ELL793

Computer Vision

Prof. Brejesh Lall

Assignment 3:

CIFAR-10
Dataset Classification

Submitted by:
Raghav Gupta (2017EE10544)
Prakhar Kanchan (2017EE30543)

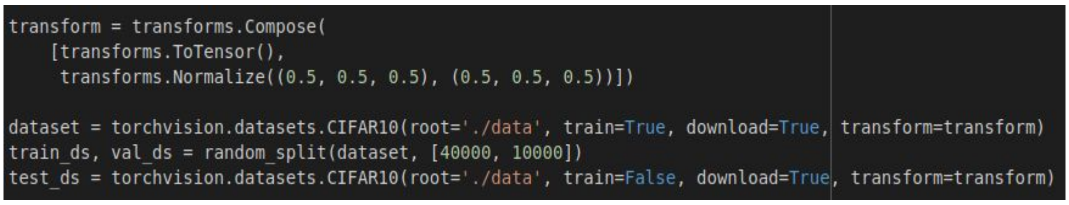
Part 1

Goal

Train a CNN network with ResNet-18 as a backbone from scratch with CIFAR-10

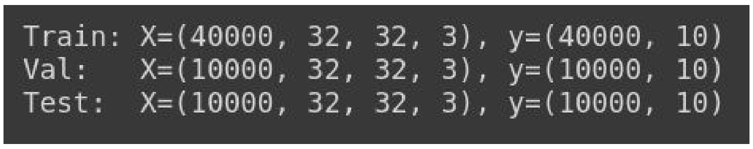
Procedure

- Loaded the CIFAR-10 dataset, created a validation split and applied normalization on the same lines as the previous problem.
- The cell shown below describes the process of importing the dataset and pre-processing it for training.



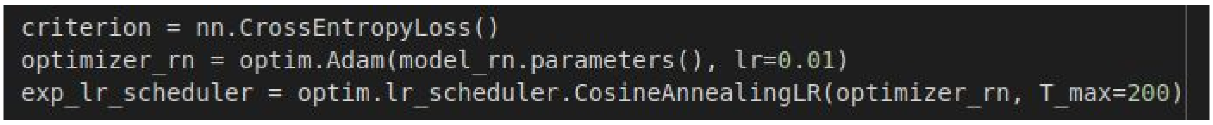
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
train_ds, val_ds = random_split(dataset, [40000, 10000])
test_ds = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
```



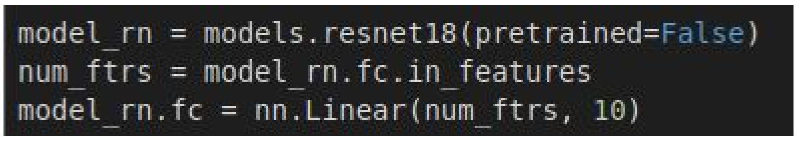
```
Train: X=(40000, 32, 32, 3), y=(40000, 10)
Val:   X=(10000, 32, 32, 3), y=(10000, 10)
Test:  X=(10000, 32, 32, 3), y=(10000, 10)
```

- The loss function chosen is Categorical Crossentropy, which is a standard loss function used for multi-class classification.
- Adam optimizer is used with a learning rate of 0.01.
- The learning rate scheduler used is CosineAnnealing.



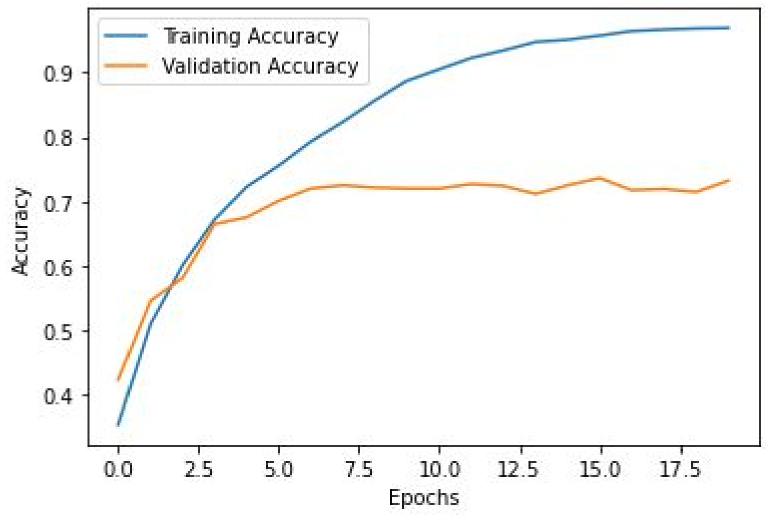
```
criterion = nn.CrossEntropyLoss()
optimizer_rn = optim.Adam(model_rn.parameters(), lr=0.01)
exp_lr_scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer_rn, T_max=200)
```

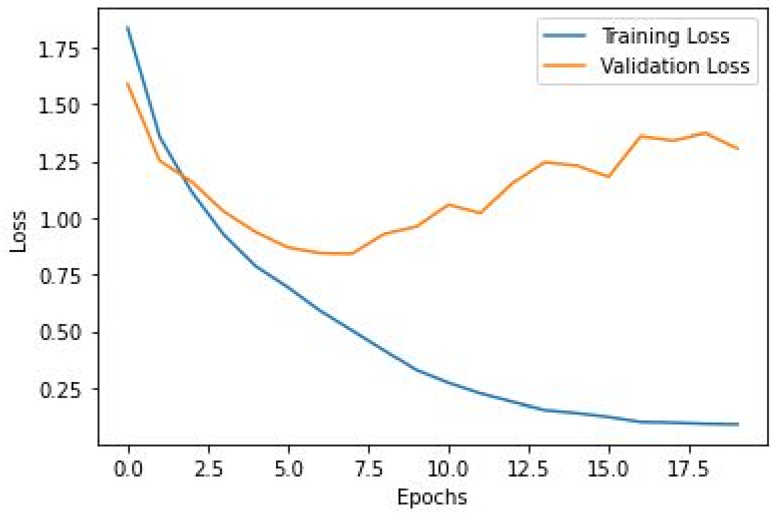
- Resnet-18 model was used and trained from scratch for 20 epochs with batch size of 128.



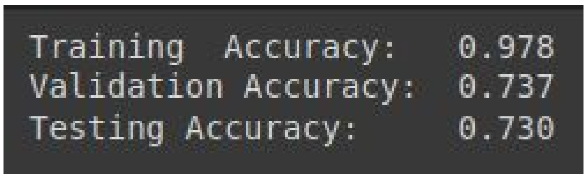
```
model_rn = models.resnet18(pretrained=False)
num_ftrs = model_rn.fc.in_features
model_rn.fc = nn.Linear(num_ftrs, 10)
```

- Plots below show the Accuracies and Loss over epochs.





- Final performance of the model is given below:



```
Training  Accuracy:   0.978
Validation Accuracy:  0.737
Testing Accuracy:     0.730
```

- The model took 6m 4s to train on Google Colab GPU.

Part 2

Goal

Initialize the ResNet-18 network with pretrained weights from ImageNet and then try to use these weights to improve the training for the CIFAR-10 dataset.

Transfer Learning:

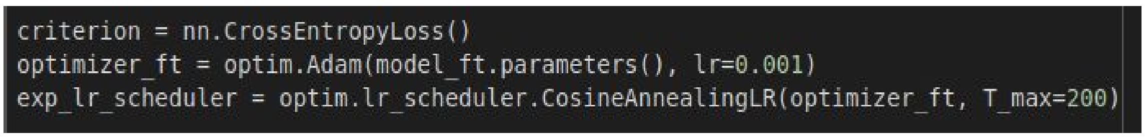
We explore 2 different kinds of weight initialization techniques:

- Finetuning the CNN: Instead of random initialization, we initialize the network with a pretrained network (ImageNet). Rest of the training looks as usual.

- CNN as fixed feature extractor: Here, we will freeze the weights for all of the network except that of the final fully connected layer. This last fully connected layer is replaced with a new one with random weights and only this layer is trained.

Finetuning the CNN:

- Loaded the CIFAR-10 dataset, created a validation split, and applied normalization on the same lines as the previous problem.
- The loss function chosen is Categorical Crossentropy, which is a standard loss function used for multi-class classification.
- Adam optimizer is used with a learning rate of 0.001.
- Learning rate scheduler used is CosineAnnealing.

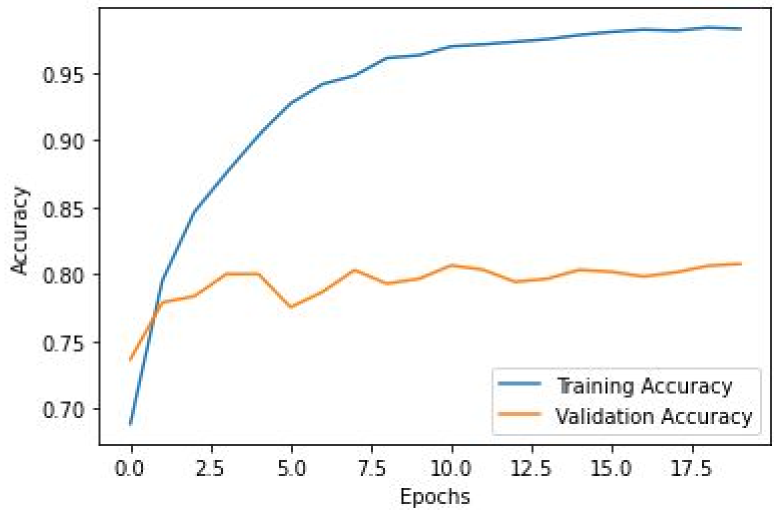


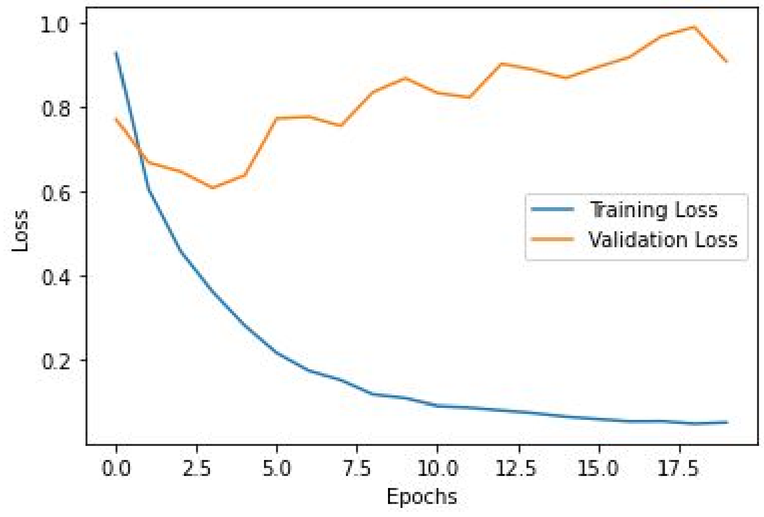
```
criterion = nn.CrossEntropyLoss()
optimizer_ft = optim.Adam(model_ft.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer_ft, T_max=200)
```

- Resnet-18 model was used which was pretrained on ImageNet dataset. The model was further trained for 20 epochs with batch size of 128.

```
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 10)
```

- Plots below show the Accuracies and Loss over epochs.





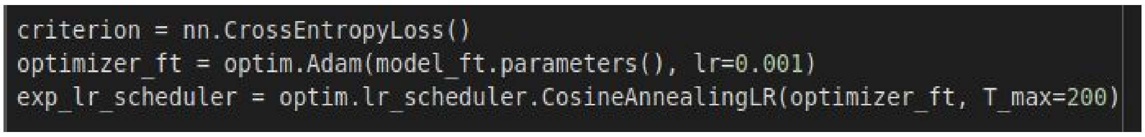
- Final performance of the model is given below:

```
Training  Accuracy:   0.992
Validation Accuracy:  0.808
Testing Accuracy:     0.805
```

- The model took <u>6m 2s</u> to train on Google Colab GPU.

CNN as fixed feature extractor:

- Loaded the CIFAR-10 dataset, created a validation split, and applied normalization on the same lines as the previous problem.
- The loss function chosen is Categorical Crossentropy, which is a standard loss function used for multi-class classification.
- Adam optimizer is used with a learning rate of 0.001.
- Learning rate scheduler used is CosineAnnealing.

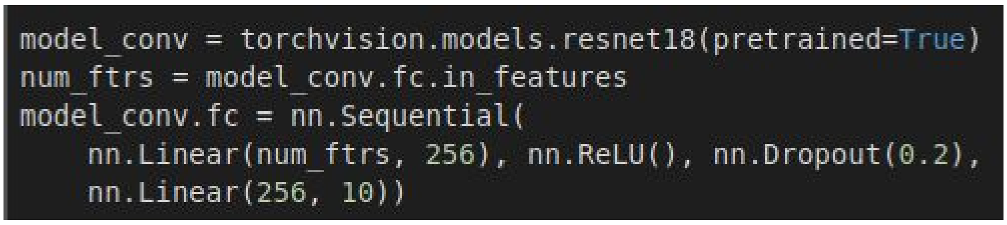


```
criterion = nn.CrossEntropyLoss()
optimizer_ft = optim.Adam(model_ft.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer_ft, T_max=200)
```

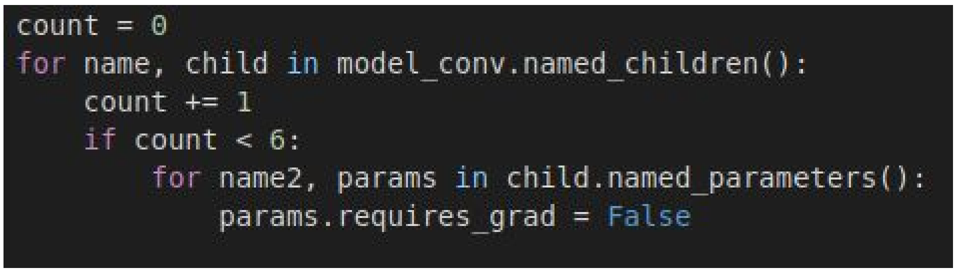
- Resnet-18 model was used which was pretrained on the ImageNet dataset.
- The layers were freezed in 2 different configurations and the experiments are shown as follows:

1. Model A:

    - The first 6 Conv layers were freezed and the last fully connected layer is replaced with a new one with random weights and only this layer and the unfreezed layers are trained.

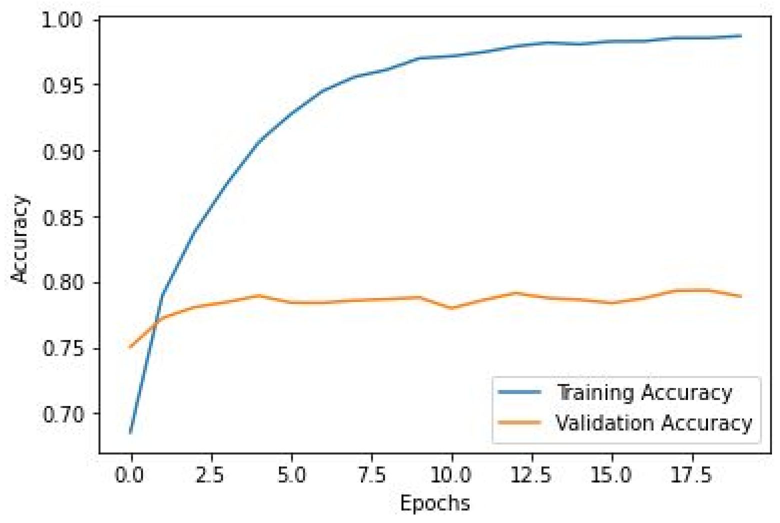


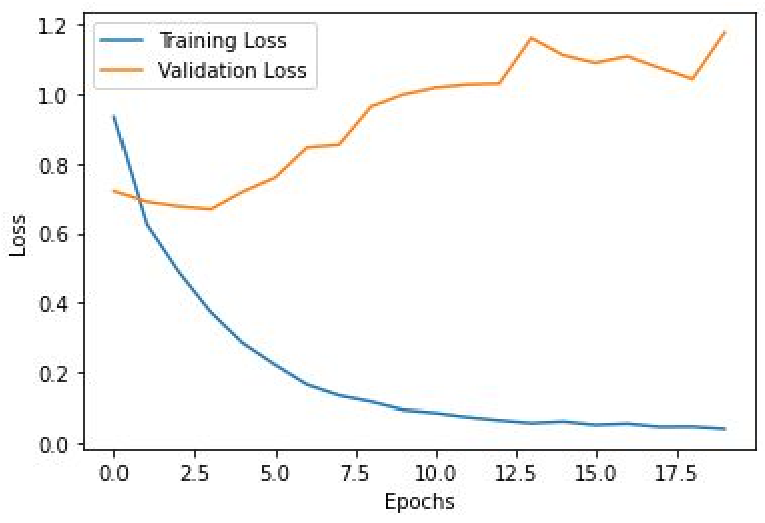
```
model_conv = torchvision.models.resnet18(pretrained=True)
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Sequential(
    nn.Linear(num_ftrs, 256), nn.ReLU(), nn.Dropout(0.2),
    nn.Linear(256, 10))
```



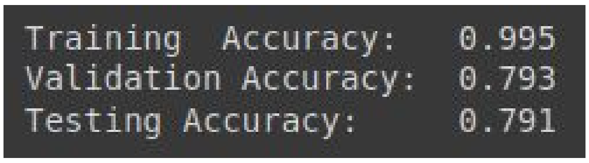
```
count = 0
for name, child in model_conv.named_children():
    count += 1
    if count < 6:
        for name2, params in child.named_parameters():
            params.requires_grad = False
```

- Plots below show the Accuracies and Loss over epochs.





- Final performance of the model is given below:

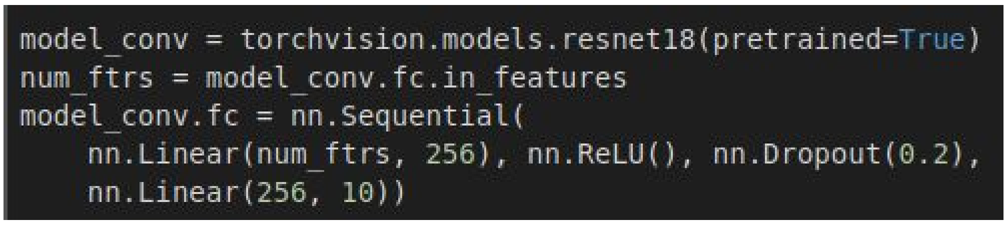


```
Training  Accuracy:   0.995
Validation Accuracy:  0.793
Testing Accuracy:     0.791
```

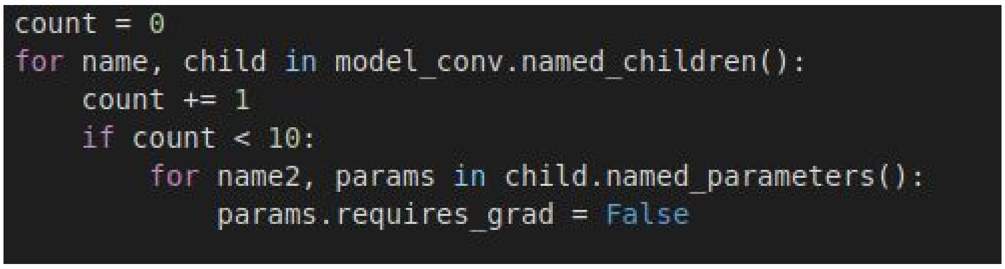
- The model took <u>5m 21s</u> to train on Google Colab GPU.

2. Model B:

- The first 10 Conv layers were freezed and the last fully connected layer is replaced with a new sequential layer (as shown below) with random weights and only this layer and the unfreezed layers are trained.

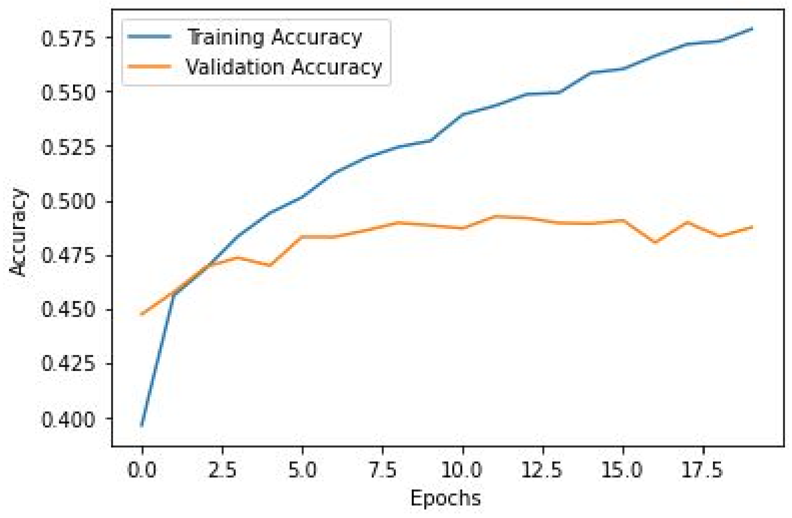


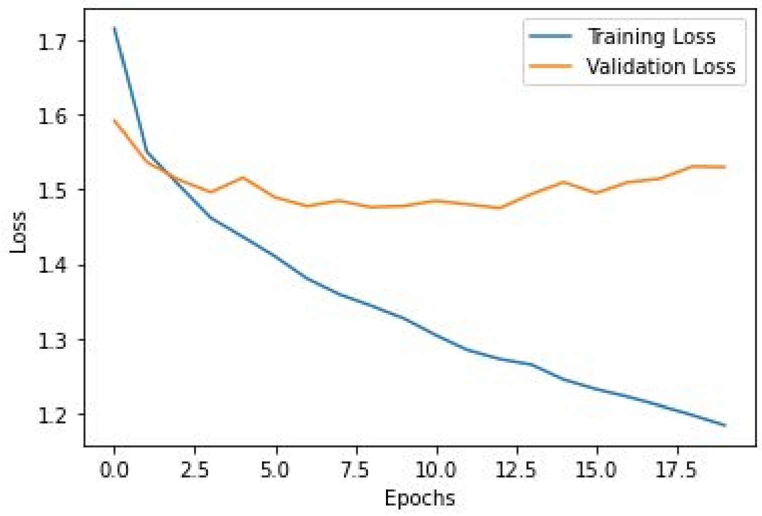
```
model_conv = torchvision.models.resnet18(pretrained=True)
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Sequential(
    nn.Linear(num_ftrs, 256), nn.ReLU(), nn.Dropout(0.2),
    nn.Linear(256, 10))
```



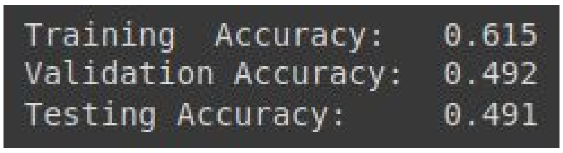
```
count = 0
for name, child in model_conv.named_children():
    count += 1
    if count < 10:
        for name2, params in child.named_parameters():
            params.requires_grad = False
```

- Plots below show the Accuracies and Loss over epochs.





- Final performance of the model is given below:

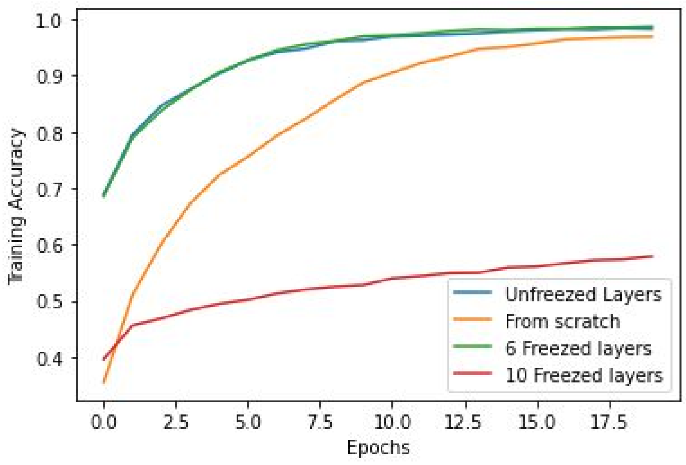


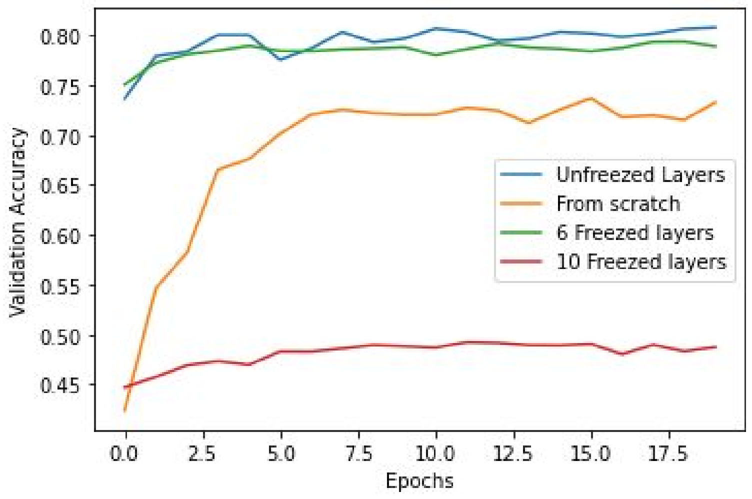
```
Training  Accuracy:   0.615
Validation Accuracy:  0.492
Testing Accuracy:     0.491
```

- The model took <u>3m 45s</u> to train on Google Colab GPU.

Comparison

- The following plot shows the comparison between different Transfer learning configurations and base Resnet18 model.





- The following table compares the training time taken and final Test accuracy of above models for 20 epochs with a batch size of 128 .

Model	Test Accuracy	Time Taken
Resnet18 from scratch	0.73	6m 4s
Resnet18 pretrained (unfreezed)	0.805	6m 2s
Resnet18 pretrained (6 freezed layers)	0.791	5m 21s
Resnet18 pretrained (10 freezed layers)	0.491	3m 45s

- The Resnet18 pretrained (unfreezed) model gives the best Testing Accuracy but it takes more time.

- The Resnet18 pretrained (10 freezed layers) takes the least time but gives significantly poor performance.

- Therefore in terms of both Test accuracy and time taken to train, Resnet18 pretrained (10 freezed layers) performs the best.

<u>Part 3</u>

Goal

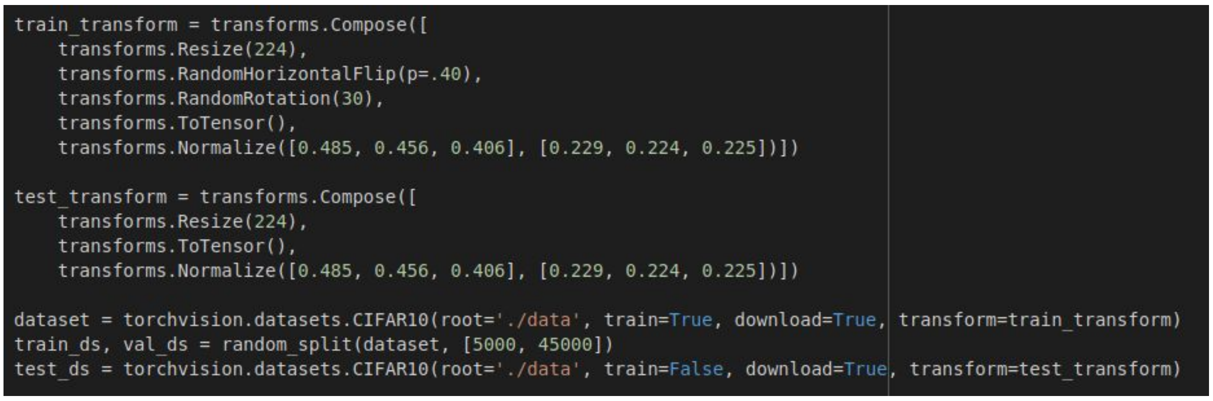
Train the network from scratch with Tiny-CIFAR-10 dataset using data augmentation techniques to improve the performance.

Procedure

- Loaded the CIFAR-10 dataset, created a split to create mini CIFAR-10 dataset.

- Data Augmentation techniques used:
    - Resize
    - Random Horizontal Flip
    - Random Rotation

- The following code snippet shows the mini CIFAR-10 dataset creation and data augmentation transformation.

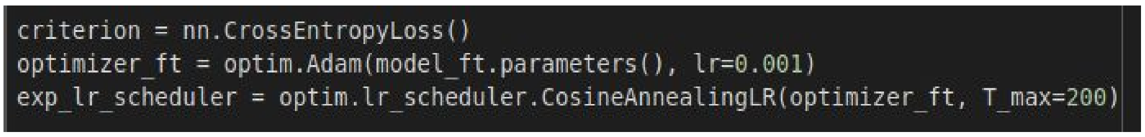


```
train_transform = transforms.Compose([
    transforms.Resize(224),
    transforms.RandomHorizontalFlip(p=.40),
    transforms.RandomRotation(30),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])

test_transform = transforms.Compose([
    transforms.Resize(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])

dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=train_transform)
train_ds, val_ds = random_split(dataset, [5000, 45000])
test_ds = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=test_transform)
```

- The loss function chosen is <u>Categorical Crossentropy</u>, which is a standard loss function used for multi-class classification.
- <u>Adam optimizer</u> is used with a learning rate of 0.001.
- Learning rate scheduler used is CosineAnnealing.

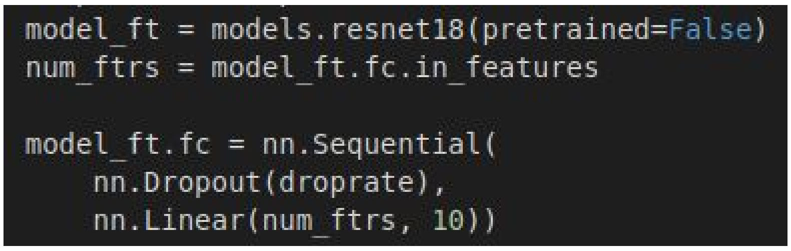


```
criterion = nn.CrossEntropyLoss()
optimizer_ft = optim.Adam(model_ft.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer_ft, T_max=200)
```

- Resnet-18 model was used and trained from scratch.

Introducing Dropout:

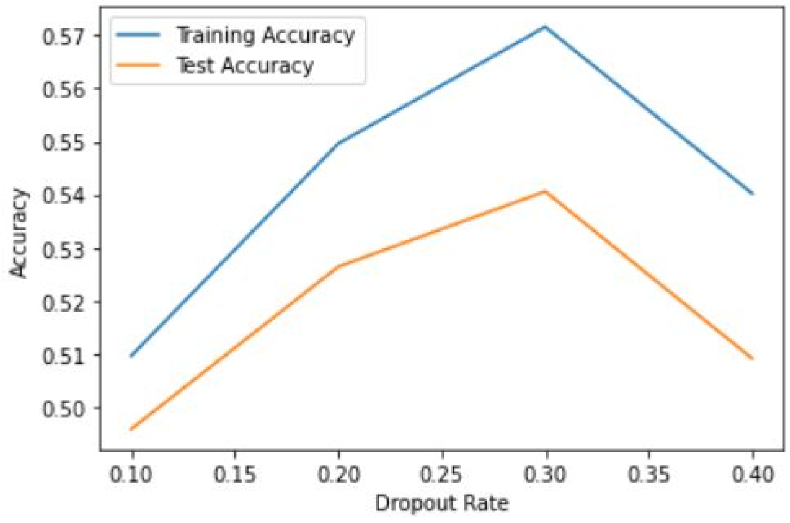
- To introduce regularization a Dropout layer was added as shown below:



```
model_ft = models.resnet18(pretrained=False)
num_ftrs = model_ft.fc.in_features

model_ft.fc = nn.Sequential(
    nn.Dropout(droprate),
    nn.Linear(num_ftrs, 10))
```

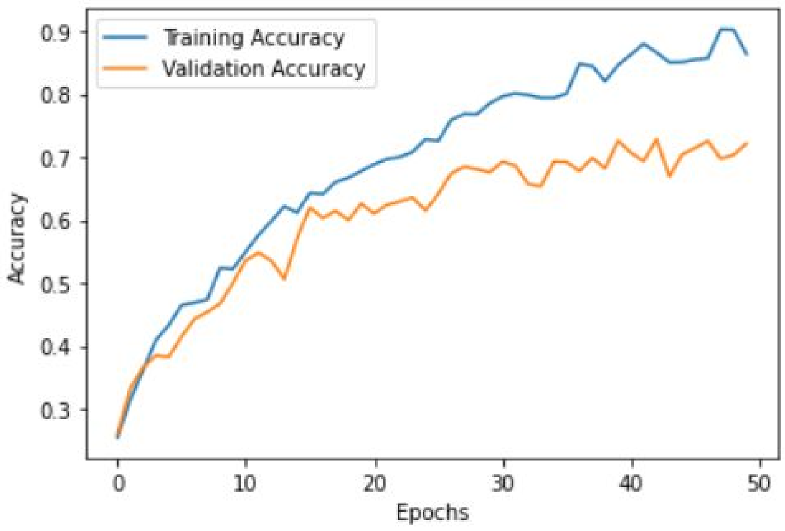
- The dropout rate was varied from 0.1 to 0.4 and Training and Testing Accuracy for the respective models is given by the graph below.

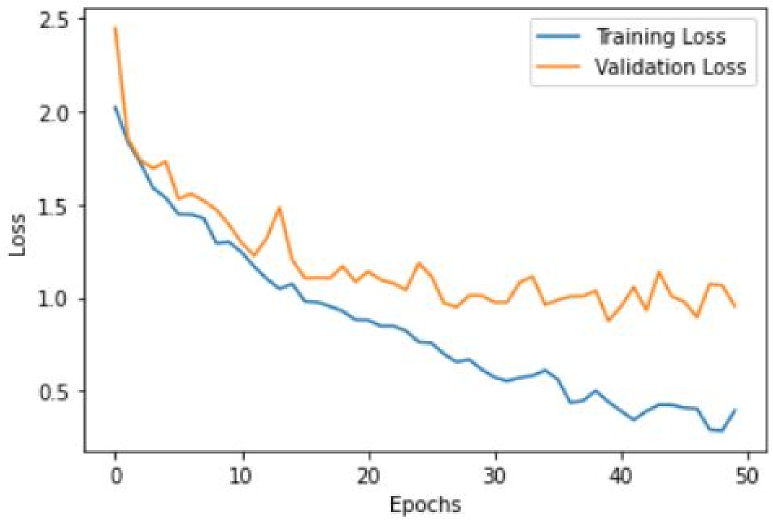


- Dropout rate of <u>0.3</u> performed the best as shown by the graph.

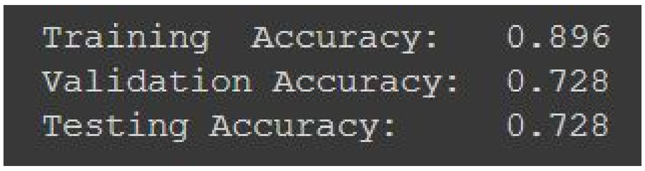
Final Model:

- The final model with Dropout rate of 0.3 was trained for 50 epochs with batch size of 128.
- Plots below show the Accuracies and Loss over epochs.





- Final performance of the model is given below.

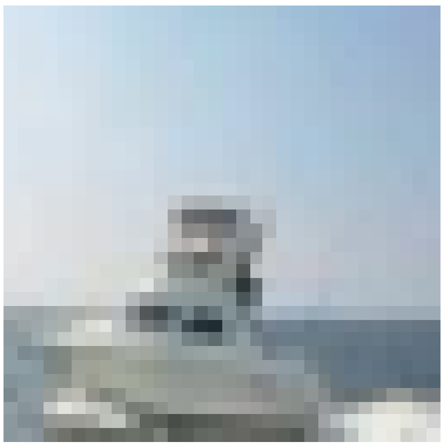


```
Training  Accuracy:   0.896
Validation Accuracy:  0.728
Testing Accuracy:     0.728
```

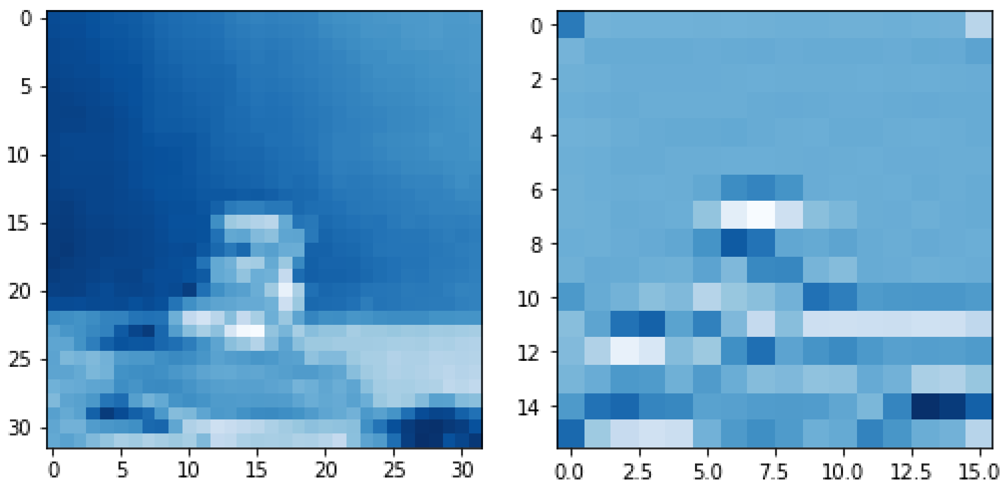
- The model took 23m 51s to train on Google Colab GPU.

<u>Part 4</u>

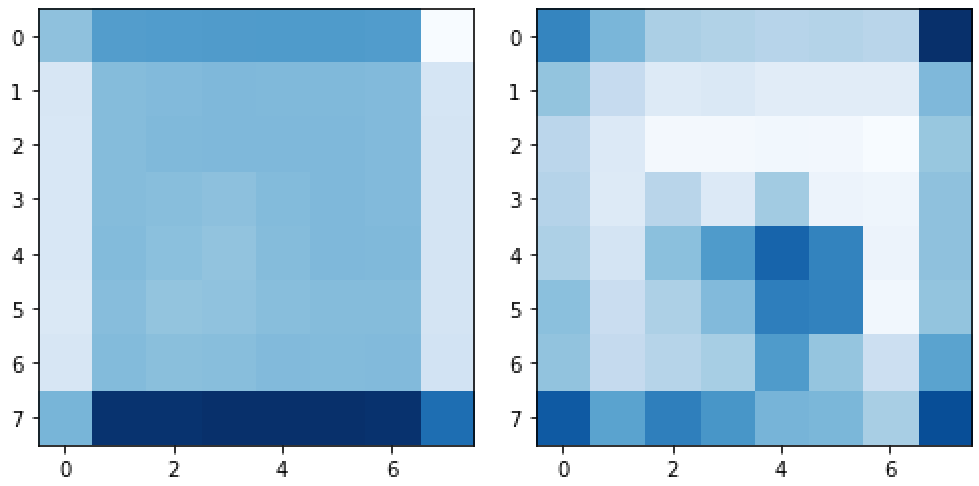
- We visualise various activation layers of ResNet-18 at different levels.
- Activations are basically the pixel values that have high correlation with correctly identifying what an object is.
- This basically gives us a way to visualize what each layer is doing and what it's trying to focus on within a certain image that is passed for classification.
- The figure of a ship shown below to make the analysis.



- These are two of the initial level filters (*conv1_conv* and *conv2_block1_1_conv*) mapped for the ship given as an input. As can be seen, it gives a rough idea of the shape of the ship and the contents on the image around it like the sky and sea.

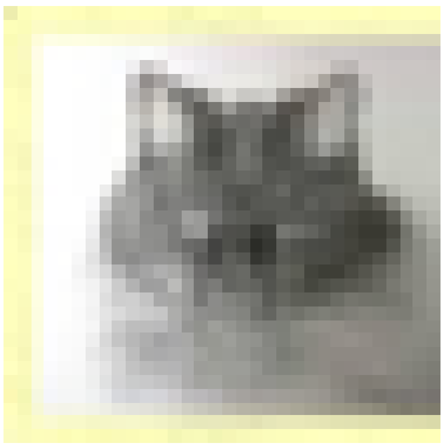


- The activation maps change significantly for later layers. The following plot corresponds to the 25th layer *(conv2_block2_3_conv)* of the model.
- As can be seen, the feature maps have considerably become less complicated than their initial counterparts and it is difficult quantitatively to understand what they try to depict.

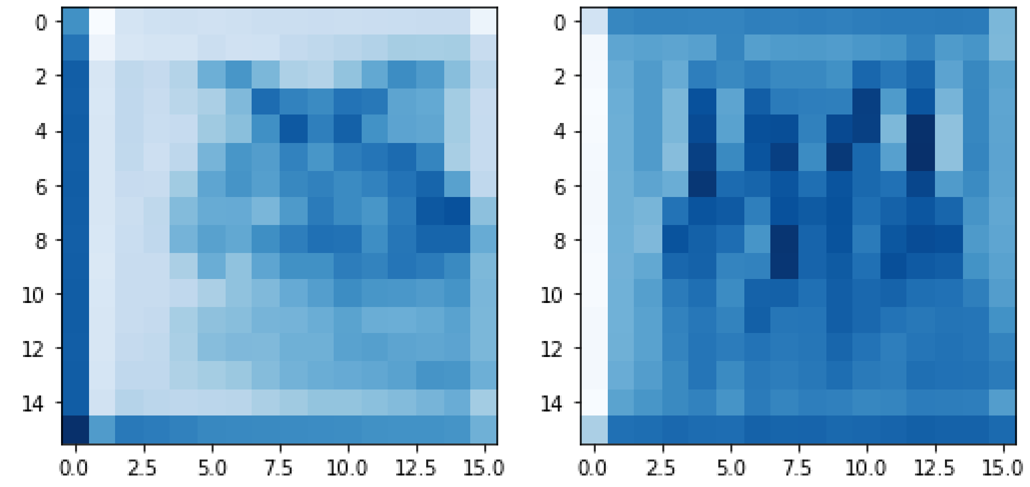


- Since CIFAR-10 has its images in a 32x32 pixel quality, we don't get the most accurate activation visualization.
- This experiment can be run again with images of a higher quality (more pixels) to see a better representation of how the activations look for a particular image.

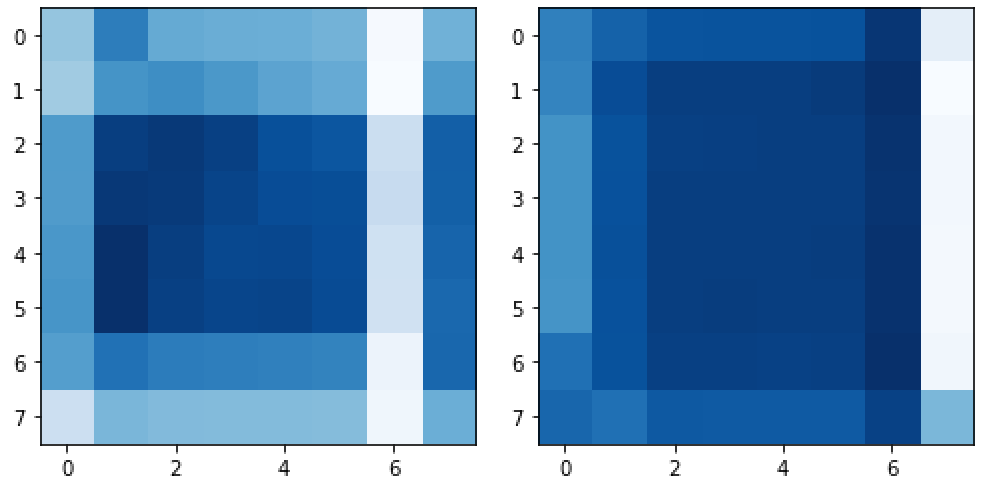
- Another example containing an image of a cat has been discussed below.



- First 2 layers have been shown below.



- 25th layer.


0
1
2
3
4
5
6
7
0
2
4
6