CS614 Vision Project

Edmond Mbadu

April 29, 2023

Pitch

This project is an image classifier based on the CIFAR-10 dataset. The project is a prelude to a classifier that classifies any images. All that is lacking is more compute power and a bigger dataset.
What we have in mind for this project is the application of image classification in the medical sector where much of the work today still involves diagnosis based on image analysis. Automating these image based diagnosis is the primary goal of the project once done at scale.

Data Source

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
dataset source: https://www.cs.toronto.edu/ kriz/cifar.html

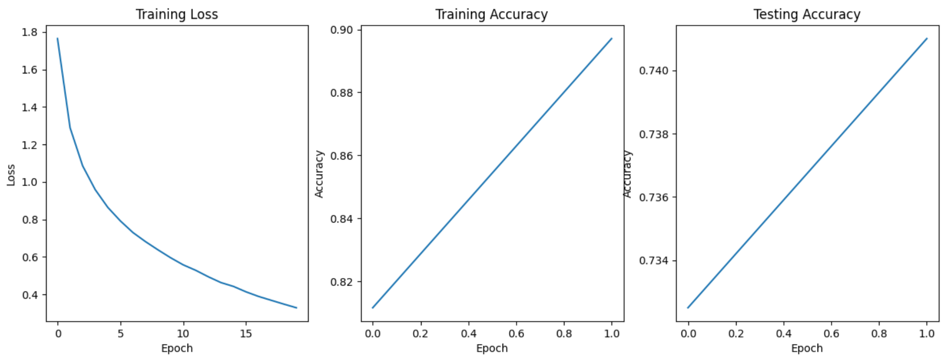
Model (CNN)

After many attempts we arrived at the optimal solution with train and test accuracy being respectively: 83.5 % and 73.74%. We obtained such accuracies with two convolutional layers, one pooling layer and 3 connected layers. Below is the snippet of code for the model:

```
class ConvNet(nn.Module):
    def __init__(self):
        super().__init__()
        # TODO: Define model here
        self.conv1 = nn.Conv2d(3, 15, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(15, 75, 5)
        self.fc1 = nn.Linear(75 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
```

Testing

Below are plots of the metrics used: log loss and accuracy of both the training and testing datasets.



Code and Instructions to run it

To run the code that gets the exact same results as ours run the following command:

```
python3 hw2.py output.png convnet --lr .001 --batch_size 9 --epochs 20 --aug
```

The output.png is the name of the output file that contains all the desired plots
the convnet is the name of the network(note that we were also experimenting with linear which can be used as a command with –linear instead)
The rests are arguments where –lr is the learning rate, –epochs is the number of epochs, and –aug is to augment the data for more inputs.

See the readme file under ReadMe.md for all the other details on how to run the code.

Observation

Two main things happened with the network: as we increased the layers the model started to overfit. To combat this, we augmented the data; hence the –aug parameter. We could have used the ResNet architecture as well, but to run even just 30 epochs takes about an hour, so there was no need go as deep. We suspect that going deeper will yield greater and greater result, (probably about 80+ percent for the test dataset) but we will need other clever ways to augment the data as well.

More Details about hyper-parameters of the CNN

- During the experimentation, an increase in the size the architecture did not yield an increase in the performance. It in fact decreased the performance overall albeit not much.
- The batch size did affect the performance of the network by a factor or 5-7 percent which is not trivial. The lesser the better up to a certain point. Through trial and error I decided to use 8 as the batch size as it yielded the best results. The reason a lower batch size yields better results is partly because with a lower batch size one has more attempts to update parameters making it possible to filter the noise. With bigger batch sizes, the model learns in big chunks making it more susceptible to overfitting, or in other words learn from noise.
- The Adam optimizer was largely inferior to the SGD optimizer to the point where I did not bother using Adam any longer. It had like a 10+% negative impact on the model used compared to the SGD optimizer. The learning rate also did impact the overall performance. In this case 0.001 was the best learning rate. Lower ones were just too slow, and higher ones had very negative impact on the learning phase.
- The learning rate scheduler did not affect the overall performance in any significant way.
- Some data augmentation methods did affect the data more than others. In my case I used a RandomHorizontalFlip. The rest of data augmentation methods taken individually or combined with other methods affected the network negatively or not at all.
- Changing the loss function to a MSE affected the results negatively by about 20% in my final model.