scale image train dataset is outperforming the skewed train dataset on Color test images and as well as grey test images .

Training on complete grey scale images and testing on color and grey scale images is the better option rather than other one.

3 Problem 2:

1. DC-GAN on Celeb A Face Dataset

Loss Plot of DC-GAN after running it for 2 epochs, iterations being plotted in the adjacent figure 5.

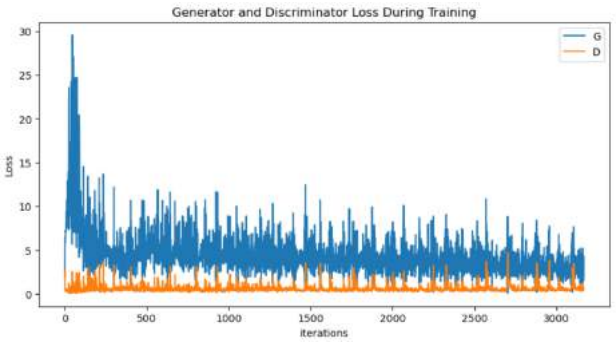

Figure 5: Augmented Dataset of CIFAR10

The following hyper parameters being used.
Number of images used for training : 202599
Number of epochs: 2
batchsize = 128
Size of feature maps in generator (ngf) = 64
Size of feature maps in discriminator(ndf) = 64
learning rate for optimizers = 0.0002
Beta1 hyperparameter for Adam optimizers (beta1) = 0.5

Figure 6: Original Images

Figure 7: Fake Images

2.Dataset of Coloured Squares

Dataset of 1000 squares with different colors, different sizes, different orientations are generated using DC-GAN.Squares are being shown in figure8.

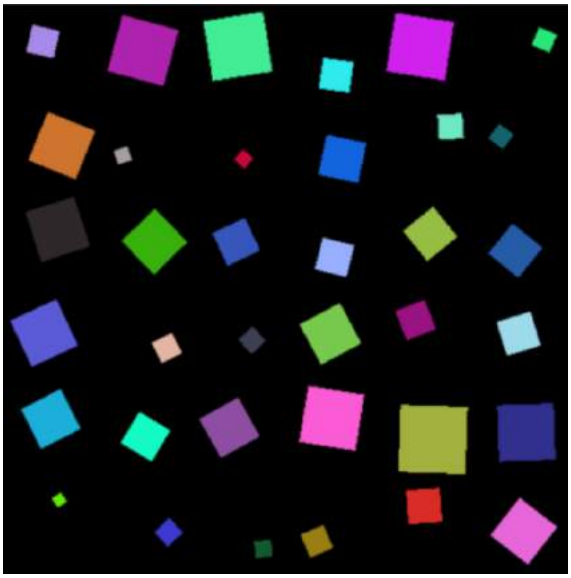

Figure 8: Squares dataset

3. Final Output of generated squares.
1000 images of squares used over 200 epochs could not generate the fake squares.Hyperparamters being as mentioned below:
Batchsize = 128, learning rate for optimizers = 0.0002
Size of feature maps in generator (ngf), discriminator(ndf) = 64
Beta1 hyperparameter for Adam optimizers (beta1) = 0.5

Then, I increased the dataset to 30000 and 20000 respectively in figure 9 and figure 10 with the same hyperparameters except change in number of epochs to 50.

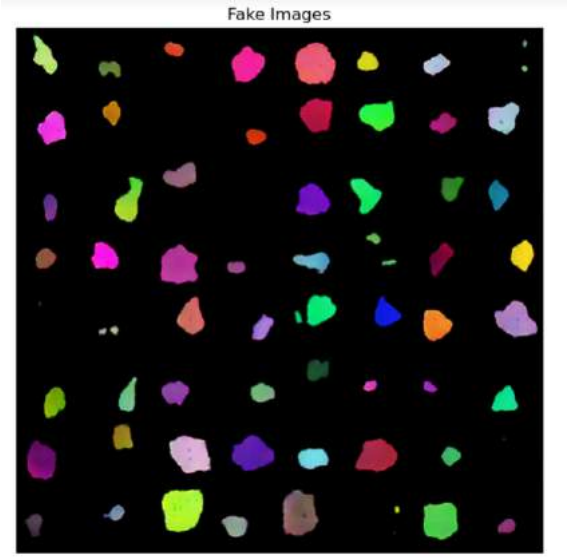

Figure 9: Fake images of N= 30000

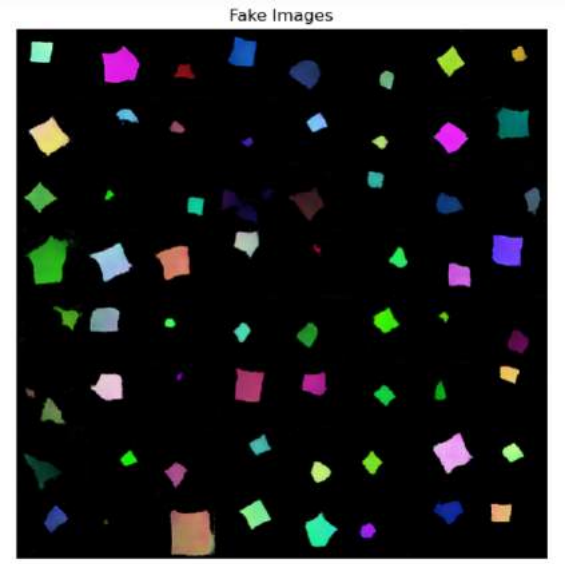

Figure 10: Fake images of N= 20000

Then increased the dataset furthur to 50000 with number of epochs being 20 produced the comparison of original and fake square result in figure11.

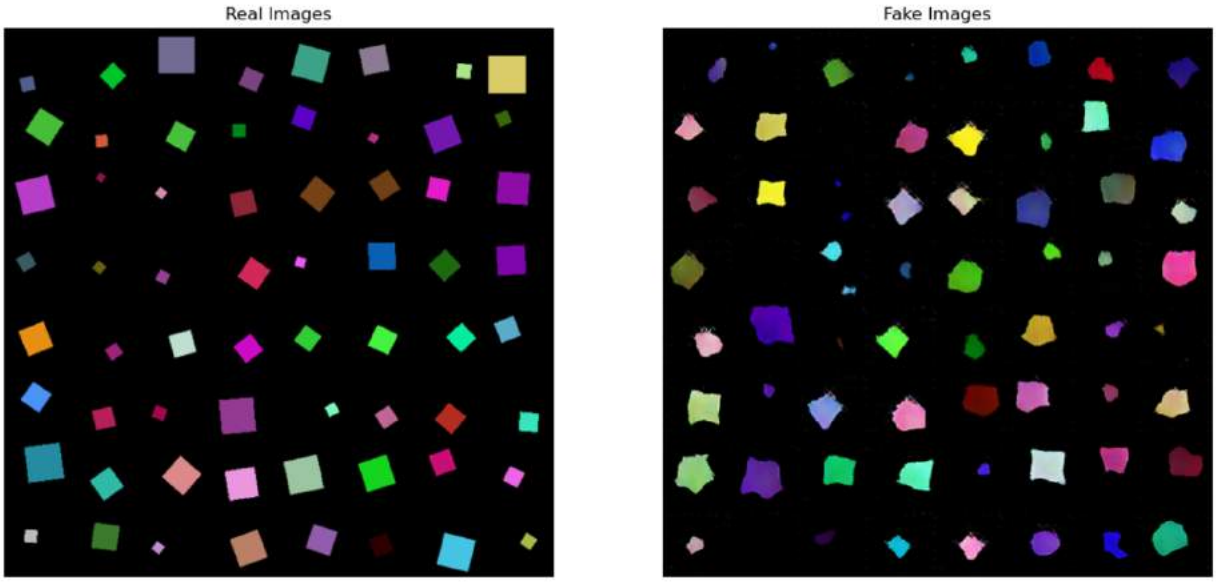

Figure 11: Fake images of N= 50000

The loss plot of DC-GAN after running it for 20 epochs, iterations being plotted in figure 12.

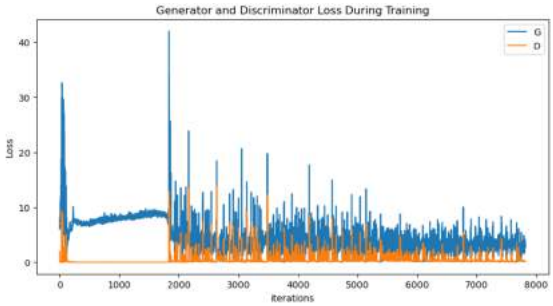

Figure 12: Loss Plot

4.Dataset of Animal of 1000 images

Dataset of 1000 animals are collated, then images are being resized to 64 X 64 and hwon in figure 13.

Figure 13: Animals dataset

5.Trained on Animal dataset and generated Following images using following techniques.
Initially dataset of 1000 animals using following parameters.
Number of epochs: 200 ; Batchsize = 128 ; learning rate = 0.0002 , Beta for Adam optimizers (beta1) = 0.5
Size of feature maps in generator (ngf), discriminator(ndf) = 64
Fake image data is not good enough, So I augmented the data with some rotation, flipping , affine transformations. Changed learning rate to 0.0001 to get the final output. Data set being converted to 5000 by including the augmented dataset to original.I changed the number of epochs to 100.
The generated fake images vs original images is being shown in the figure14.

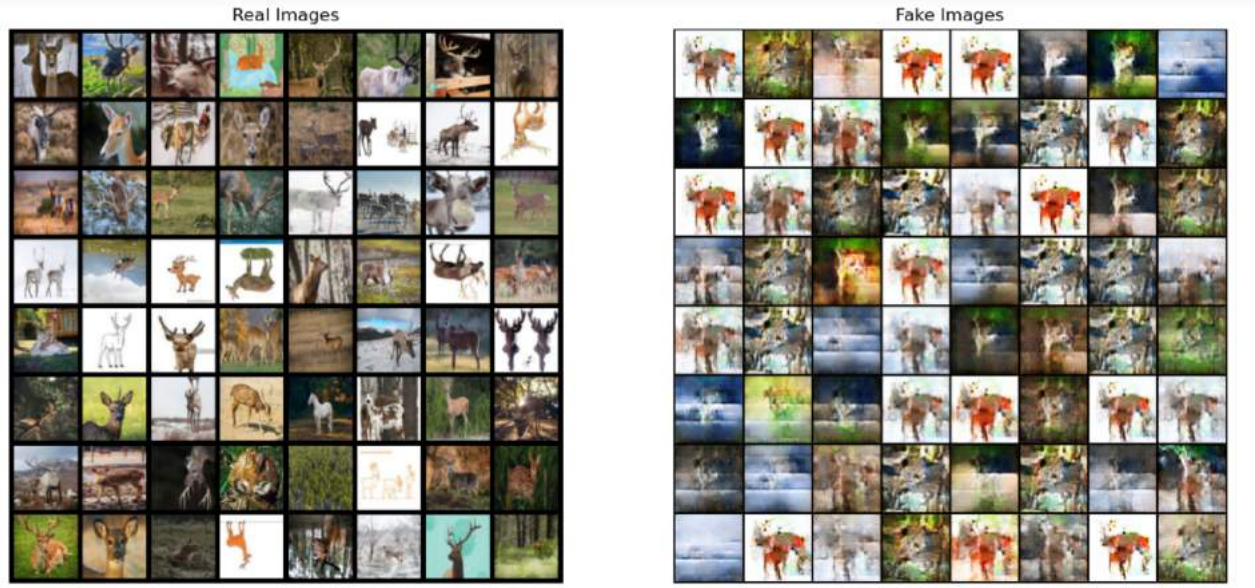

Figure 14: Comparison of original and fake images.

I changed the dataset number of points by optimizing augmentation number of sets such that data won't overfit and generate good fake images.The result being shown in figure15.

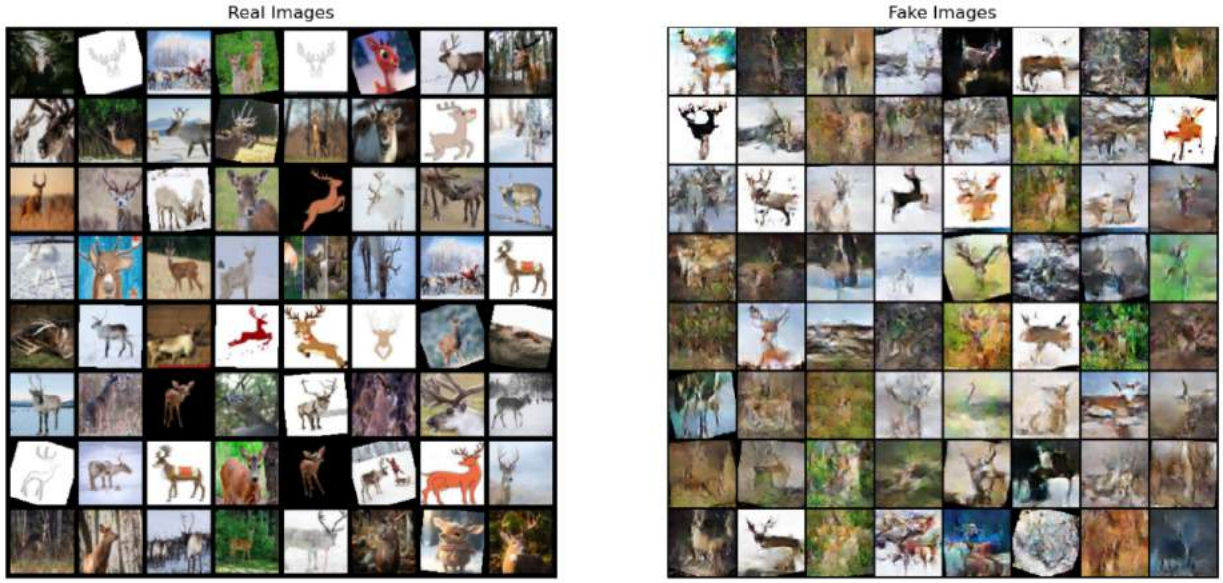

Figure 15: Comparsion of original and fake after optimizing augmentation

The loss plot of DC-GAN after running it for 100 epochs, iterations being plotted in figure 16.

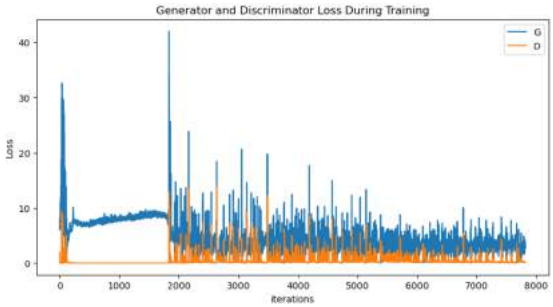

Figure 16: Loss plot of Animal dataset