COL780
Assignment 4

Avani Jain
2020MT10792

Contents

The value of Y is 0.

Since the data was small, the CNN part was not fine-tuned due to overfitting concerns. Also, as the data is similar to the original data i.e, ImageNet, higher-level features in the CNN must be relevant to this dataset. Hence, just a linear classifier is trained on the CNN codes.

1 Final Results

```
Test Batch number: 000, Test: Loss: 0.0958, Accuracy: 1.0000
Test Batch number: 001, Test: Loss: 0.0921, Accuracy: 1.0000
Test Batch number: 002, Test: Loss: 0.0528, Accuracy: 1.0000
Test accuracy : 1.0
```

The accuracies obtained are as shown above. For details regarding each step, refer to the next section.

- Data Augmentation
- 2 connected layers
- Dropout of 0.4
- L1/L2 regularisation
- Learning rate of $10^{-}4$
- Training for 3 epochs

2 Improvement Techniques

2.1 Regularisation

2.1.1 Data Augmentation

Data augmentation is definitely useful because it can improve the predictive accuracy and general performance of machine learning models by reducing the risk of over-fitting. Without it, the accuracy was coming out to be less than the optimal one obtained. It is implemented by randomly rotating and resizing the images in train data.

```
'train': transforms.Compose([
       transforms.RandomResizedCrop(size=256, scale=(0.8, 1.0)),
       transforms.RandomRotation(degrees=15),
       transforms.RandomHorizontalFlip(),
       transforms.CenterCrop(size=224),
       transforms.ToTensor(),
       transforms.Normalize([0.485, 0.456, 0.406],
                            [0.229, 0.224, 0.225])
   ])
```

2.1.2 L1/L2 regularization

Adding a term for L1/L2 regularization by adding the weight decay parameter in the optimiser, instead of using just a simple Adam optimiser increased the accuracy. As shown in the code below.

```
optimizer = optimizer = torch.optim.Adam(model.parameters(), lr = 1e-4, weight_decay = 1e-5)
```

2.1.3 Dropout

Using dropout helps in decorrelating the weights. And it indeed lead to an increase in accuracy. I tried 3 different values for the p-value - 0.3, 0.4, 0.5 and found highest accuracy on p = 0.4. The inplace is kept to be false.

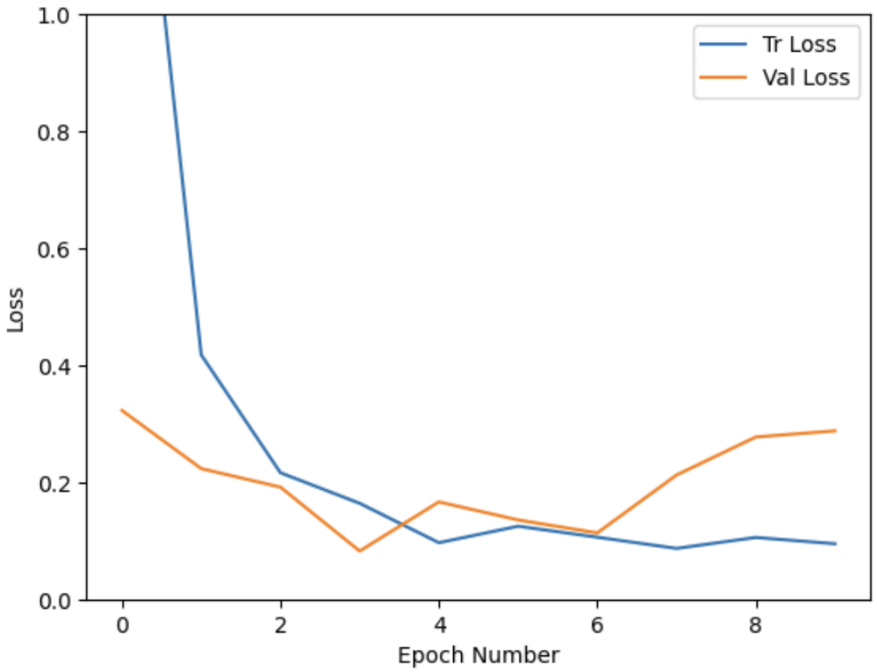
```
(classifier): Sequential(
   (0): Linear(in_features=25088, out_features=256, bias=True)
   (1): ReLU()
   (2): Dropout(p = 0.4, inplace = False)
   (3): Linear(in_features=256, out_features=25, bias=True)
   (4): LogSoftmax(dim=1)
 )
```

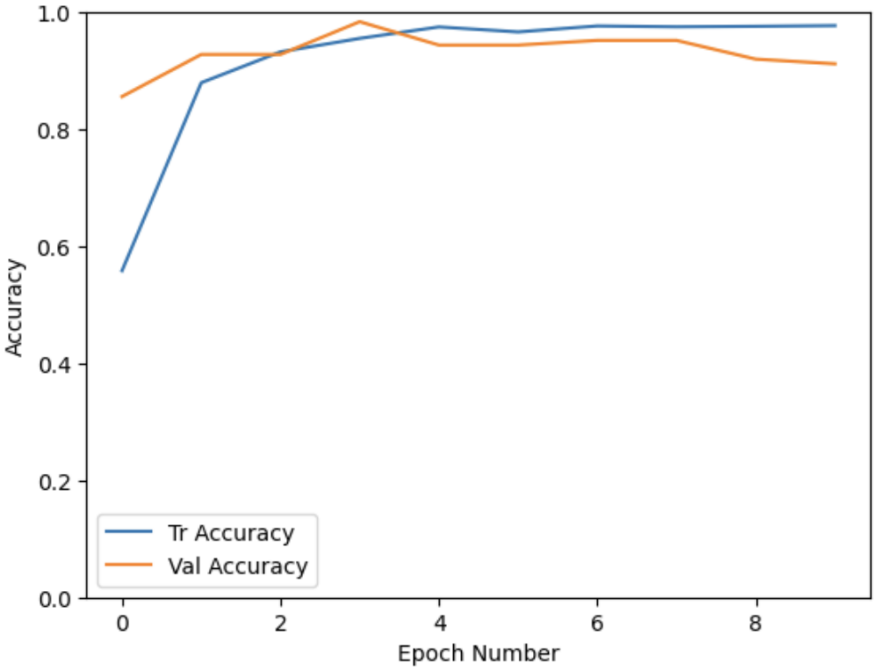
2.1.4 Number of epochs

Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model. Since, one epoch only took around 35-40 seconds to complete, I increased the number of epochs to 10 and used the graph between validation accuracy, training accuracy with the number of epochs and found that the optimal number of epochs to stop is 4 since after that the changes in accuracy become minimal and validation error starts increasing.
Refer to the images on next page

2.2 Architecture of Classifiers

Initially, I kept the same number of layers in classifiers as in original VGG16 model, but after reducing the number of linear layers to two, better results were obtained. Final architecture is as shown below, with the CNN part being same as in the original model.


Tr Loss
Val Loss
Loss
Epoch Number
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8


Tr Accuracy
Val Accuracy
Accuracy
Epoch Number
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (11): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): ReLU(inplace=True)
    (13): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (14): ReLU(inplace=True)
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (16): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace=True)
    (18): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (19): ReLU(inplace=True)
    (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=256, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.4, inplace=False)
    (3): Linear(in_features=256, out_features=25, bias=True)
    (4): LogSoftmax(dim=1)
  )
)
```

2.3 Learning rate

Learning rate was picked from 0.01, 0.001, 0.0001. 0.0001 gave the highest value of accuracy among these.

```
optimizer = optimizer = torch.optim.Adam(model.parameters(), lr = 1e-4, weight_decay = 1e-5)
```

2.4 Loss function

The loss function used is negative log likelihood loss.

2.5 Batch size

A batch size of 50 is used. Also tried with batch size 1, but it lead to poorer results.

2.6 Optimiser

Adam optimiser is used. Hence, no need for batch normalisation.