```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torchsummary import summary
import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix
import seaborn as sn
import pandas as pd
import numpy as np

# Set the path where you want to save the checkpoint
path = "/content/drive/MyDrive/my_checkpoint_cosineLR_v1.pth"

# Define the device to use for training
device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")

# Define the CNN architecture
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32,
kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.relu1 = nn.ReLU()
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64,
kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.relu2 = nn.ReLU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=128,
kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.relu3 = nn.ReLU()
        self.conv4 = nn.Conv2d(in_channels=128, out_channels=256,
kernel_size=3, padding=1)
        self.bn4 = nn.BatchNorm2d(256)
        self.relu4 = nn.ReLU()
        self.maxpool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv5 = nn.Conv2d(in_channels=256, out_channels=512,
kernel_size=3, padding=1)
```

```
        self.bn5 = nn.BatchNorm2d(512)
        self.relu5 = nn.ReLU()
        self.conv6 = nn.Conv2d(in_channels=512, out_channels=1024,
kernel_size=3, padding=1)
        self.bn6 = nn.BatchNorm2d(1024)
        self.relu6 = nn.ReLU()
        self.maxpool3 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(in_features=4*4*1024, out_features=512)
        self.relu7 = nn.ReLU()
        self.dropout1 = nn.Dropout(p=0.5)
        self.fc2 = nn.Linear(in_features=512, out_features=10)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu1(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = self.relu2(x)
        x = self.maxpool1(x)
        x = self.conv3(x)
        x = self.bn3(x)
        x = self.relu3(x)
        x = self.conv4(x)
        x = self.bn4(x)
        x = self.relu4(x)
        x = self.maxpool2(x)
        x = self.conv5(x)
        x = self.bn5(x)
        x = self.relu5(x)
        x = self.conv6(x)
        x = self.bn6(x)
        x = self.relu6(x)
        x = self.maxpool3(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = self.relu7(x)
        x = self.dropout1(x)
        x = self.fc2(x)
        return x
net = Net()
net.cuda()
```

```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu1): ReLU()
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1),
```

```
padding=(1, 1))
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu2): ReLU()
  (maxpool1): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
  (bn3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu3): ReLU()
  (conv4): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
  (bn4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu4): ReLU()
  (maxpool2): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
  (conv5): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
  (bn5): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu5): ReLU()
  (conv6): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
  (bn6): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu6): ReLU()
  (maxpool3): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=16384, out_features=512, bias=True)
  (relu7): ReLU()
  (dropout1): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=512, out_features=10, bias=True)
)
```

```
summary(net, (3, 32, 32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 32, 32]             896
       BatchNorm2d-2           [-1, 32, 32, 32]              64
              ReLU-3           [-1, 32, 32, 32]               0
            Conv2d-4           [-1, 64, 32, 32]          18,496
       BatchNorm2d-5           [-1, 64, 32, 32]             128
              ReLU-6           [-1, 64, 32, 32]               0
         MaxPool2d-7           [-1, 64, 16, 16]               0
            Conv2d-8          [-1, 128, 16, 16]          73,856
       BatchNorm2d-9          [-1, 128, 16, 16]             256
             ReLU-10          [-1, 128, 16, 16]               0
```

```
         Conv2d-11          [-1, 256, 16, 16]         295,168
    BatchNorm2d-12          [-1, 256, 16, 16]             512
           ReLU-13          [-1, 256, 16, 16]               0
      MaxPool2d-14            [-1, 256, 8, 8]               0
         Conv2d-15            [-1, 512, 8, 8]       1,180,160
    BatchNorm2d-16            [-1, 512, 8, 8]           1,024
           ReLU-17            [-1, 512, 8, 8]               0
         Conv2d-18           [-1, 1024, 8, 8]       4,719,616
    BatchNorm2d-19           [-1, 1024, 8, 8]           2,048
           ReLU-20           [-1, 1024, 8, 8]               0
      MaxPool2d-21           [-1, 1024, 4, 4]               0
         Linear-22                  [-1, 512]       8,389,120
           ReLU-23                  [-1, 512]               0
        Dropout-24                  [-1, 512]               0
         Linear-25                   [-1, 10]           5,130
================================================================
Total params: 14,686,474
Trainable params: 14,686,474
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 7.14
Params size (MB): 56.02
Estimated Total Size (MB): 63.17
----------------------------------------------------------------

# Define the transformations for the dataset
transform_train = transforms.Compose(
    [transforms.Resize((32,32)),  #resises the image so it can be
perfect for our model.
     transforms.RandomHorizontalFlip(), # FLips the image w.r.t
horizontal axis
     transforms.RandomRotation(10),     #Rotates the image to a
specified angel
     transforms.RandomAffine(0, shear=10, scale=(0.8,1.2)), #Performs
actions like zooms, change shear angles.
     transforms.RandomCrop(32, padding=4),
     transforms.ColorJitter(brightness=0.2, contrast=0.2,
saturation=0.2),
     transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

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


# Download and load the CIFAR-10 dataset
train_dataset = torchvision.datasets.CIFAR10(root='./data',
train=True,
```

```
                                        download=True,
transform=transform_train)
train_loader = torch.utils.data.DataLoader(train_dataset,
batch_size=32,
                                          shuffle=True, num_workers=2)

test_dataset = torchvision.datasets.CIFAR10(root='./data',
train=False,
                                       download=True,
transform=transform_test)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=32,
                                         shuffle=False, num_workers=2)

Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
./data/cifar-10-python.tar.gz

100%|██████████| 170498071/170498071 [00:02<00:00, 76882319.51it/s]

Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified

# Define the loss function and the optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.005, momentum=0.9,
weight_decay=5e-4)
#scheduler = ReduceLROnPlateau(optimizer, mode='min', factor=0.1,
patience=3, verbose=True)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=50,
eta_min=1e-8)

# Set the device to use (GPU or CPU)
device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")

# initialize lists to store loss and accuracy
train_loss = []
train_acc = []
v_loss = []
v_acc = []
learn_rate = []

num_epochs = 50
# train the model
for epoch in range(num_epochs):
    running_loss = 0.0
    running_corrects = 0
    for images, labels in train_loader:
        # move data to device
        images, labels = images.to(device), labels.to(device)
        # zero the parameter gradients
        optimizer.zero_grad()
```

```
        # forward pass
        outputs = net(images)
        loss = criterion(outputs, labels)
        # backward pass and optimize
        loss.backward()
        optimizer.step()
        # calculate running loss and accuracy
        running_loss += loss.item() * images.size(0)
        _, preds = torch.max(outputs, 1)
        running_corrects += torch.sum(preds == labels.data)
    epoch_loss = running_loss / len(train_dataset)
    epoch_acc = 100 * running_corrects.double() / len(train_dataset)
    train_loss.append(epoch_loss)
    train_acc.append(epoch_acc)
    # print statistics
    print('Epoch [{}/{}],Training Loss: {:.4f}, Training Accuracy:
{:.4f} %'.format(epoch+1, num_epochs, epoch_loss, epoch_acc))

    # Validate the model
    net.eval()
    val_loss = 0.0
    val_correct = 0
    total = 0

    with torch.no_grad():
        for data in test_loader:
            inputs, labels = data[0].to(device), data[1].to(device)
            outputs = net(inputs)
            loss = criterion(outputs, labels)

            val_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            val_correct += (predicted == labels).sum().item()

        val_acc = 100 * val_correct / total
        val_loss /= len(test_loader)
        v_acc.append(val_acc)
        v_loss.append(val_loss)

    # Print the training and validation metrics for each epoch
    print('Test Epoch [{}/{}], Validation Loss: {:.4f}, Validation
Accuracy: {:.2f}%'
          .format(epoch+1, num_epochs,val_loss, val_acc))

    # Update the learning rate scheduler
    # scheduler.step(val_loss)
    scheduler.step()
    epoch_lr = optimizer.param_groups[0]['lr']
```

```
    print("Epoch:", epoch+1, "Learning rate:", epoch_lr )
    learn_rate.append(epoch_lr)
```

```
Epoch [1/50],Training Loss: 2.0764, Training Accuracy: 20.7200 %
Test Epoch [1/50], Validation Loss: 1.8293, Validation Accuracy:
33.61%
Epoch: 1 Learning rate: 0.0049950668309370365
Epoch [2/50],Training Loss: 1.5277, Training Accuracy: 43.4580 %
Test Epoch [2/50], Validation Loss: 1.2671, Validation Accuracy:
54.79%
Epoch: 2 Learning rate: 0.004980286792712688
Epoch [3/50],Training Loss: 1.1756, Training Accuracy: 57.6320 %
Test Epoch [3/50], Validation Loss: 0.8849, Validation Accuracy:
68.19%
Epoch: 3 Learning rate: 0.004955718215385468
Epoch [4/50],Training Loss: 0.9834, Training Accuracy: 65.4460 %
Test Epoch [4/50], Validation Loss: 0.7823, Validation Accuracy:
72.45%
Epoch: 4 Learning rate: 0.004921458059905771
Epoch [5/50],Training Loss: 0.8572, Training Accuracy: 70.0420 %
Test Epoch [5/50], Validation Loss: 0.6847, Validation Accuracy:
75.89%
Epoch: 5 Learning rate: 0.004877641535455302
Epoch [6/50],Training Loss: 0.7656, Training Accuracy: 73.2780 %
Test Epoch [6/50], Validation Loss: 0.5794, Validation Accuracy:
80.40%
Epoch: 6 Learning rate: 0.004824441565838198
Epoch [7/50],Training Loss: 0.6979, Training Accuracy: 75.7200 %
Test Epoch [7/50], Validation Loss: 0.6195, Validation Accuracy:
78.46%
Epoch: 7 Learning rate: 0.004762068107029786
Epoch [8/50],Training Loss: 0.6493, Training Accuracy: 77.4260 %
Test Epoch [8/50], Validation Loss: 0.5301, Validation Accuracy:
82.39%
Epoch: 8 Learning rate: 0.004690767318576258
Epoch [9/50],Training Loss: 0.6088, Training Accuracy: 78.9640 %
Test Epoch [9/50], Validation Loss: 0.5176, Validation Accuracy:
82.20%
Epoch: 9 Learning rate: 0.0046108205921154095
Epoch [10/50],Training Loss: 0.5670, Training Accuracy: 80.2120 %
Test Epoch [10/50], Validation Loss: 0.4583, Validation Accuracy:
84.41%
Epoch: 10 Learning rate: 0.004522543440852396
Epoch [11/50],Training Loss: 0.5326, Training Accuracy: 81.5060 %
Test Epoch [11/50], Validation Loss: 0.4503, Validation Accuracy:
84.60%
Epoch: 11 Learning rate: 0.0044262842543732585
Epoch [12/50],Training Loss: 0.5094, Training Accuracy: 82.2900 %
Test Epoch [12/50], Validation Loss: 0.4118, Validation Accuracy:
86.04%
```

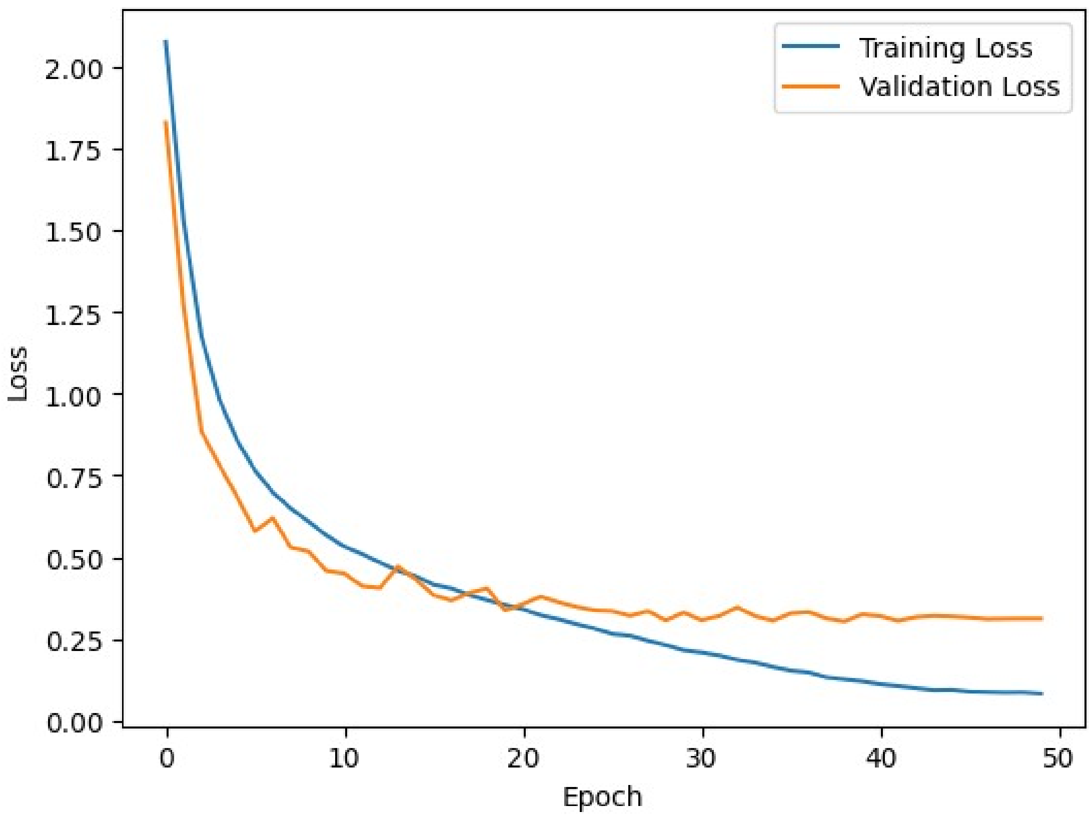
```
Epoch: 12 Learning rate: 0.0043224229237103905
Epoch [13/50],Training Loss: 0.4836, Training Accuracy: 83.1900 %
Test Epoch [13/50], Validation Loss: 0.4059, Validation Accuracy:
85.92%
Epoch: 13 Learning rate: 0.004211369342086191
Epoch [14/50],Training Loss: 0.4585, Training Accuracy: 84.1280 %
Test Epoch [14/50], Validation Loss: 0.4719, Validation Accuracy:
84.16%
Epoch: 14 Learning rate: 0.004093561787251775
Epoch [15/50],Training Loss: 0.4419, Training Accuracy: 84.6680 %
Test Epoch [15/50], Validation Loss: 0.4321, Validation Accuracy:
85.26%
Epoch: 15 Learning rate: 0.003969465191804921
Epoch [16/50],Training Loss: 0.4162, Training Accuracy: 85.4600 %
Test Epoch [16/50], Validation Loss: 0.3843, Validation Accuracy:
86.77%
Epoch: 16 Learning rate: 0.0038395693083135155
Epoch [17/50],Training Loss: 0.4047, Training Accuracy: 85.9460 %
Test Epoch [17/50], Validation Loss: 0.3678, Validation Accuracy:
87.51%
Epoch: 17 Learning rate: 0.003704386776485917
Epoch [18/50],Training Loss: 0.3841, Training Accuracy: 86.6380 %
Test Epoch [18/50], Validation Loss: 0.3907, Validation Accuracy:
86.88%
Epoch: 18 Learning rate: 0.003564451100016223
Epoch [19/50],Training Loss: 0.3688, Training Accuracy: 87.0560 %
Test Epoch [19/50], Validation Loss: 0.4047, Validation Accuracy:
86.59%
Epoch: 19 Learning rate: 0.003420314541088931
Epoch [20/50],Training Loss: 0.3540, Training Accuracy: 87.6880 %
Test Epoch [20/50], Validation Loss: 0.3375, Validation Accuracy:
88.49%
Epoch: 20 Learning rate: 0.0032725459408523955
Epoch [21/50],Training Loss: 0.3410, Training Accuracy: 88.0680 %
Test Epoch [21/50], Validation Loss: 0.3565, Validation Accuracy:
88.33%
Epoch: 21 Learning rate: 0.0031217284744627
Epoch [22/50],Training Loss: 0.3224, Training Accuracy: 88.6340 %
Test Epoch [22/50], Validation Loss: 0.3796, Validation Accuracy:
87.34%
Epoch: 22 Learning rate: 0.002968457349557738
Epoch [23/50],Training Loss: 0.3102, Training Accuracy: 89.1260 %
Test Epoch [23/50], Validation Loss: 0.3621, Validation Accuracy:
88.32%
Epoch: 23 Learning rate: 0.0028133374572445924
Epoch [24/50],Training Loss: 0.2955, Training Accuracy: 89.6360 %
Test Epoch [24/50], Validation Loss: 0.3482, Validation Accuracy:
88.61%
Epoch: 24 Learning rate: 0.002656980984870685
Epoch [25/50],Training Loss: 0.2822, Training Accuracy: 90.1680 %
```

```
Test Epoch [25/50], Validation Loss: 0.3376, Validation Accuracy:
88.73%
Epoch: 25 Learning rate: 0.002500004999999999
Epoch [26/50],Training Loss: 0.2655, Training Accuracy: 90.6500 %
Test Epoch [26/50], Validation Loss: 0.3352, Validation Accuracy:
89.13%
Epoch: 26 Learning rate: 0.0023430290151293135
Epoch [27/50],Training Loss: 0.2599, Training Accuracy: 90.9260 %
Test Epoch [27/50], Validation Loss: 0.3208, Validation Accuracy:
89.70%
Epoch: 27 Learning rate: 0.0021866725427554068
Epoch [28/50],Training Loss: 0.2448, Training Accuracy: 91.4860 %
Test Epoch [28/50], Validation Loss: 0.3353, Validation Accuracy:
89.39%
Epoch: 28 Learning rate: 0.002031552650442261
Epoch [29/50],Training Loss: 0.2313, Training Accuracy: 91.9760 %
Test Epoch [29/50], Validation Loss: 0.3068, Validation Accuracy:
90.04%
Epoch: 29 Learning rate: 0.0018782815255372984
Epoch [30/50],Training Loss: 0.2155, Training Accuracy: 92.3560 %
Test Epoch [30/50], Validation Loss: 0.3307, Validation Accuracy:
89.71%
Epoch: 30 Learning rate: 0.0017274640591476039
Epoch [31/50],Training Loss: 0.2088, Training Accuracy: 92.7820 %
Test Epoch [31/50], Validation Loss: 0.3075, Validation Accuracy:
90.57%
Epoch: 31 Learning rate: 0.001579695458911069
Epoch [32/50],Training Loss: 0.1994, Training Accuracy: 93.0040 %
Test Epoch [32/50], Validation Loss: 0.3204, Validation Accuracy:
90.02%
Epoch: 32 Learning rate: 0.0014355588999837758
Epoch [33/50],Training Loss: 0.1861, Training Accuracy: 93.4720 %
Test Epoch [33/50], Validation Loss: 0.3466, Validation Accuracy:
89.84%
Epoch: 33 Learning rate: 0.0012956232235140817
Epoch [34/50],Training Loss: 0.1776, Training Accuracy: 93.8200 %
Test Epoch [34/50], Validation Loss: 0.3200, Validation Accuracy:
90.32%
Epoch: 34 Learning rate: 0.0011604406916864824
Epoch [35/50],Training Loss: 0.1644, Training Accuracy: 94.3000 %
Test Epoch [35/50], Validation Loss: 0.3059, Validation Accuracy:
90.88%
Epoch: 35 Learning rate: 0.0010305448081950786
Epoch [36/50],Training Loss: 0.1533, Training Accuracy: 94.7240 %
Test Epoch [36/50], Validation Loss: 0.3285, Validation Accuracy:
90.63%
Epoch: 36 Learning rate: 0.0009064482127482241
Epoch [37/50],Training Loss: 0.1479, Training Accuracy: 94.8620 %
Test Epoch [37/50], Validation Loss: 0.3326, Validation Accuracy:
90.37%
```

```
Epoch: 37 Learning rate: 0.0007886406579138076
Epoch [38/50],Training Loss: 0.1328, Training Accuracy: 95.4200 %
Test Epoch [38/50], Validation Loss: 0.3128, Validation Accuracy:
91.10%
Epoch: 38 Learning rate: 0.0006775870762896086
Epoch [39/50],Training Loss: 0.1267, Training Accuracy: 95.6260 %
Test Epoch [39/50], Validation Loss: 0.3037, Validation Accuracy:
91.40%
Epoch: 39 Learning rate: 0.0005737257456267409
Epoch [40/50],Training Loss: 0.1205, Training Accuracy: 95.8280 %
Test Epoch [40/50], Validation Loss: 0.3252, Validation Accuracy:
90.69%
Epoch: 40 Learning rate: 0.00047746655914760334
Epoch [41/50],Training Loss: 0.1119, Training Accuracy: 96.1400 %
Test Epoch [41/50], Validation Loss: 0.3200, Validation Accuracy:
90.88%
Epoch: 41 Learning rate: 0.00038918940788459027
Epoch [42/50],Training Loss: 0.1063, Training Accuracy: 96.2800 %
Test Epoch [42/50], Validation Loss: 0.3056, Validation Accuracy:
91.51%
Epoch: 42 Learning rate: 0.0003092426814237412
Epoch [43/50],Training Loss: 0.1000, Training Accuracy: 96.5680 %
Test Epoch [43/50], Validation Loss: 0.3167, Validation Accuracy:
91.27%
Epoch: 43 Learning rate: 0.00023794189297021392
Epoch [44/50],Training Loss: 0.0937, Training Accuracy: 96.7180 %
Test Epoch [44/50], Validation Loss: 0.3208, Validation Accuracy:
91.09%
Epoch: 44 Learning rate: 0.00017556843416180104
Epoch [45/50],Training Loss: 0.0948, Training Accuracy: 96.7520 %
Test Epoch [45/50], Validation Loss: 0.3189, Validation Accuracy:
91.31%
Epoch: 45 Learning rate: 0.0001223684645446976
Epoch [46/50],Training Loss: 0.0887, Training Accuracy: 96.9640 %
Test Epoch [46/50], Validation Loss: 0.3152, Validation Accuracy:
91.48%
Epoch: 46 Learning rate: 7.85519400942282e-05
Epoch [47/50],Training Loss: 0.0874, Training Accuracy: 97.0360 %
Test Epoch [47/50], Validation Loss: 0.3108, Validation Accuracy:
91.46%
Epoch: 47 Learning rate: 4.429178461453183e-05
Epoch [48/50],Training Loss: 0.0862, Training Accuracy: 97.0780 %
Test Epoch [48/50], Validation Loss: 0.3116, Validation Accuracy:
91.51%
Epoch: 48 Learning rate: 1.9723207287312153e-05
Epoch [49/50],Training Loss: 0.0867, Training Accuracy: 97.0460 %
Test Epoch [49/50], Validation Loss: 0.3121, Validation Accuracy:
91.58%
Epoch: 49 Learning rate: 4.943169062963242e-06
Epoch [50/50],Training Loss: 0.0830, Training Accuracy: 97.1860 %
```

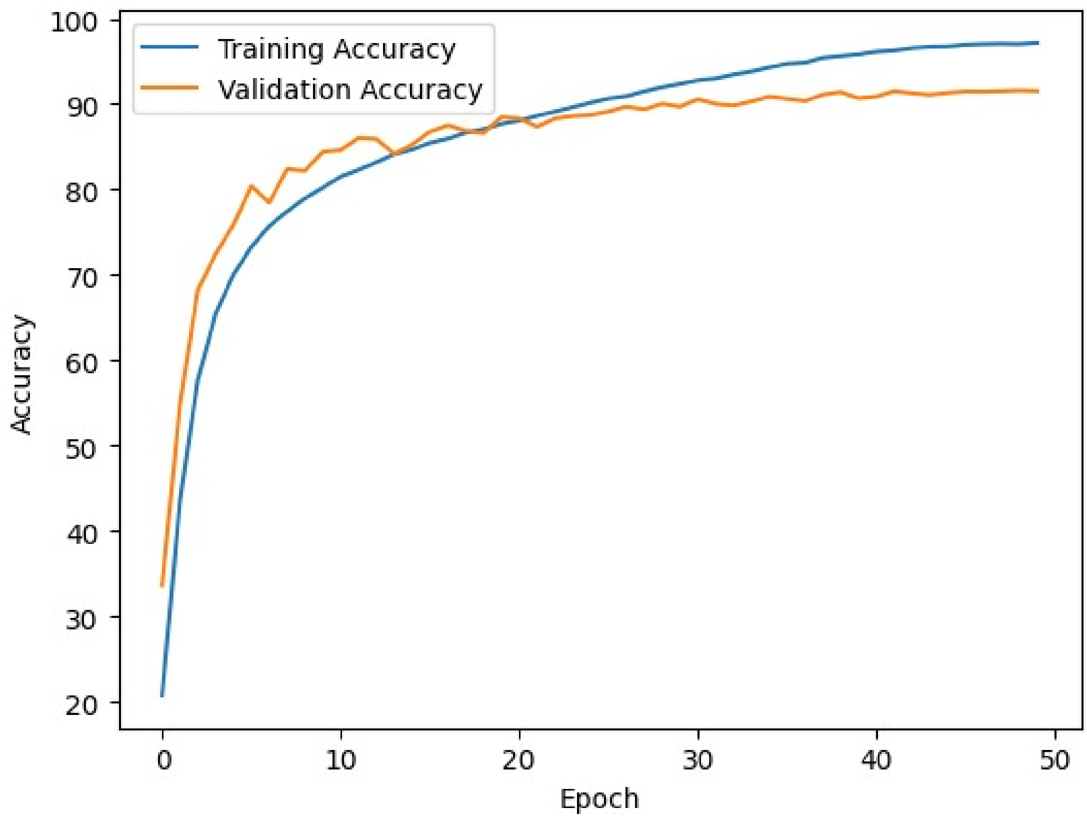
```
Test Epoch [50/50], Validation Loss: 0.3121, Validation Accuracy:
91.53%
Epoch: 50 Learning rate: 1e-08
```

```python
training_acc = []
for tensor in train_acc:
    training_acc.append(tensor.double().tolist())

# Create a list of epoch numbers
epochs = list(range(len(train_loss)))

# Plot the training loss and validation loss
plt.plot(epochs, train_loss, label='Training Loss')
plt.plot(epochs, v_loss, label='Validation Loss')

# Add a legend and axis labels
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')

# Show the plot
plt.show()
```

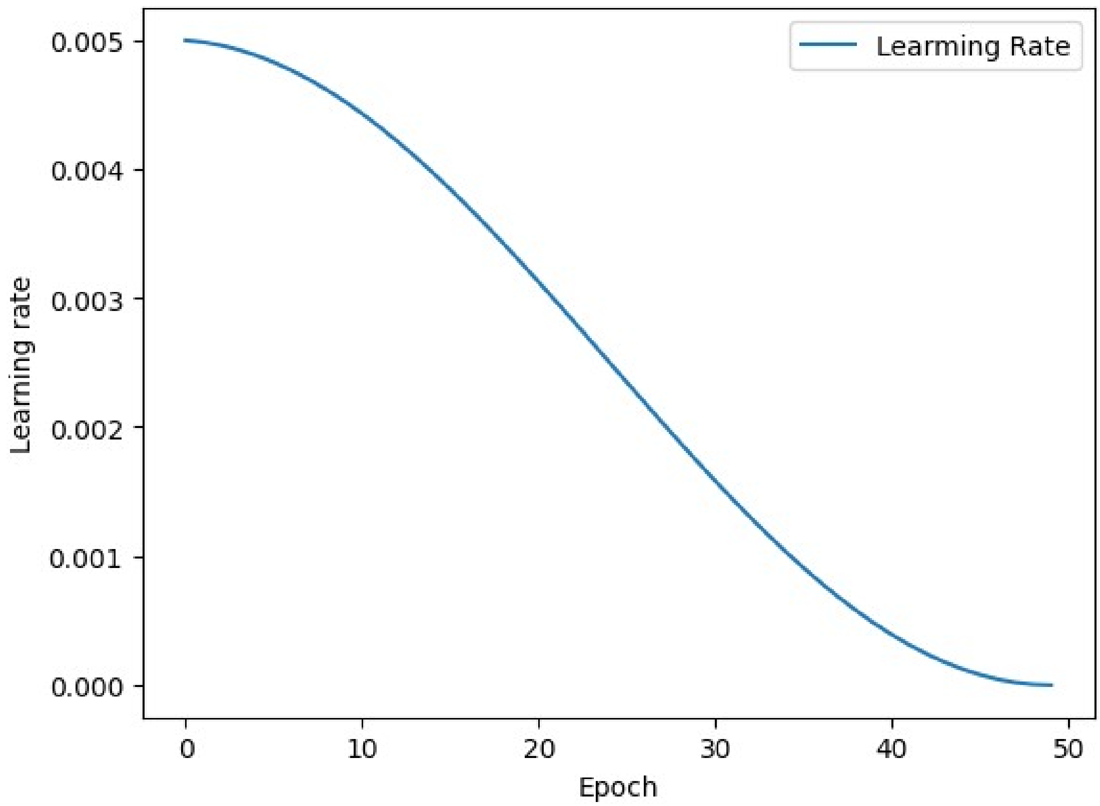



```python
# Plot the training accuracy and validation accuracy
plt.plot(epochs, training_acc, label='Training Accuracy')
```

```
plt.plot(epochs, v_acc, label='Validation Accuracy')

# Add a legend and axis labels
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Accuracy')

# Show the plot
plt.show()
```




```
# Plot the training accuracy and validation accuracy
plt.plot(epochs, learn_rate, label='Learming Rate')
# Add a legend and axis labels
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Learning rate')

# Show the plot
plt.show()
```



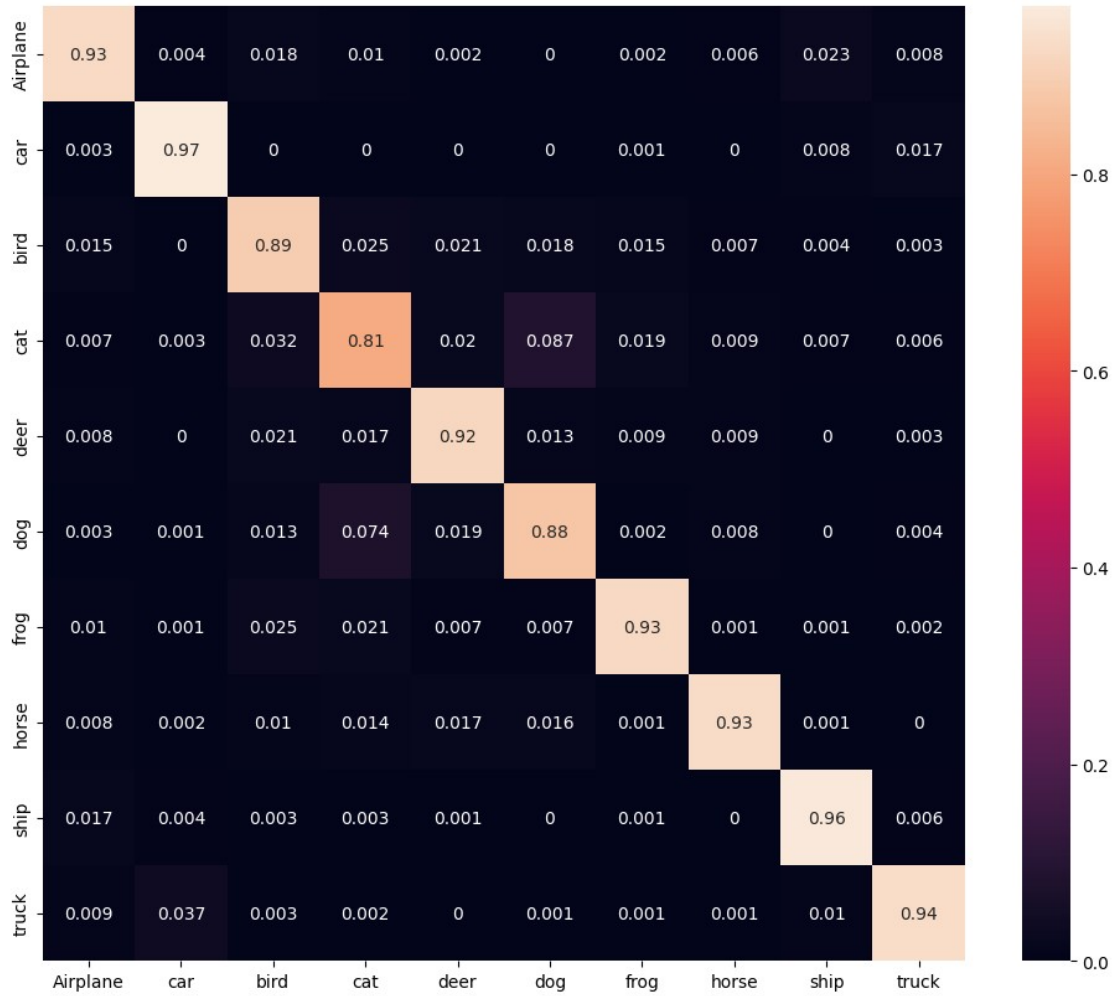

```
# Create a dictionary containing all the necessary information
checkpoint = {
    'model_state_dict': net.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    'epoch': num_epochs,
    'accuracy': epoch_acc,
    'loss': epoch_loss
}

# Save the checkpoint to a file
torch.save(checkpoint, path)

# Load the saved data
checkpoint = torch.load(path)

# Extract the model state dictionary, optimizer state dictionary,
epoch number, accuracy, and loss
model_state_dict = checkpoint['model_state_dict']
optimizer_state_dict = checkpoint['optimizer_state_dict']
epoch = checkpoint['epoch']
accuracy = checkpoint['accuracy']
loss = checkpoint['loss']

# Load the saved state dictionaries into the model and optimizer
net.load_state_dict(model_state_dict)
```

```
optimizer.load_state_dict(optimizer_state_dict)

# Print all parameters of the model
for name, param in net.named_parameters():
    print(name, param.shape)
print(f"Loaded checkpoint from epoch {epoch} with accuracy
{accuracy}")
```

```
conv1.weight torch.Size([32, 3, 3, 3])
conv1.bias torch.Size([32])
bn1.weight torch.Size([32])
bn1.bias torch.Size([32])
conv2.weight torch.Size([64, 32, 3, 3])
conv2.bias torch.Size([64])
bn2.weight torch.Size([64])
bn2.bias torch.Size([64])
conv3.weight torch.Size([128, 64, 3, 3])
conv3.bias torch.Size([128])
bn3.weight torch.Size([128])
bn3.bias torch.Size([128])
conv4.weight torch.Size([256, 128, 3, 3])
conv4.bias torch.Size([256])
bn4.weight torch.Size([256])
bn4.bias torch.Size([256])
conv5.weight torch.Size([512, 256, 3, 3])
conv5.bias torch.Size([512])
bn5.weight torch.Size([512])
bn5.bias torch.Size([512])
conv6.weight torch.Size([1024, 512, 3, 3])
conv6.bias torch.Size([1024])
bn6.weight torch.Size([1024])
bn6.bias torch.Size([1024])
fc1.weight torch.Size([512, 16384])
fc1.bias torch.Size([512])
fc2.weight torch.Size([10, 512])
fc2.bias torch.Size([10])
Loaded checkpoint from epoch 50 with accuracy 97.186
```

```
y_pred = []
y_true = []

# set model to eval mode and move to GPU if available
net.eval()
device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
net.to(device)

# iterate over test data
for inputs, labels in test_loader:
    inputs, labels = inputs.to(device), labels.to(device)
```

```
    with torch.no_grad():
        output = net(inputs) # Feed Network

        output = (torch.max(torch.exp(output), 1)
[1]).data.cpu().numpy()
        y_pred.extend(output) # Save Prediction

        labels = labels.data.cpu().numpy()
        y_true.extend(labels) # Save Truth

# constant for classes
classes = ('Airplane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog',
'horse', 'ship', 'truck')

# Build confusion matrix
cf_matrix = confusion_matrix(y_true, y_pred)
df_cm = pd.DataFrame(cf_matrix / np.sum(cf_matrix, axis=1)[:, None],
index = [i for i in classes],
                     columns = [i for i in classes])
plt.figure(figsize = (12,10))
sn.heatmap(df_cm, annot=True)
plt.savefig('outputjugal.png')
```


Airplane
car
bird
cat
deer
dog
frog
horse
ship
truck
0.93 0.004 0.018 0.01 0.002 0 0.002 0.006 0.023 0.008
0.003 0.97 0 0 0 0 0.001 0 0.008 0.017
0.015 0 0.89 0.025 0.021 0.018 0.015 0.007 0.004 0.003
0.007 0.003 0.032 0.81 0.02 0.087 0.019 0.009 0.007 0.006
0.008 0 0.021 0.017 0.92 0.013 0.009 0.009 0 0.003
0.003 0.001 0.013 0.074 0.019 0.88 0.002 0.008 0 0.004
0.01 0.001 0.025 0.021 0.007 0.007 0.93 0.001 0.001 0.002
0.008 0.002 0.01 0.014 0.017 0.016 0.001 0.93 0.001 0
0.017 0.004 0.003 0.003 0.001 0 0.001 0 0.96 0.006
0.009 0.037 0.003 0.002 0 0.001 0.001 0.001 0.01 0.94
Airplane
car
bird
cat
deer
dog
frog
horse
ship
truck
0.8
0.6
0.4
0.2
0.0