Pranjali Pathre
2019112002

REPORT

Assignment-2: Deep Learning
20th June 2021

Question 3: Hands on!

The report is made analysing the following points:

- With Batch Norm
- Adding new layers
- With Dropout
- Different activation functions at the end
- Different pooling strategies
- Different optimizers
- Basic Augmentation like Rotation, Translation, Color Change

The comparison is shown through error plots. This analysis is based on training a subset of images which were divided into training sets (99.8) and validation set (0.2).

THE NETWORK USED:

```
class Net(nn.Module):
   def __init__(self):
        super().__init__()

        self.conv1 = nn.Conv2d(3, 96, 11, stride=4)
        self.conv1_bn = nn.BatchNorm2d(96)

        self.pool = nn.MaxPool2d(3, stride=2)
        # self.pool = nn.AvgPool2d(3, stride=2)

        self.conv2 = nn.Conv2d(96, 256, 5, stride=1, padding=2)
        self.conv2_bn = nn.BatchNorm2d(256)

        self.conv3 = nn.Conv2d(256, 384, 3, stride=1, padding=1)
        # self.conv4 = nn.Conv2d(384, 384, 3, padding=1)
```

```
        self.conv5 = nn.Conv2d(384, 256, 3, stride=1, padding=1)
        self.conv5_bn = nn.BatchNorm2d(256)

        self.drop_layer = nn.Dropout(p = 0.5)

        self.fc1 = nn.Linear(6400, 4096)
        self.fc1_bn = nn.BatchNorm1d(4096)

        self.fc2 = nn.Linear(4096, 4096)
        self.fc2_bn = nn.BatchNorm1d(4096)

        self.fc3 = nn.Linear(4096, 62)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = F.relu(self.conv3(x))
        # x = F.relu(self.conv4(x))
        x = self.pool(F.relu(self.conv5(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch

        x = F.relu(self.fc1(x))
        # x = self.drop_layer(x)
        # x = self.fc1_bn(x)

        x = F.relu(self.fc2(x))
        # x = self.drop_layer(x)

        # x = self.fc2_bn(x)
        x = self.fc3(x)
        return x
```

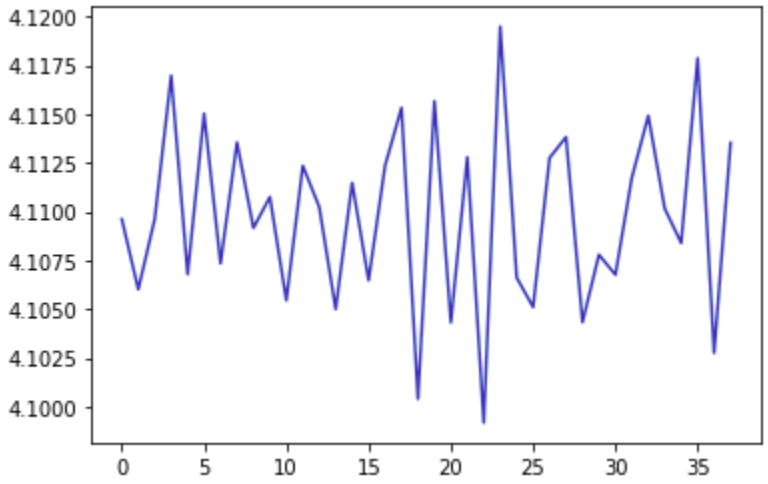
1. Initial Network

```
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 * 47 * 47, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 62)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
```

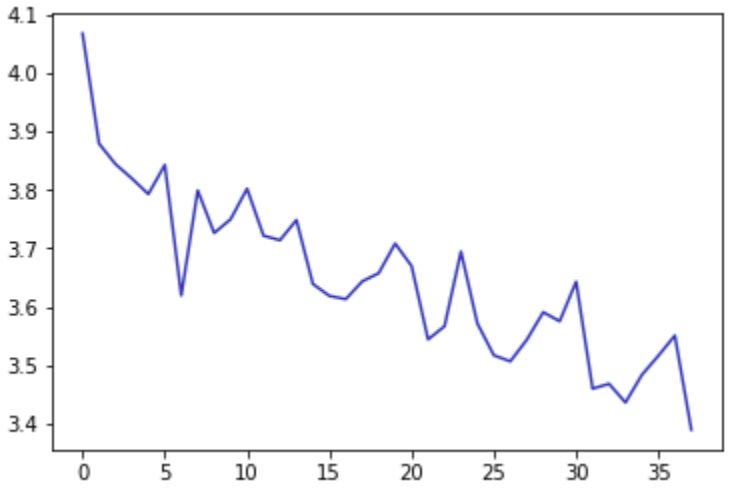
```
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Error plot for each mini batch:



*For all the plots the loss is plotted for every twelfth mini-batch.

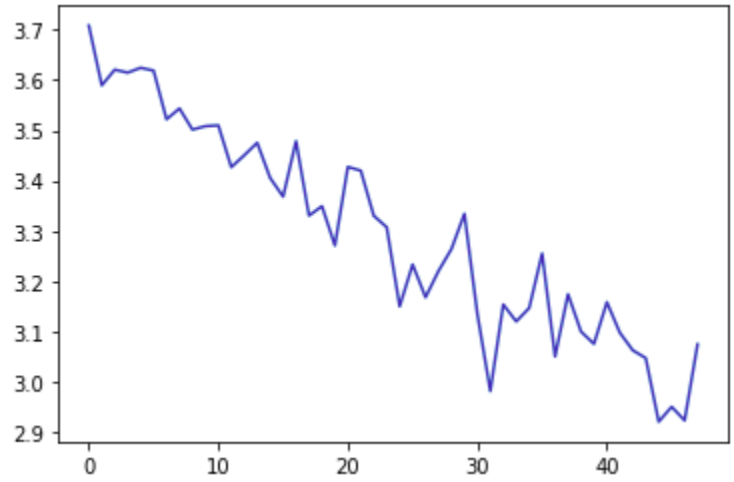
2. Adding new layers



*For all the plots the loss is plotted for every twelfth mini-batch.

After adding a few more convolutional layers and the fully connected layers as given in the network above the error plot was observed to train faster with the new network.

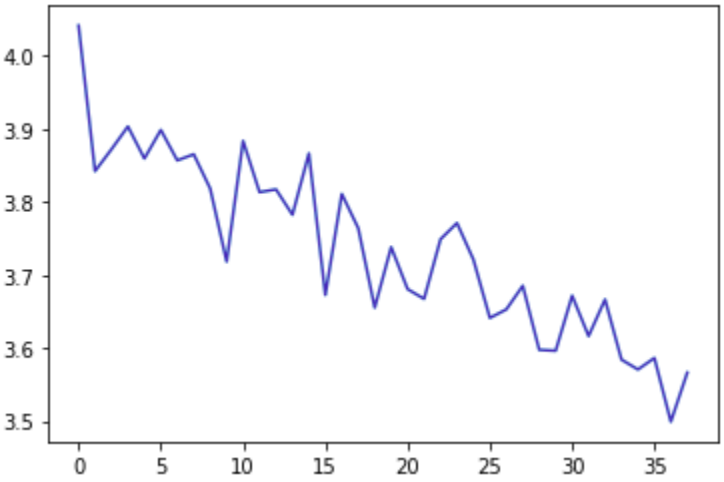
3. With Batch Norm



Batch normalization helped training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This is effected by stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

Batch normalization was applied for all the convolutional as well as the fully connected layers.

4. With Dropout and Batch Norm

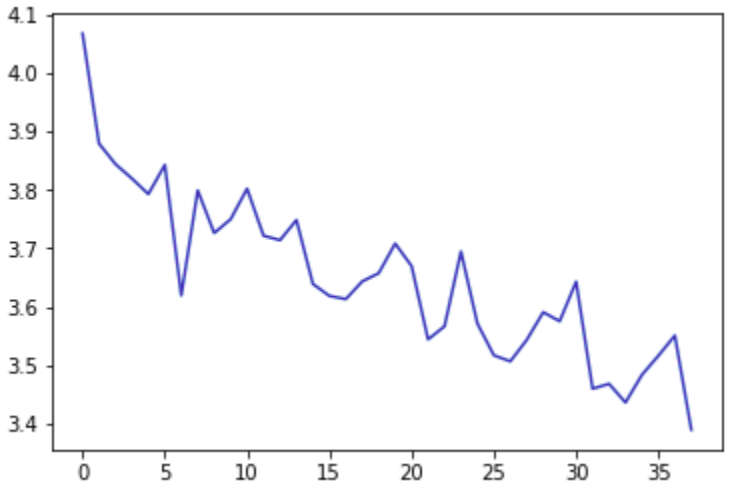


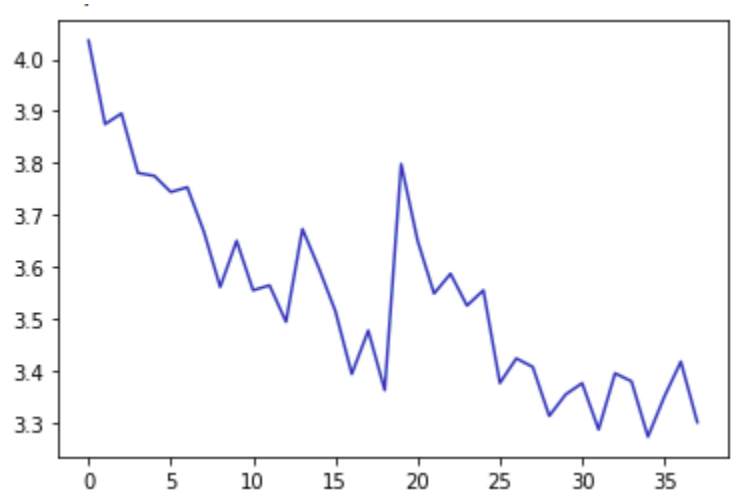
Dropout was implemented with batch normalization. For the used neural network architecture, and after two epochs using dropout did not make significant difference in reducing the loss.

5. Different activation functions at the end

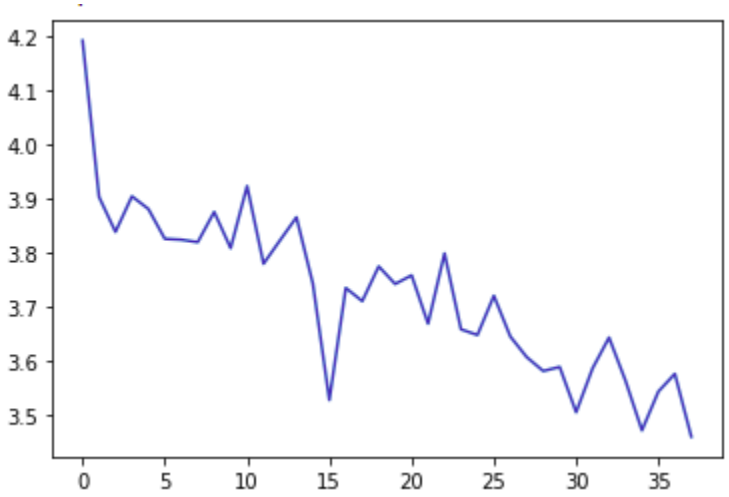
Trying through the different activation functions, not much difference was observed over multiple mini-batches. I could have observed it if I had trained it for the 5 epochs as the training data is very large.

6. Different pooling strategies





7. Basic Augmentation like Rotation, Translation, Color Change



Not much change was observed in the change of loss over training multiple mini-batches by applying different transformations. This would help when we have less training data available.