In [0]:
```
#Deep Learning Assignment 1 - Part 2
#Joshua Abraham - jma672
#Ayan Agrawal - aka398
```

In [0]:
```
#installation of the pytorch library
from os import path
from wheel.pep425tags import get_abbr_impl, get_impl_ver, get_abi_tag
platform = '{}{}-{}'.format(get_abbr_impl(), get_impl_ver(), get_abi_tag())

accelerator = 'cu80' if path.exists('/opt/bin/nvidia-smi') else 'cpu'

!pip3 install -q http://download.pytorch.org/whl/{accelerator}/torch-0.4.0-{pl
atform}-linux_x86_64.whl torchvision
```

In [0]:
```
import numpy as np
import torch
import torchvision
import torchvision.transforms as transforms
```

In [4]:
```
#downloading the dataset
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((
0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train_set = torchvision.datasets.CIFAR10(root='./cifardata', train=True, downl
oad=True, transform=transform)

test_set = torchvision.datasets.CIFAR10(root='./cifardata', train=False, downl
oad=True, transform=transform)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [0]:
```
#dividinng the train set into 9:1 to perform training and validation
#based on the validation loss we are tuning the hyperparameters
from torch.utils.data.sampler import SubsetRandomSampler

train_samples = 45000
trainData = SubsetRandomSampler(np.arange(train_samples, dtype=np.int64))

validation_samples = 5000
validationData = SubsetRandomSampler(np.arange(train_samples, train_samples +
validation_samples, dtype=np.int64))

test_samples = 10000
testData = SubsetRandomSampler(np.arange(test_samples, dtype=np.int64))
```

In [0]:
```
from torch.autograd import Variable
import torch.nn.functional as F
import torch.nn as nn

class CNN(torch.nn.Module):


    def __init__(self):
        super(CNN, self).__init__()
        ####################Architecture##################
        #original_image --> conv_1 -->max_pool-->conv_2-->max_pool-->conv3-->F
C Layer1-->FC2--FC2
        ####################################################

        #original image size is 32*32*3
        #first layer of convolution
        #in_channel=3,out_channel=36,#kernel_size=3*3,stride=1,padding=1
        #with just having one convolution layer he accuracy came out to be 66%
        #thus we increased the convolutional layers to 3 to match the validati
on loss with the train loss approximately
        self.conv1 = nn.Conv2d(3, 36, 3, 1, padding=1) #66%
        #max pool layer of 2*2 with stride=2
        self.pool = nn.MaxPool2d(2, 2)
        #in_channel=36,out_channel=144,#kernel_size=3*3,stride=1,padding=1
        self.conv2 = nn.Conv2d(36, 144, 3,1,padding=1)
        #in_channel=144,out_channel=288,#kernel_size=3*3,stride=1,padding=1
        self.conv3 = nn.Conv2d(144, 288, 3,1,padding=1)
        #the image size input to the fully connected layer is now 8*8 after co
nv and max_pool
        #288 is the number of input hidden units
        self.fc1 = nn.Linear(288 * 8 * 8, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)


    def forward(self, x):
        x = F.relu(self.conv1(x))
        x= self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        #x = self.pool()
        #flatten the input input image
        x = x.view(-1, 288 * 8 * 8)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

In [0]:
```
#def outputSize(in_size, kernel_size, stride, padding):

#  output = int((in_size - kernel_size + 2*(padding)) / stride) + 1

#  return(output)
```

In [0]:
```
def get_train_loader(batch_size):
    train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_siz
e,
                                               sampler=trainData, num_workers=2)
    return(train_loader)
```

In [0]:
```
val_loader = torch.utils.data.DataLoader(train_set, batch_size=128, sampler=va
lidationData, num_workers=2)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=4, sampler=test
Data, num_workers=2)
```

In [0]:
```
import torch.optim as optim

def LossOptimizerFunction(net, learning_rate=0.001):
    loss = torch.nn.CrossEntropyLoss()
    optimizer = optim.Adam(net.parameters(), lr=learning_rate)
    return(loss, optimizer)
```

In [0]:
```
def train(net, batch_size, learning_rate,epochs):
    #Load the training data
    train_loader = get_train_loader(batch_size)
    n_batches = len(train_loader)

    #Call the loss and optimizer functions
    loss, optimizer = LossOptimizerFunction(net, learning_rate)
    for epoch in range(epochs):

        running_loss = 0.0
        print_every = n_batches // 10

        for i, data in enumerate(train_loader, 0):

            image_input, labels = data

            #Make a variable object wrapper
            image_input, labels = Variable(image_input), Variable(labels)

            #initialize the gradient parameters to zero
            optimizer.zero_grad()

            #Forward propogation
            outputs = net(image_input)
            #calculate the loss
            loss_size = loss(outputs, labels)
            #Backward propogation
            loss_size.backward()
            #Optimizer usage
            optimizer.step()


            running_loss += loss_size.data[0]

            #Print every 10th batch of an epoch
            if (i + 1) % (print_every + 1) == 0:
                print("Epoch {}, Train_Loss: {:.2f}".format(
                        epoch+1, running_loss / print_every))
                #Reset running loss and time
                running_loss = 0.0

        #Calculate the validation loss
        total_validation_loss = 0
        for image_inputs, labels in val_loader:

            #Make a variable object wrapper
            image_inputs, labels = Variable(image_inputs), Variable(labels)

            #To calculate the loss on validation we require just the Forward p
ropogation
            val_outputs = net(image_inputs)
            val_loss_size = loss(val_outputs, labels)
            total_validation_loss += val_loss_size.data[0]

        print("Validation loss = {:.2f}".format(total_validation_loss / len(va
l_loader)))
```

In [12]:
```
CNN = CNN()
train(CNN, 32, 0.001,5)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:33: UserWarning:
invalid index of a 0-dim tensor. This will be an error in PyTorch 0.5. Use te
nsor.item() to convert a 0-dim tensor to a Python number
```

```
Epoch 1, Train_Loss: 2.00
Epoch 1, Train_Loss: 1.65
Epoch 1, Train_Loss: 1.46
Epoch 1, Train_Loss: 1.36
Epoch 1, Train_Loss: 1.29
Epoch 1, Train_Loss: 1.25
Epoch 1, Train_Loss: 1.21
Epoch 1, Train_Loss: 1.16
Epoch 1, Train_Loss: 1.11
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:52: UserWarning:
invalid index of a 0-dim tensor. This will be an error in PyTorch 0.5. Use te
nsor.item() to convert a 0-dim tensor to a Python number
```

```
Validation loss = 1.02
Epoch 2, Train_Loss: 0.99
Epoch 2, Train_Loss: 0.97
Epoch 2, Train_Loss: 0.96
Epoch 2, Train_Loss: 0.95
Epoch 2, Train_Loss: 0.91
Epoch 2, Train_Loss: 0.92
Epoch 2, Train_Loss: 0.92
Epoch 2, Train_Loss: 0.85
Epoch 2, Train_Loss: 0.86
Validation loss = 0.86
Epoch 3, Train_Loss: 0.71
Epoch 3, Train_Loss: 0.72
Epoch 3, Train_Loss: 0.74
Epoch 3, Train_Loss: 0.71
Epoch 3, Train_Loss: 0.69
Epoch 3, Train_Loss: 0.74
Epoch 3, Train_Loss: 0.72
Epoch 3, Train_Loss: 0.68
Epoch 3, Train_Loss: 0.71
Validation loss = 0.75
Epoch 4, Train_Loss: 0.55
Epoch 4, Train_Loss: 0.57
Epoch 4, Train_Loss: 0.55
Epoch 4, Train_Loss: 0.55
Epoch 4, Train_Loss: 0.54
Epoch 4, Train_Loss: 0.56
Epoch 4, Train_Loss: 0.60
Epoch 4, Train_Loss: 0.56
Epoch 4, Train_Loss: 0.56
Validation loss = 0.70
Epoch 5, Train_Loss: 0.37
Epoch 5, Train_Loss: 0.41
Epoch 5, Train_Loss: 0.41
Epoch 5, Train_Loss: 0.42
Epoch 5, Train_Loss: 0.42
Epoch 5, Train_Loss: 0.43
Epoch 5, Train_Loss: 0.44
Epoch 5, Train_Loss: 0.42
Epoch 5, Train_Loss: 0.44
Validation loss = 0.77
```

In [30]:
```
correct = 0
total = 0
predicted_labels = []
#for prediction there is no need of storing the gradients
with torch.no_grad():
    for data in test_loader:
        images, labels = data
        outputs = CNN(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        predicted_labels.append(predicted)
print('Accuracy == %d %%' % (100 * correct / total))
```

```
Accuracy == 73 %
```

In [0]:
```
def savepredictions(filename,y):
  np.save(filename,y)
predicted_values = []
for batch_labels in predicted_labels:
  for images in batch_labels:
    predicted_values.append(images)


savepredictions("ans2-aka398.npy",predicted_values)
```

In [42]:
```
data = np.load("ans2-aka398.npy")
print(data.shape)
```

```
(10000,)
```

1) We started with one convolutional layer, 3 fully connected layer and number of epoch as 2. We kept the optimizer as "SGD". We got an accuracy of 7%. Also, we observed that there was a marginable differnce between the validation loss and the train loss. So we decided to increase the number of epochs considering that it requires more time to train.

2) We increased the number of epochs to 5. Our accuracy came out to be 54%. We observed that the validation loss was 1.24 and the train loss was 0.74. Such a large difference indicated that our model was overfitting the train data. We dicided to to go with other optimizer as "Adam" and removed the momentum parameter from the argument.

3) Rest all we kept as it is and started training. We got an accuracy of 62%. Now the difference between the train loss and validation loss reduced but it was still not satisfactory.

4) But we noticed that using Adam Optimizer was beneficial since it is a combination of both Momentum and RMS prop.

5) Then we increased the convolutional layer to 3. Change the number of filters, added the padding argument. We observed that at epoch number of approximately 3 and 4 we were getting matching values of train loss and validation loss. So we decided to stay with this hyperparameters.