```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torchvision.utils as utils
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import seaborn as sn
import pandas as pd
import numpy as np
import time
```

```
#Using GPU if exists
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
device
```

```
device(type='cuda')
```

```
# Load Data
CIFAR10_dataset = datasets.CIFAR10(root='dataset/CIFAR10',
                                         train=True,
                                         transform=transforms.ToTensor(), download=True)
```

```
transform_images = transforms.Compose(
               [transforms.RandomHorizontalFlip(),
                transforms.RandomCrop(size=32, padding=[0, 2, 3, 4]),
                transforms.ToTensor()])
CIFAR10_dataset_augmented = datasets.CIFAR10(root='dataset/CIFAR10',
                                         train=True,
                                         transform=transform_images, download=True)
```

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

```
CIFAR10_dataset
```

```
Dataset CIFAR10
    Number of datapoints: 50000
    Root location: dataset/CIFAR10
    Split: Train
    StandardTransform
Transform: ToTensor()
```

```
CIFAR10_dataset_augmented
```

```
Dataset CIFAR10
    Number of datapoints: 50000
    Root location: dataset/CIFAR10
    Split: Train
    StandardTransform
Transform: Compose(
               RandomHorizontalFlip(p=0.5)
               RandomCrop(size=(32, 32), padding=[0, 2, 3, 4])
               ToTensor()
           )
```

```
@torch.no_grad()
def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))

@torch.no_grad()
def evaluate(model, data_loader, testOrValidation):
    model.eval()
    outputs = [model.testOrValidation_step(batch, testOrValidation) for batch in data_load
    return model.at_testOrValidation_epoch_end(outputs,testOrValidation)

class Utils(nn.Module):
    def training_step(self, batch):
        images, true_labels = batch
        images = images.to(device=device)
        true_labels = true_labels.to(device=device)
        pred_labels = self(images)
        loss = F.cross_entropy(pred_labels, true_labels)
        accur = accuracy(pred_labels,true_labels)
        return loss,accur

    def testOrValidation_step(self, batch, testOrValidation):
        images, true_labels = batch
        images = images.to(device=device)
        true_labels = true_labels.to(device=device)
        pred_labels = self(images)
        loss = F.cross_entropy(pred_labels, true_labels)
        accur= accuracy(pred_labels, true_labels)
        return {testOrValidation +'_loss': loss.detach(), testOrValidation+'_accuracy': ac

    def at_testOrValidation_epoch_end(self, outputs, testOrValidation):
        batch_losses = [x[testOrValidation+'_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()
        batch_accuracies = [x[testOrValidation+'_accuracy'] for x in outputs]
        epoch_accuracy = torch.stack(batch_accuracies).mean()
        return {testOrValidation +'_loss': epoch_loss.item(), testOrValidation+'_accuracy'

    def at_epoch_end(self, epoch, result):
        print("Epoch :",epoch + 1)
        print(f'Train Accuracy:{result["train_accuracy"]*100:.2f}% Validation Accuracy:{re
        print(f'Train Loss:{result["train_loss"]:.4f} Validation Loss:{result["validation_
def conv_block_1(in_f, out_f, enable_BN, enable_dropOut):
    return nn.Sequential(
        nn.Conv2d(in_f, out_f, kernel_size=3, stride=1, padding=1),
        nn.ReLU()
    )



def conv_block_2(in_f, out_f, enable_BN, enable_dropOut):
    if(enable_BN):
        return  nn.Sequential(
            conv_block_1(in_f, out_f, enable_BN, enable_dropOut),
            nn.MaxPool2d(2, 2),
            nn.BatchNorm2d(out_f)
        )
    else :
        return  nn.Sequential(
            conv_block_1(in_f, out_f, enable_BN, enable_dropOut),
            nn.MaxPool2d(2, 2)
        )



def conv_block_3(in1_f, out1_f,in2_f, out2_f, enable_BN, enable_dropOut):
        return  nn.Sequential(
            conv_block_1(in1_f, out1_f, enable_BN, enable_dropOut),
            conv_block_2(in2_f, out2_f, enable_BN, enable_dropOut)
        )

def linear_block(in_f, out_f, enable_dropOut, activation_function):
    if(enable_dropOut):
        return nn.Sequential(
             nn.Linear(in_f, out_f),
             nn.Dropout(0.2),
             activation_function
            )
    else:
        return nn.Sequential(
             nn.Linear(in_f, out_f),
             activation_function
            )

class Cifar10CnnModel_2Layer(Utils):
  def __init__(self, enable_BN, enable_dropOut):
        super().__init__()
        self.ConvLayer1 = conv_block_3(3, 32, 32, 64, enable_BN, enable_dropOut)
        self.ConvLayer2 = conv_block_3(64, 128, 128, 128, enable_BN, enable_dropOut) # out
        self.flatten =  nn.Flatten()
        self.Linear1 = linear_block(128 *8*8, 1024,enable_dropOut,nn.ReLU())
        self.Linear2 = linear_block(1024, 512, enable_dropOut,nn.ReLU())
        self.Linear3 = nn.Linear(512, 10)

  def forward(self, xb):
        x = self.ConvLayer1(xb)
        x = self.ConvLayer2(x)
        x = self.flatten(x)
        x = self.Linear1(x)
        x = self.Linear2(x)
        x = self.Linear3(x)
        return x

  def outputConv1(self, xb):
         x = self.ConvLayer1(xb)
         return x

  def outputConv2(self, xb):
         x = self.ConvLayer1(xb)
         x = self.ConvLayer2(x)
         x = self.flatten(x)
         return x



class Cifar10CnnModel_3Layer(Utils):
    def __init__(self, enable_BN, enable_dropOut):
        super().__init__()
        self.ConvLayer1 = conv_block_3(3, 32, 32, 64, enable_BN, enable_dropOut)
        self.ConvLayer2 = conv_block_3(64, 128, 128, 128, enable_BN, enable_dropOut) # out
        self.ConvLayer3 = conv_block_3(128, 256, 256, 256, enable_BN, enable_dropOut)
        self.flatten =  nn.Flatten()
        self.Linear1 = linear_block(256 *4*4, 1024,enable_dropOut,nn.ReLU())
        self.Linear2 = linear_block(1024, 512, enable_dropOut,nn.ReLU())
        self.Linear3 = nn.Linear(512, 10)

    def forward(self, xb):
        x = self.ConvLayer1(xb)
        x = self.ConvLayer2(x)
        x = self.ConvLayer3(x)
        x = self.flatten(x)
        x = self.Linear1(x)
        x = self.Linear2(x)
        x = self.Linear3(x)
        return x

    def outputConv1(self, xb):
         x = self.ConvLayer1(xb)
         return x

    def outputConv2(self, xb):
         x = self.ConvLayer1(xb)
         x = self.ConvLayer2(x)
         x = self.flatten(x)
         return x

    def outputConv3(self, xb):
         x = self.ConvLayer1(xb)
         x = self.ConvLayer2(x)
         x = self.ConvLayer3(x)
         x = self.flatten(x)
         return x
```

```
def split_dataset(dataset, ratio):
    subsetALength = (int)( len(dataset) * ratio)
    subsetA, subsetB = torch.utils.data.random_split(dataset,
                         [subsetALength, len(dataset)-subsetALength])
    subsetALoader = DataLoader(dataset=subsetA, batch_size=batch_size, shuffle=True)
    subsetBLoader =  DataLoader(dataset=subsetB, batch_size=batch_size, shuffle=True)
    return(subsetALoader,subsetBLoader,subsetA,subsetB)

def trainNetwork(model, train_loader, validation_loader,epochs,learning_rate):
    t1 = time.time()
    best_valid = None
    history = []
    optimizer = torch.optim.Adam(model.parameters(), learning_rate,weight_decay=0.0005)
    for epoch in range(epochs):
        # Training
        model.train()
        train_losses = []
        train_accuracy = []
        for batch in tqdm(train_loader):
            loss,accu = model.training_step(batch)
            train_losses.append(loss)
            train_accuracy.append(accu)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
        # Validation
        result = evaluate(model, validation_loader,"validation")
        result['train_loss'] = torch.stack(train_losses).mean().item()
        result['train_accuracy'] = torch.stack(train_accuracy).mean().item()
        model.at_epoch_end(epoch, result)
        if(best_valid == None or best_valid<result['validation_accuracy']):
            best_valid=result['validation_accuracy']
            torch.save(model.state_dict(), 'Lab2_P1'+ model.name +'.pth')
        history.append(result)
    t2 = time.time()
    timeToTrain = t2 - t1
    print(f'TIme to train {model.name} : {timeToTrain}')
    return (history, timeToTrain)


def plot_accuracies(history, model_name):
    Validation_accuracies = [x['validation_accuracy'] for x in history]
    Training_Accuracies = [x['train_accuracy'] for x in history]
    plt.figure()
    plt.plot(Training_Accuracies, '-rx')
    plt.plot(Validation_accuracies, '-bx')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.legend(['Training', 'Validation'])
    plt.title('Acc vs epochs for model ' + model_name);
    plt.savefig(f'{model_name}_acc.png')
    plt.close()

def plot_losses(history,model_name):
    train_losses = [x.get('train_loss') for x in history]
    val_losses = [x['validation_loss'] for x in history]
    plt.figure()
    plt.plot(train_losses, '-bx')
    plt.plot(val_losses, '-rx')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend(['Training', 'Validation'])
    plt.title('Loss vs epochs '+ model_name);
    plt.savefig(f'{model_name}_loss.png')
    plt.close()



#Splitting Dataset to Training & Test
tr = 0.7
batch_size = 64
trainLoader, testValidationLoader, trainDataset, testValidationDataset = split_dataset(CIF
testLoader, validationLoader, testDataset, validationDataset = split_dataset(testValidatio

#Splitting Dataset to Training & Test
tr = 0.7
batch_size = 64
trainLoaderAugmented, testValidationLoaderAugmented, trainDatasetAugmented, testValidation
testLoaderAugmented, validationLoaderAugmented, testDatasetAugmentedt, validationDatasetAu


#Question 1

# Hyperparameters
tr = 0.7
learning_rate = 0.001
batch_size = 64
epochs = 10

models = []

## No Regularizer

model = Cifar10CnnModel_2Layer(False, False).to(device)
model.name = "Lab2_P1_" +"Regularizer_None_"+"2LayerCNN"
models.append(model)

model = Cifar10CnnModel_3Layer(False, False).to(device)
model.name = "Lab2_P1_" +"Regularizer_None_"+"3LayerCNN"
models.append(model)


##Batch Norm
model = Cifar10CnnModel_2Layer(True, False).to(device)
model.name = "Lab2_P1_" +"Regularizer_BatchNorm_"+"2LayerCNN"
models.append(model)

model = Cifar10CnnModel_3Layer(True, False).to(device)
model.name = "Lab2_P1_" +"Regularizer_BatchNorm_"+"3LayerCNN"
models.append(model)
```

```
## Drop Out
model = Cifar10CnnModel_2Layer(False, True).to(device)
model.name = "Lab2_P1_" +"Regularizer_DropOut_"+"2LayerCNN"
models.append(model)

model = Cifar10CnnModel_3Layer(False, True).to(device)
model.name = "Lab2_P1_" +"Regularizer_DropOut_"+"3LayerCNN"
models.append(model)
```

```
testAccuracies = []
trainingTimes = []
for model in models:
    print(model.name)
    (listAccuracies, trainingTime) = trainNetwork(model,trainLoader, validationLoader,epoc
    trainingTimes.append(trainingTime)
    plot_accuracies(listAccuracies,model.name)
    plot_losses(listAccuracies,model.name)
    result = evaluate(model, testLoader, "test")
    testAccuracies.append(result)
```

```
Epoch : 7
Train Accuracy:78.40% Validation Accuracy:73.44%
Train Loss:0.6195 Validation Loss:0.7843
100%|██████████| 547/547 [00:09<00:00, 60.26it/s]
Epoch : 8
Train Accuracy:81.08% Validation Accuracy:74.11%
Train Loss:0.5350 Validation Loss:0.7785
100%|██████████| 547/547 [00:09<00:00, 60.02it/s]
Epoch : 9
Train Accuracy:83.83% Validation Accuracy:73.97%
Train Loss:0.4638 Validation Loss:0.7657
100%|██████████| 547/547 [00:09<00:00, 60.12it/s]
Epoch : 10
Train Accuracy:85.55% Validation Accuracy:75.49%
Train Loss:0.4105 Validation Loss:0.7356
TIme to train Lab2_P1_Regularizer_DropOut_2LayerCNN : 104.29744410514832
Lab2_P1_Regularizer_DropOut_3LayerCNN
100%|██████████| 547/547 [00:09<00:00, 55.09it/s]
Epoch : 1
Train Accuracy:9.73% Validation Accuracy:9.79%
Train Loss:2.3031 Validation Loss:2.3026
100%|██████████| 547/547 [00:09<00:00, 55.72it/s]
Epoch : 2
Train Accuracy:9.75% Validation Accuracy:10.49%
Train Loss:2.3028 Validation Loss:2.3027
100%|██████████| 547/547 [00:09<00:00, 55.92it/s]
Epoch : 3
Train Accuracy:9.86% Validation Accuracy:10.43%
Train Loss:2.3028 Validation Loss:2.3028
100%|██████████| 547/547 [00:09<00:00, 56.62it/s]
Epoch : 4
```

```
Train Accuracy:9.87% Validation Accuracy:9.70%
Train Loss:2.3027 Validation Loss:2.3029
100%|██████████| 547/547 [00:09<00:00, 57.06it/s]
Epoch : 5
Train Accuracy:9.78% Validation Accuracy:9.59%
Train Loss:2.3027 Validation Loss:2.3032
100%|██████████| 547/547 [00:09<00:00, 56.79it/s]
Epoch : 6
Train Accuracy:10.05% Validation Accuracy:9.76%
Train Loss:2.3028 Validation Loss:2.3028
100%|██████████| 547/547 [00:09<00:00, 57.14it/s]
Epoch : 7
Train Accuracy:9.95% Validation Accuracy:9.73%
Train Loss:2.3027 Validation Loss:2.3028
100%|██████████| 547/547 [00:09<00:00, 56.79it/s]
Epoch : 8
Train Accuracy:9.92% Validation Accuracy:9.42%
Train Loss:2.3028 Validation Loss:2.3032
100%|██████████| 547/547 [00:09<00:00, 56.55it/s]
Epoch : 9
Train Accuracy:10.06% Validation Accuracy:9.42%
Train Loss:2.3028 Validation Loss:2.3032
100%|██████████| 547/547 [00:09<00:00, 56.52it/s]
Epoch : 10
Train Accuracy:10.01% Validation Accuracy:9.76%
Train Loss:2.3028 Validation Loss:2.3030
TIme to train Lab2_P1_Regularizer_DropOut_3LayerCNN : 110.35315823554993
```

```
## No Regularizer
modelsAugmented = []
model = Cifar10CnnModel_2Layer(False, False).to(device)
model.name = "Lab2_P1_" +"Regularizer_DataAugmentation_"+"2LayerCNN"
modelsAugmented.append(model)

model = Cifar10CnnModel_3Layer(False, False).to(device)
model.name = "Lab2_P1_" +"Regularizer_DataAugmentation_"+"3LayerCNN"
modelsAugmented.append(model)

for model in modelsAugmented:
    print(model.name)
    (listAccuracies, trainingTime) = trainNetwork(model,trainLoaderAugmented, validationLo
    trainingTimes.append(trainingTime)
    plot_accuracies(listAccuracies,model.name)
    plot_losses(listAccuracies,model.name)
    result = evaluate(model, testLoaderAugmented, "test")
    testAccuracies.append(result)
```

```
Epoch : 7
Train Accuracy:71.19% Validation Accuracy:74.22%
Train Loss:0.8112 Validation Loss:0.7325
100%|██████████| 547/547 [00:12<00:00, 43.49it/s]
Epoch : 8
Train Accuracy:72.65% Validation Accuracy:74.78%
Train Loss:0.7707 Validation Loss:0.7231
100%|██████████| 547/547 [00:12<00:00, 43.55it/s]
Epoch : 9
Train Accuracy:73.39% Validation Accuracy:75.54%
Train Loss:0.7502 Validation Loss:0.6911
100%|██████████| 547/547 [00:12<00:00, 43.23it/s]
```

```
100%|██████████| 547/547 [00:12<00:00, 43.23it/s]
Epoch : 10
Train Accuracy:74.23% Validation Accuracy:76.23%
Train Loss:0.7285 Validation Loss:0.6630
TIme to train Lab2_P1_Regularizer_DataAugmentation_2LayerCNN : 139.46855878829956
Lab2_P1_Regularizer_DataAugmentation_3LayerCNN
100%|██████████| 547/547 [00:13<00:00, 40.57it/s]
Epoch : 1
Train Accuracy:10.15% Validation Accuracy:9.77%
Train Loss:2.3030 Validation Loss:2.3031
100%|██████████| 547/547 [00:13<00:00, 40.17it/s]
Epoch : 2
Train Accuracy:10.15% Validation Accuracy:9.77%
Train Loss:2.3027 Validation Loss:2.3030
100%|██████████| 547/547 [00:13<00:00, 40.57it/s]
Epoch : 3
Train Accuracy:10.29% Validation Accuracy:9.83%
Train Loss:2.3027 Validation Loss:2.3028
100%|██████████| 547/547 [00:13<00:00, 40.54it/s]
Epoch : 4
Train Accuracy:10.14% Validation Accuracy:9.83%
Train Loss:2.3027 Validation Loss:2.3028
100%|██████████| 547/547 [00:13<00:00, 40.76it/s]
Epoch : 5
Train Accuracy:10.29% Validation Accuracy:9.89%
Train Loss:2.3027 Validation Loss:2.3028
100%|██████████| 547/547 [00:13<00:00, 40.72it/s]
Epoch : 6
Train Accuracy:10.29% Validation Accuracy:9.77%
Train Loss:2.3027 Validation Loss:2.3027
100%|██████████| 547/547 [00:13<00:00, 40.65it/s]
Epoch : 7
Train Accuracy:10.29% Validation Accuracy:9.89%
Train Loss:2.3027 Validation Loss:2.3027
100%|██████████| 547/547 [00:13<00:00, 40.52it/s]
Epoch : 8
Train Accuracy:10.12% Validation Accuracy:9.89%
Train Loss:2.3028 Validation Loss:2.3027
100%|██████████| 547/547 [00:13<00:00, 40.91it/s]
Epoch : 9
Train Accuracy:10.18% Validation Accuracy:9.77%
Train Loss:2.3027 Validation Loss:2.3031
100%|██████████| 547/547 [00:13<00:00, 40.03it/s]
Epoch : 10
Train Accuracy:10.19% Validation Accuracy:9.94%
Train Loss:2.3028 Validation Loss:2.3027
TIme to train Lab2_P1_Regularizer_DataAugmentation_3LayerCNN : 148.44200730323792
```

```
testAccuracies
```

```
[{'test_loss': 0.8836011290550232, 'test_accuracy': 0.7393185496330261},
 {'test_loss': 0.7560953497886658, 'test_accuracy': 0.7452330589294434},
 {'test_loss': 1.0305886268615723, 'test_accuracy': 0.741525411605835},
 {'test_loss': 0.7129078507423401, 'test_accuracy': 0.7845162153244019},
 {'test_loss': 0.7489970326423645, 'test_accuracy': 0.754016637802124},
 {'test_loss': 2.3027384281158447, 'test_accuracy': 0.09706038236618042},
 {'test_loss': 0.6891036033630371, 'test_accuracy': 0.7572386860847473},
 {'test_loss': 2.302833080291748, 'test_accuracy': 0.09666313230991364}]
```

```
for model in models:
  print(model.name)
```

```
Lab2_P1_Regularizer_None_2LayerCNN
Lab2_P1_Regularizer_None_3LayerCNN
Lab2_P1_Regularizer_BatchNorm_2LayerCNN
Lab2_P1_Regularizer_BatchNorm_3LayerCNN
Lab2_P1_Regularizer_DropOut_2LayerCNN
Lab2_P1_Regularizer_DropOut_3LayerCNN
```

```
finalModel = models[3]
```

```
finalModel.name
```

```
'Lab2_P1_Regularizer_BatchNorm_3LayerCNN'
```

```
finalModel
```

```
Cifar10CnnModel_3Layer(
  (ConvLayer1): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (1): Sequential(
      (0): Sequential(
        (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (ConvLayer2): Sequential(
    (0): Sequential(
      (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (1): Sequential(
      (0): Sequential(
        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (ConvLayer3): Sequential(
    (0): Sequential(
      (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (1): Sequential(
      (0): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (Linear1): Sequential(
    (0): Linear(in_features=4096, out_features=1024, bias=True)
    (1): ReLU()
  )
  (Linear2): Sequential(
    (0): Linear(in_features=1024, out_features=512, bias=True)
    (1): ReLU()
  )
```

```
modelq2 = Cifar10CnnModel_3Layer(True, False)
print(modelq2)
modelq2.name = 'modelq2'
print(modelq2.name)
```

```
Cifar10CnnModel_3Layer(
  (ConvLayer1): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (1): Sequential(
      (0): Sequential(
        (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
    )
  )
  (ConvLayer2): Sequential(
    (0): Sequential(
      (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (1): Sequential(
      (0): Sequential(
        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
    )
  )
  (ConvLayer3): Sequential(
    (0): Sequential(
      (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
        (1): ReLU()
      )
      (1): Sequential(
        (0): Sequential(
          (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): ReLU()
        )
        (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False
        (2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats
      )
    )
    (flatten): Flatten(start_dim=1, end_dim=-1)
    (Linear1): Sequential(
      (0): Linear(in_features=4096, out_features=1024, bias=True)
      (1): ReLU()
    )
    (Linear2): Sequential(
      (0): Linear(in_features=1024, out_features=512, bias=True)
      (1): ReLU()
    )
    (Linear3): Linear(in_features=512, out_features=10, bias=True)
  )
  modelq2
```

```
modelq2.to(device=device)
```

```
Cifar10CnnModel_3Layer(
  (ConvLayer1): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (1): Sequential(
      (0): Sequential(
        (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (ConvLayer2): Sequential(
    (0): Sequential(
      (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (1): Sequential(
      (0): Sequential(
        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (ConvLayer3): Sequential(
    (0): Sequential(
      (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (1): Sequential(
      (0): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (Linear1): Sequential(
    (0): Linear(in_features=4096, out_features=1024, bias=True)
    (1): ReLU()
  )
  (Linear2): Sequential(
    (0): Linear(in_features=1024, out_features=512, bias=True)
    (1): ReLU()
  )
```

```
tr = 0.85
batch_size = 64
trainLoader, testLoader, trainDataset, testDataset = split_dataset(CIFAR10_dataset, tr)
print(modelq2.name)
(listAccuraciesq2, trainingTimeq2) = trainNetwork(modelq2,trainLoader, testLoader,epochs,l
plot_accuracies(listAccuraciesq2,modelq2.name)
plot_losses(listAccuraciesq2,modelq2.name)
resultq2 = evaluate(modelq2, testLoader, "test")
```

```
modelq2
100%|██████████| 665/665 [00:11<00:00, 56.09it/s]
Epoch : 1
Train Accuracy:90.28% Validation Accuracy:88.87%
Train Loss:0.2876 Validation Loss:0.3318
100%|██████████| 665/665 [00:11<00:00, 55.98it/s]
Epoch : 2
Train Accuracy:91.76% Validation Accuracy:88.32%
Train Loss:0.2375 Validation Loss:0.3449
100%|██████████| 665/665 [00:11<00:00, 55.72it/s]
Epoch : 3
Train Accuracy:93.00% Validation Accuracy:86.04%
Train Loss:0.2017 Validation Loss:0.4331
100%|██████████| 665/665 [00:11<00:00, 55.89it/s]
Epoch : 4
Train Accuracy:93.20% Validation Accuracy:85.28%
Train Loss:0.1959 Validation Loss:0.4619
100%|██████████| 665/665 [00:11<00:00, 55.64it/s]
Epoch : 5
Train Accuracy:93.59% Validation Accuracy:84.20%
Train Loss:0.1845 Validation Loss:0.5067
```

```
100%|██████████| 665/665 [00:12<00:00, 55.24it/s]
Epoch : 6
Train Accuracy:93.80% Validation Accuracy:84.71%
Train Loss:0.1787 Validation Loss:0.4919
100%|██████████| 665/665 [00:12<00:00, 54.89it/s]
Epoch : 7
Train Accuracy:94.20% Validation Accuracy:84.29%
Train Loss:0.1659 Validation Loss:0.5337
100%|██████████| 665/665 [00:12<00:00, 55.08it/s]
Epoch : 8
Train Accuracy:94.91% Validation Accuracy:85.17%
Train Loss:0.1430 Validation Loss:0.5012
100%|██████████| 665/665 [00:12<00:00, 55.37it/s]
Epoch : 9
Train Accuracy:95.01% Validation Accuracy:84.38%
Train Loss:0.1443 Validation Loss:0.5296
100%|██████████| 665/665 [00:12<00:00, 55.07it/s]
Epoch : 10
Train Accuracy:94.94% Validation Accuracy:81.55%
Train Loss:0.1444 Validation Loss:0.6436
TIme to train modelq2 : 133.0724036693573
```

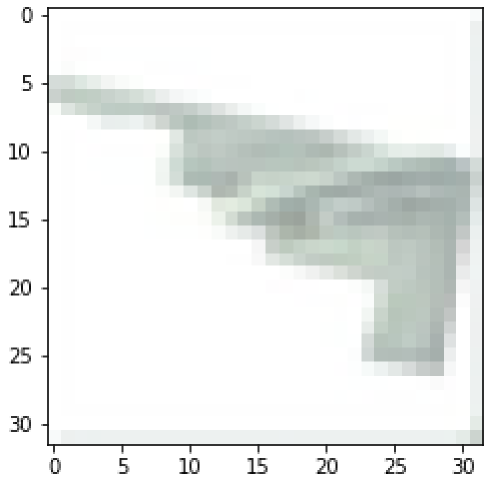
```
resultq2
```

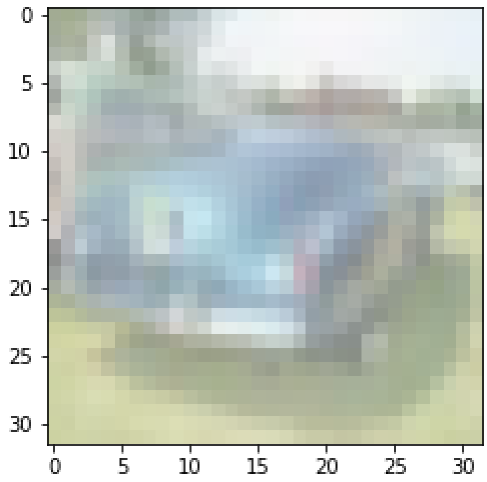
```
{'test_loss': 0.6509191989898682, 'test_accuracy': 0.8138241767883301}
```

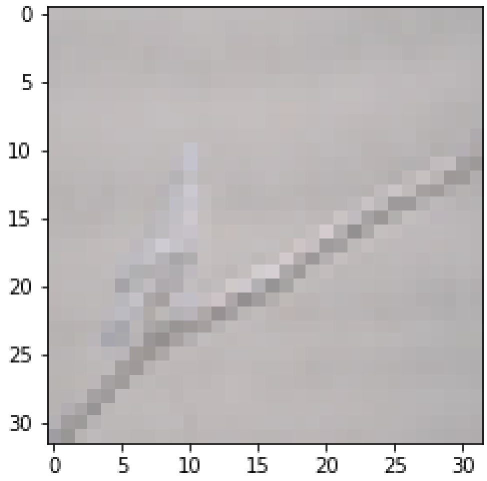
```
def imshow(img):
  img = img / 2 + 0.5   # unnormalize
  npimg = img.numpy()   # convert from tensor
  plt.imshow(np.transpose(npimg, (1, 2, 0)))
  plt.show()

classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog','frog', 'horse', 'ship', 'truck']

#   # get first 100 training images
# dataiter = iter(trainLoader)
# n = nn.Softmax(dim=1);
# imgs, lbls = dataiter.next()

# a = torch.Tensor(50000, 3, 32, 32)
# for i in range(1000):  # show just the frogs
#     imshow(utils.make_grid(imgs[i]))
#     print(classes[torch.argmax(n(modelq2.forward(imgs[i].reshape(1,3,32,32).to(device=de
```

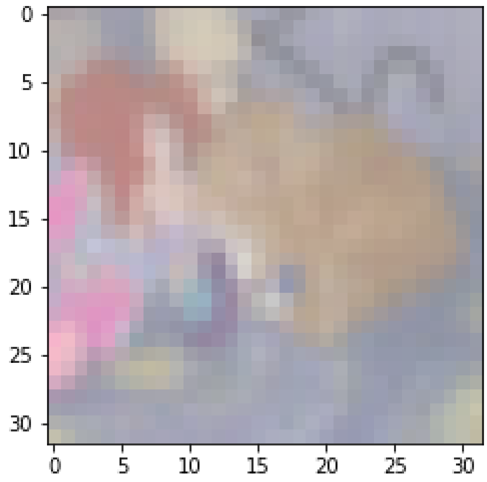
```
l = []
a = torch.Tensor(15000, 3, 32, 32)
for i, (image, label) in enumerate(testDataset):
        a[i, :, :, :] = image
        l.append(label)
```

```
finalList = []
forwardEmbeddings = []
Conv2Embeddings = []
for i in enumerate(classes):
  finalList.append([])
  forwardEmbeddings.append([])
  Conv2Embeddings.append([])

for i in range(7500):
    finalList[l[i]].append(a[i, :, : , :])



for i in range(len(finalList)):
  for j in range(1):
    imshow(utils.make_grid(finalList[i][j]))
```

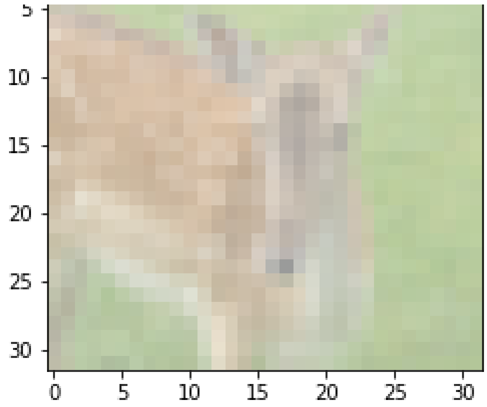

0
5
10
15
20
25
30
0 5 10 15 20 25 30

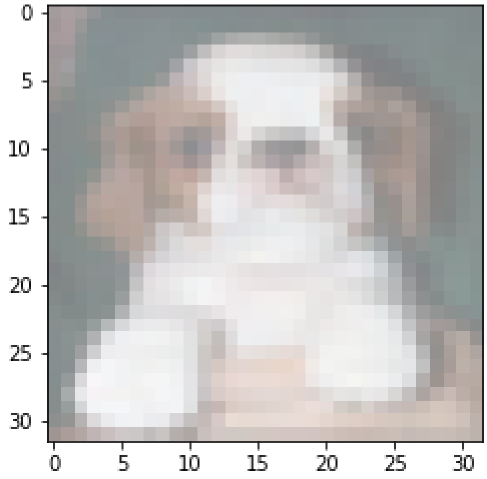

0
5
10
15
20
25
30
0 5 10 15 20 25 30

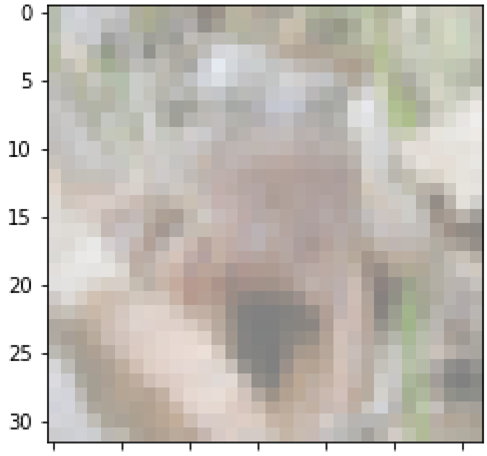

0
5
10
15
20
25
30
0 5 10 15 20 25 30


0
5
10
15
20
25
30
0 5 10 15 20 25 30

0







```
forwardEmbeddingsCov = []
conv2dEmbeddingsConv = []
for i in range(len(classes)):
  forwardEmbeddingsCov.append([])
  conv2dEmbeddingsConv.append([])
```

```
modelq2.eval()
with torch.no_grad():
  for i in range(len(finalList)):
      for j in range(len(finalList[i])):
        forwardEmbeddingsCov[i].append(modelq2.forward(finalList[i][j].reshape(1,3,32,32).
        conv2dEmbeddingsConv[i].append(modelq2.outputConv2(finalList[i][j].reshape(1,3,32,
```

```
def plot_forward(m):
  f, axarr = plt.subplots(2,6,figsize=(10,4))
  k = 0
  for i in range(2):
    for j in range(6):
      img = utils.make_grid(finalList[simiL_forward[k][1]][simiL_forward[k][2]])
      img = img / 2 + 0.5   # unnormalize
      npimg = img.numpy()
      #print(npimg.shape)  # convert from tensor
      axarr[i,j].imshow(np.transpose(npimg, (1, 2, 0)))
      axarr[i,j].set_title(f'{classes[simiL_forward[k][1]]}')
      axarr[i,j].axis('off')
      k = k+1

  f.suptitle(f'{classes[m]} similar Images from final Layer Representations')
  plt.savefig(f'{classes[m]}1.png', bbox_inches='tight')


def plot_conv2d(m) :
  f, axarr = plt.subplots(2,6,figsize=(10,4))
  k = 0
  for i in range(2):
    for j in range(6):
      img = utils.make_grid(finalList[simiL_convolution[k][1]][simiL_convolution[k][2]])
      img = img / 2 + 0.5   # unnormalize
      npimg = img.numpy()
      #print(npimg.shape)  # convert from tensor
      axarr[i,j].imshow(np.transpose(npimg, (1, 2, 0)))
      axarr[i,j].set_title(f'{classes[simiL_convolution[k][1]]}')
      axarr[i,j].axis('off')
      k = k+1

  f.suptitle(f'{classes[m]} similar Images from second Convolution Layer Representations')
  plt.savefig(f'{classes[m]}'+'2.png', bbox_inches='tight')




for m in range(10):
  simiL_forward = []
  simiL_convolution = []
  cos = nn.CosineSimilarity(dim=0, eps=1e-6)
  for i in range(10) :
    for j in range(100):
      simiL_forward.append((cos(forwardEmbeddingsCov[m][0].squeeze(),forwardEmbeddingsCov[
      simiL_convolution.append((cos(conv2dEmbeddingsConv[m][0].squeeze(),conv2dEmbeddingsC
  simiL_forward.sort(key = lambda x: x[0], reverse=True)
  simiL_convolution.sort(key = lambda x: x[0], reverse=True)
  plot_forward(m)
  plot_conv2d(m)
```