In [ ]:

```python
import torch
from torchvision import datasets, transforms,models
import torchvision.utils as vutils
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.nn.functional as F
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import numpy as np
from torch.utils.data.dataset import random_split
from torchsummary import summary
import time
import seaborn as sns
# import seaborn as sns

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
# device = "cpu"
```

In [ ]:

```python
device
```

Out[ ]:

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

I have provided visualizations at the bottom of the notebook. I have included some explanations for how I planned & constructed my model

In [ ]:

```python
transform = transforms.Compose([
    transforms.ToTensor(),  # Convert images to PyTorch tensors
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),  # Normalize the images
    transforms.RandomHorizontalFlip(),  # Randomly flip the images horizontally wit
    transforms.RandomRotation(10),
    transforms.GaussianBlur()
])


train_data = datasets.CIFAR10(
    root='data',
    train=True,
    download=True,
    transform=transform
)

test_data = datasets.CIFAR10(
    root='data',
    train=False,
    download=True,
    transform=transform
)

batch_size = 64
total_size = len(train_data)

# Determine the split ratio
train_ratio = 0.9
```

```
validation_ratio = 1 - train_ratio

# Calculate lengths
train_length = int(total_size * train_ratio)
validation_length = total_size - train_length  # ensures no data point is left out

# Split the dataset
train_dataset, valid_dataset = random_split(train_data, lengths=[train_length, vali
# train_dataset, valid_dataset = random_split(train_data, lengths=[55000, 5000])
trainloader = DataLoader(train_data,batch_size=50)
testloader = DataLoader(test_data,batch_size=50)

valid_loader = DataLoader(
    dataset=valid_dataset,
    batch_size=batch_size,
    drop_last=False,
    num_workers=3,
    shuffle=False
)
```

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

I decided to experiment with modularizing my convolutions. I defined a 'longblock' class as a container for my deep convolutions. 'Shortblock' is for shorter but wider convolutions (bigger kernel and less channels)

After experimenting, I found that adding skip connections vastly improved performance. The original model had the long & shortblocks, as well as batch norm. The test performance wasn't good, so I added dropout and it brought it above 80%

In [ ]:
```
class Longblock(nn.Module):
    """
    Takes input with 3 channels. Follows shortblock
    Outputs
    """
    def __init__(self,in_channels,out_channels,first=False):
        super(Longblock,self).__init__()
        self.first = first
        #Test convolution, just ads more channels
        self.conv1 = nn.Conv2d(in_channels=in_channels, out_channels=128, kernel_si
        self.conv2 = nn.Conv2d(in_channels=128, out_channels=160, kernel_size=3, pa
        self.conv3 = nn.Conv2d(in_channels=160, out_channels=out_channels, kernel_s
        self.bn1 = nn.BatchNorm2d(num_features=128)
        self.bn2 = nn.BatchNorm2d(num_features=160)
        # self.bn3 = nn.BatchNorm2d(num_features=3)
        self.adjust_channels = nn.Conv2d(in_channels=in_channels, out_channels=out_
        self.in_channels = in_channels
        self.out_channels = out_channels
        self.dropout = nn.Dropout(p=0.2)
    def forward(self, x):
        #For every step, apply RELU & Batch norm
        identity = x
        identity = self.adjust_channels(identity)
        x = F.relu(self.bn1(self.conv1(x)))
```

```python
        x = F.relu(self.bn2(self.conv2(x)))
        # x = F.relu(self.bn3(self.conv3(x)))
        x = self.conv3(x)
        # x = F.max_pool2d(x,kernel_size=2)
        x += identity
        x = self.dropout(x)

        return x

class Shortblock(nn.Module):
    """
    Takes input with 3 channels
    Fatter than the longblock.
    """
    def __init__(self,in_channels,out_channels):
        super(Shortblock,self).__init__()
        self.conv1 = nn.Conv2d(in_channels=in_channels, out_channels=20, kernel_siz
        self.conv2 = nn.Conv2d(in_channels=20, out_channels=64, kernel_size=7, padd
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=out_channels, kernel_si
        self.bn1 = nn.BatchNorm2d(num_features=20)
        self.bn2 = nn.BatchNorm2d(num_features=64)
        self.bn3 = nn.BatchNorm2d(num_features=out_channels)
        self.dropout = nn.Dropout(p=0.2)

    def forward(self, x):
        #For every step, apply RELU & Batch norm. Then dropout and maxpool
        x = F.relu(self.bn1(self.conv1(x)))
        x = F.relu(self.bn2(self.conv2(x)))
        x = F.relu(self.bn3(self.conv3(x)))
        x = self.dropout(x)
        x = F.max_pool2d(x,kernel_size=3)
        return x

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        #Resizes to 3 channels from 1
        # self.resizeconv = nn.Conv2d(in_channels=3,out_channels=3,kernel_size=3,pa
        self.shortblock = Shortblock(3,128)
        self.longblock1 = Longblock(128,128)
        self.longblock2 = Longblock(128,256)
        self.longblock3 = Longblock(256,512)
        self.longblock4 = Longblock(512,256)
        self.shortblock2 = Shortblock(256,128)
        self.longblock5 = Longblock(128,64)

        self.Linear1 = nn.Linear(576,100)
        self.Linear2 = nn.Linear(100,500)
        self.Linear3 = nn.Linear(500,10)

    def forward(self,x):

        #Convolution Blocks
        #This totals 10 convolutions. Resize is to add more channels initially, the
        # x = self.resizeconv(x)
        x = self.shortblock(x)
```

```python
        x = self.longblock1(x)
        x = self.longblock2(x)
        x = self.longblock3(x)
        x = self.longblock4(x)
        x = self.shortblock2(x)
        x = self.longblock5(x)
        self.dropout = nn.Dropout(0.2)
        # print(x.shape)

        #Flatten and run through linear layers
        x = torch.flatten(x, start_dim=1)
        x = F.relu(self.Linear1(x))
        x = F.relu(self.Linear2(x))
        x = self.dropout(x)
        x = self.Linear3(x)
        return x
```

In [ ]:

```python
print(Model())
```

```
Model(
  (shortblock): Shortblock(
    (conv1): Conv2d(3, 20, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
    (conv2): Conv2d(20, 64, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
    (conv3): Conv2d(64, 128, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
    (bn1): BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
    (bn3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    (dropout): Dropout(p=0.2, inplace=False)
  )
  (longblock1): Longblock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv2): Conv2d(128, 160, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv3): Conv2d(160, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    (bn2): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    (adjust_channels): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
    (dropout): Dropout(p=0.2, inplace=False)
  )
  (longblock2): Longblock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv2): Conv2d(128, 160, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv3): Conv2d(160, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    (bn2): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    (adjust_channels): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1))
    (dropout): Dropout(p=0.2, inplace=False)
  )
  (longblock3): Longblock(
    (conv1): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv2): Conv2d(128, 160, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv3): Conv2d(160, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    (bn2): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    (adjust_channels): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1))
    (dropout): Dropout(p=0.2, inplace=False)
  )
  (longblock4): Longblock(
    (conv1): Conv2d(512, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv2): Conv2d(128, 160, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv3): Conv2d(160, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    (bn2): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    (adjust_channels): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (dropout): Dropout(p=0.2, inplace=False)
```

```
  )
  (shortblock2): Shortblock(
    (conv1): Conv2d(256, 20, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
    (conv2): Conv2d(20, 64, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
    (conv3): Conv2d(64, 128, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
    (bn1): BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
    (bn3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    (dropout): Dropout(p=0.2, inplace=False)
  )
  (longblock5): Longblock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv2): Conv2d(128, 160, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv3): Conv2d(160, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    (bn2): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    (adjust_channels): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1))
    (dropout): Dropout(p=0.2, inplace=False)
  )
  (Linear1): Linear(in_features=576, out_features=100, bias=True)
  (Linear2): Linear(in_features=100, out_features=500, bias=True)
  (Linear3): Linear(in_features=500, out_features=10, bias=True)
)
```

In [ ]:

```
summary(Model().to(device),(3,32,32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 20, 32, 32]           2,960
       BatchNorm2d-2           [-1, 20, 32, 32]              40
            Conv2d-3           [-1, 64, 32, 32]          62,784
       BatchNorm2d-4           [-1, 64, 32, 32]             128
            Conv2d-5          [-1, 128, 32, 32]         401,536
       BatchNorm2d-6          [-1, 128, 32, 32]             256
           Dropout-7          [-1, 128, 32, 32]               0
        Shortblock-8          [-1, 128, 10, 10]               0
            Conv2d-9          [-1, 128, 10, 10]          16,512
           Conv2d-10          [-1, 128, 10, 10]         147,584
      BatchNorm2d-11          [-1, 128, 10, 10]             256
           Conv2d-12          [-1, 160, 10, 10]         184,480
      BatchNorm2d-13          [-1, 160, 10, 10]             320
           Conv2d-14          [-1, 128, 10, 10]         184,448
          Dropout-15          [-1, 128, 10, 10]               0
        Longblock-16          [-1, 128, 10, 10]               0
           Conv2d-17          [-1, 256, 10, 10]          33,024
           Conv2d-18          [-1, 128, 10, 10]         147,584
      BatchNorm2d-19          [-1, 128, 10, 10]             256
           Conv2d-20          [-1, 160, 10, 10]         184,480
      BatchNorm2d-21          [-1, 160, 10, 10]             320
           Conv2d-22          [-1, 256, 10, 10]         368,896
          Dropout-23          [-1, 256, 10, 10]               0
        Longblock-24          [-1, 256, 10, 10]               0
           Conv2d-25          [-1, 512, 10, 10]         131,584
           Conv2d-26          [-1, 128, 10, 10]         295,040
      BatchNorm2d-27          [-1, 128, 10, 10]             256
           Conv2d-28          [-1, 160, 10, 10]         184,480
      BatchNorm2d-29          [-1, 160, 10, 10]             320
           Conv2d-30          [-1, 512, 10, 10]         737,792
          Dropout-31          [-1, 512, 10, 10]               0
        Longblock-32          [-1, 512, 10, 10]               0
           Conv2d-33          [-1, 256, 10, 10]         131,328
           Conv2d-34          [-1, 128, 10, 10]         589,952
      BatchNorm2d-35          [-1, 128, 10, 10]             256
           Conv2d-36          [-1, 160, 10, 10]         184,480
      BatchNorm2d-37          [-1, 160, 10, 10]             320
           Conv2d-38          [-1, 256, 10, 10]         368,896
          Dropout-39          [-1, 256, 10, 10]               0
        Longblock-40          [-1, 256, 10, 10]               0
           Conv2d-41           [-1, 20, 10, 10]         250,900
      BatchNorm2d-42           [-1, 20, 10, 10]              40
           Conv2d-43           [-1, 64, 10, 10]          62,784
      BatchNorm2d-44           [-1, 64, 10, 10]             128
           Conv2d-45          [-1, 128, 10, 10]         401,536
      BatchNorm2d-46          [-1, 128, 10, 10]             256
          Dropout-47          [-1, 128, 10, 10]               0
       Shortblock-48            [-1, 128, 3, 3]               0
           Conv2d-49             [-1, 64, 3, 3]           8,256
           Conv2d-50            [-1, 128, 3, 3]         147,584
      BatchNorm2d-51            [-1, 128, 3, 3]             256
           Conv2d-52            [-1, 160, 3, 3]         184,480
      BatchNorm2d-53            [-1, 160, 3, 3]             320
```

```
          Conv2d-54             [-1, 64, 3, 3]          92,224
         Dropout-55             [-1, 64, 3, 3]               0
       Longblock-56             [-1, 64, 3, 3]               0
          Linear-57                  [-1, 100]          57,700
          Linear-58                  [-1, 500]          50,500
          Linear-59                   [-1, 10]           5,010
================================================================
Total params: 5,622,542
Trainable params: 5,622,542
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 10.18
Params size (MB): 21.45
Estimated Total Size (MB): 31.64
----------------------------------------------------------------
```

In [ ]:

```
random_seed = 1
batch_size = 256
learning_rate = 0.03
num_epochs = 15
num_classes = 10
def accuracy(model, data_loader, device):
    with torch.no_grad():
        model = model.train()
        true_pred = 0
        tot_samples = 0
        for imgs, labels in data_loader:
            imgs = imgs.to(device)
            labels = labels.to(device)
            logits = model(imgs)
            _, label_pred = torch.max(logits, axis=1)
            true_pred += (label_pred==labels).sum()
            tot_samples += labels.shape[0]
        acc = (true_pred/float(tot_samples))*100
    return acc
```

In [ ]:

```
torch.manual_seed(random_seed)
model = Model()
model = model.to(device)

optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=2, gamma=0.5)
#For storing data on losses
train_loss= []

start = time.time()
for epoch in range(num_epochs):
    model = model.train()
    for batch_idx, (imgs, labels) in enumerate(trainloader):
        imgs = imgs.to(device)
        labels = labels.to(device)

        ## Forward Propagation - extract features and classify
        logits = model(imgs)
        loss = F.cross_entropy(logits, labels)
```

```python
        #zero out the gradients
        optimizer.zero_grad()
        #estimate new gradients
        loss.backward()
        #update parameters
        optimizer.step()

        if not (batch_idx + 1) % 100:
            print(
                f"Epoch: {epoch + 1:03d}/{num_epochs:03d} | "
                f"Batch: {batch_idx + 1:03d}/{len(trainloader):03d} | "
                f"Loss: {loss:.4f}"
            )

    # Tracking the Learning Rate Scheduler
    prev_lr = optimizer.param_groups[0]["lr"]
    scheduler.step()
    current_lr = optimizer.param_groups[0]["lr"]
    print(f"Epoch: {epoch+1:03d} Learning Rate {prev_lr:.8f} -> {current_lr:.8f}")

    # Evaluate Performance after each epoch
    model = model.eval()
    tr_acc = accuracy(model, trainloader, device)
    valid_acc = accuracy(model, valid_loader, device)
    print(f"Train Accuracy: {tr_acc:0.3f}")
    print(f"Validation Accuracy: {valid_acc:0.3f}")
    if epoch % 5 == 0:
        ts_acc = accuracy(model, testloader, device)
        print(f"Test Accuracy: {ts_acc:0.3f}")

    print(f"Time elapsed so far: {(time.time() - start) / 60:.2f} min")
    train_loss.append(loss)
print(f"Total Train Time: {(time.time() - start) / 60:.2f} min")
```

```
Epoch: 001/015 | Batch: 100/1000 | Loss: 1.8576
Epoch: 001/015 | Batch: 200/1000 | Loss: 1.7261
Epoch: 001/015 | Batch: 300/1000 | Loss: 1.6687
Epoch: 001/015 | Batch: 400/1000 | Loss: 1.5434
Epoch: 001/015 | Batch: 500/1000 | Loss: 1.2419
Epoch: 001/015 | Batch: 600/1000 | Loss: 1.3140
Epoch: 001/015 | Batch: 700/1000 | Loss: 1.6638
Epoch: 001/015 | Batch: 800/1000 | Loss: 1.1280
Epoch: 001/015 | Batch: 900/1000 | Loss: 1.4253
Epoch: 001/015 | Batch: 1000/1000 | Loss: 1.4685
Epoch: 001 Learning Rate 0.03000000 -> 0.03000000
Train Accuracy: 56.016
Validation Accuracy: 56.160
Test Accuracy: 56.000
Time elapsed so far: 0.78 min
Epoch: 002/015 | Batch: 100/1000 | Loss: 1.5867
Epoch: 002/015 | Batch: 200/1000 | Loss: 1.2620
Epoch: 002/015 | Batch: 300/1000 | Loss: 1.2938
Epoch: 002/015 | Batch: 400/1000 | Loss: 1.1416
Epoch: 002/015 | Batch: 500/1000 | Loss: 0.8978
Epoch: 002/015 | Batch: 600/1000 | Loss: 1.0315
Epoch: 002/015 | Batch: 700/1000 | Loss: 1.3266
Epoch: 002/015 | Batch: 800/1000 | Loss: 0.8658
Epoch: 002/015 | Batch: 900/1000 | Loss: 1.4317
Epoch: 002/015 | Batch: 1000/1000 | Loss: 1.2024
Epoch: 002 Learning Rate 0.03000000 -> 0.01500000
Train Accuracy: 65.932
Validation Accuracy: 66.360
Time elapsed so far: 1.51 min
Epoch: 003/015 | Batch: 100/1000 | Loss: 1.1343
Epoch: 003/015 | Batch: 200/1000 | Loss: 1.0333
Epoch: 003/015 | Batch: 300/1000 | Loss: 0.9750
Epoch: 003/015 | Batch: 400/1000 | Loss: 0.7618
Epoch: 003/015 | Batch: 500/1000 | Loss: 0.5869
Epoch: 003/015 | Batch: 600/1000 | Loss: 0.6248
Epoch: 003/015 | Batch: 700/1000 | Loss: 0.8990
Epoch: 003/015 | Batch: 800/1000 | Loss: 0.8073
Epoch: 003/015 | Batch: 900/1000 | Loss: 1.1276
Epoch: 003/015 | Batch: 1000/1000 | Loss: 1.0206
Epoch: 003 Learning Rate 0.01500000 -> 0.01500000
Train Accuracy: 73.822
Validation Accuracy: 73.820
Time elapsed so far: 2.22 min
Epoch: 004/015 | Batch: 100/1000 | Loss: 1.0532
Epoch: 004/015 | Batch: 200/1000 | Loss: 0.9303
Epoch: 004/015 | Batch: 300/1000 | Loss: 0.7329
Epoch: 004/015 | Batch: 400/1000 | Loss: 0.7059
Epoch: 004/015 | Batch: 500/1000 | Loss: 0.4901
Epoch: 004/015 | Batch: 600/1000 | Loss: 0.6889
Epoch: 004/015 | Batch: 700/1000 | Loss: 1.0142
Epoch: 004/015 | Batch: 800/1000 | Loss: 0.7893
Epoch: 004/015 | Batch: 900/1000 | Loss: 0.7046
Epoch: 004/015 | Batch: 1000/1000 | Loss: 0.8285
Epoch: 004 Learning Rate 0.01500000 -> 0.00750000
Train Accuracy: 76.318
Validation Accuracy: 76.040
```

```
Time elapsed so far: 2.94 min
Epoch: 005/015 | Batch: 100/1000 | Loss: 1.0377
Epoch: 005/015 | Batch: 200/1000 | Loss: 0.8578
Epoch: 005/015 | Batch: 300/1000 | Loss: 0.6407
Epoch: 005/015 | Batch: 400/1000 | Loss: 0.6001
Epoch: 005/015 | Batch: 500/1000 | Loss: 0.5147
Epoch: 005/015 | Batch: 600/1000 | Loss: 0.5106
Epoch: 005/015 | Batch: 700/1000 | Loss: 0.7989
Epoch: 005/015 | Batch: 800/1000 | Loss: 0.6127
Epoch: 005/015 | Batch: 900/1000 | Loss: 0.7235
Epoch: 005/015 | Batch: 1000/1000 | Loss: 0.8392
Epoch: 005 Learning Rate 0.00750000 -> 0.00750000
Train Accuracy: 79.652
Validation Accuracy: 80.320
Time elapsed so far: 3.67 min
Epoch: 006/015 | Batch: 100/1000 | Loss: 0.9017
Epoch: 006/015 | Batch: 200/1000 | Loss: 0.8267
Epoch: 006/015 | Batch: 300/1000 | Loss: 0.6443
Epoch: 006/015 | Batch: 400/1000 | Loss: 0.5671
Epoch: 006/015 | Batch: 500/1000 | Loss: 0.4894
Epoch: 006/015 | Batch: 600/1000 | Loss: 0.3716
Epoch: 006/015 | Batch: 700/1000 | Loss: 0.7554
Epoch: 006/015 | Batch: 800/1000 | Loss: 0.6736
Epoch: 006/015 | Batch: 900/1000 | Loss: 0.6263
Epoch: 006/015 | Batch: 1000/1000 | Loss: 0.6891
Epoch: 006 Learning Rate 0.00750000 -> 0.00375000
Train Accuracy: 80.872
Validation Accuracy: 81.780
Test Accuracy: 77.730
Time elapsed so far: 4.44 min
Epoch: 007/015 | Batch: 100/1000 | Loss: 0.7641
Epoch: 007/015 | Batch: 200/1000 | Loss: 0.7283
Epoch: 007/015 | Batch: 300/1000 | Loss: 0.5325
Epoch: 007/015 | Batch: 400/1000 | Loss: 0.5867
Epoch: 007/015 | Batch: 500/1000 | Loss: 0.4014
Epoch: 007/015 | Batch: 600/1000 | Loss: 0.4202
Epoch: 007/015 | Batch: 700/1000 | Loss: 0.7304
Epoch: 007/015 | Batch: 800/1000 | Loss: 0.5312
Epoch: 007/015 | Batch: 900/1000 | Loss: 0.5786
Epoch: 007/015 | Batch: 1000/1000 | Loss: 0.6558
Epoch: 007 Learning Rate 0.00375000 -> 0.00375000
Train Accuracy: 83.194
Validation Accuracy: 83.860
Time elapsed so far: 5.17 min
Epoch: 008/015 | Batch: 100/1000 | Loss: 0.8473
Epoch: 008/015 | Batch: 200/1000 | Loss: 0.5557
Epoch: 008/015 | Batch: 300/1000 | Loss: 0.3619
Epoch: 008/015 | Batch: 400/1000 | Loss: 0.6176
Epoch: 008/015 | Batch: 500/1000 | Loss: 0.3344
Epoch: 008/015 | Batch: 600/1000 | Loss: 0.3872
Epoch: 008/015 | Batch: 700/1000 | Loss: 0.6459
Epoch: 008/015 | Batch: 800/1000 | Loss: 0.4836
Epoch: 008/015 | Batch: 900/1000 | Loss: 0.4790
Epoch: 008/015 | Batch: 1000/1000 | Loss: 0.7204
Epoch: 008 Learning Rate 0.00375000 -> 0.00187500
Train Accuracy: 83.798
```

```
Validation Accuracy: 84.960
Time elapsed so far: 5.88 min
Epoch: 009/015 | Batch: 100/1000 | Loss: 0.6746
Epoch: 009/015 | Batch: 200/1000 | Loss: 0.5347
Epoch: 009/015 | Batch: 300/1000 | Loss: 0.4934
Epoch: 009/015 | Batch: 400/1000 | Loss: 0.4639
Epoch: 009/015 | Batch: 500/1000 | Loss: 0.3056
Epoch: 009/015 | Batch: 600/1000 | Loss: 0.3671
Epoch: 009/015 | Batch: 700/1000 | Loss: 0.5834
Epoch: 009/015 | Batch: 800/1000 | Loss: 0.4388
Epoch: 009/015 | Batch: 900/1000 | Loss: 0.5162
Epoch: 009/015 | Batch: 1000/1000 | Loss: 0.5077
Epoch: 009 Learning Rate 0.00187500 -> 0.00187500
Train Accuracy: 85.042
Validation Accuracy: 86.300
Time elapsed so far: 6.58 min
Epoch: 010/015 | Batch: 100/1000 | Loss: 0.6461
Epoch: 010/015 | Batch: 200/1000 | Loss: 0.5911
Epoch: 010/015 | Batch: 300/1000 | Loss: 0.4761
Epoch: 010/015 | Batch: 400/1000 | Loss: 0.4635
Epoch: 010/015 | Batch: 500/1000 | Loss: 0.3038
Epoch: 010/015 | Batch: 600/1000 | Loss: 0.2460
Epoch: 010/015 | Batch: 700/1000 | Loss: 0.5415
Epoch: 010/015 | Batch: 800/1000 | Loss: 0.5303
Epoch: 010/015 | Batch: 900/1000 | Loss: 0.5549
Epoch: 010/015 | Batch: 1000/1000 | Loss: 0.4717
Epoch: 010 Learning Rate 0.00187500 -> 0.00093750
Train Accuracy: 85.350
Validation Accuracy: 85.940
Time elapsed so far: 7.31 min
Epoch: 011/015 | Batch: 100/1000 | Loss: 0.6829
Epoch: 011/015 | Batch: 200/1000 | Loss: 0.5794
Epoch: 011/015 | Batch: 300/1000 | Loss: 0.4116
Epoch: 011/015 | Batch: 400/1000 | Loss: 0.5402
Epoch: 011/015 | Batch: 500/1000 | Loss: 0.2365
Epoch: 011/015 | Batch: 600/1000 | Loss: 0.2585
Epoch: 011/015 | Batch: 700/1000 | Loss: 0.3914
Epoch: 011/015 | Batch: 800/1000 | Loss: 0.7106
Epoch: 011/015 | Batch: 900/1000 | Loss: 0.5158
Epoch: 011/015 | Batch: 1000/1000 | Loss: 0.5201
Epoch: 011 Learning Rate 0.00093750 -> 0.00093750
Train Accuracy: 86.306
Validation Accuracy: 86.720
Test Accuracy: 81.060
Time elapsed so far: 8.08 min
Epoch: 012/015 | Batch: 100/1000 | Loss: 0.8055
Epoch: 012/015 | Batch: 200/1000 | Loss: 0.5166
Epoch: 012/015 | Batch: 300/1000 | Loss: 0.4148
Epoch: 012/015 | Batch: 400/1000 | Loss: 0.4014
Epoch: 012/015 | Batch: 500/1000 | Loss: 0.2833
Epoch: 012/015 | Batch: 600/1000 | Loss: 0.2760
Epoch: 012/015 | Batch: 700/1000 | Loss: 0.5072
Epoch: 012/015 | Batch: 800/1000 | Loss: 0.4418
Epoch: 012/015 | Batch: 900/1000 | Loss: 0.4441
Epoch: 012/015 | Batch: 1000/1000 | Loss: 0.3717
Epoch: 012 Learning Rate 0.00093750 -> 0.00046875
```

```
Train Accuracy: 86.440
Validation Accuracy: 87.560
Time elapsed so far: 8.78 min
Epoch: 013/015 | Batch: 100/1000 | Loss: 0.7890
Epoch: 013/015 | Batch: 200/1000 | Loss: 0.5606
Epoch: 013/015 | Batch: 300/1000 | Loss: 0.3905
Epoch: 013/015 | Batch: 400/1000 | Loss: 0.3948
Epoch: 013/015 | Batch: 500/1000 | Loss: 0.3171
Epoch: 013/015 | Batch: 600/1000 | Loss: 0.2461
Epoch: 013/015 | Batch: 700/1000 | Loss: 0.4524
Epoch: 013/015 | Batch: 800/1000 | Loss: 0.3789
Epoch: 013/015 | Batch: 900/1000 | Loss: 0.3493
Epoch: 013/015 | Batch: 1000/1000 | Loss: 0.3727
Epoch: 013 Learning Rate 0.00046875 -> 0.00046875
Train Accuracy: 86.846
Validation Accuracy: 86.980
Time elapsed so far: 9.48 min
Epoch: 014/015 | Batch: 100/1000 | Loss: 0.7272
Epoch: 014/015 | Batch: 200/1000 | Loss: 0.4903
Epoch: 014/015 | Batch: 300/1000 | Loss: 0.4136
Epoch: 014/015 | Batch: 400/1000 | Loss: 0.4935
Epoch: 014/015 | Batch: 500/1000 | Loss: 0.1816
Epoch: 014/015 | Batch: 600/1000 | Loss: 0.2322
Epoch: 014/015 | Batch: 700/1000 | Loss: 0.4211
Epoch: 014/015 | Batch: 800/1000 | Loss: 0.5411
Epoch: 014/015 | Batch: 900/1000 | Loss: 0.4382
Epoch: 014/015 | Batch: 1000/1000 | Loss: 0.4898
Epoch: 014 Learning Rate 0.00046875 -> 0.00023437
Train Accuracy: 86.966
Validation Accuracy: 88.160
Time elapsed so far: 10.20 min
Epoch: 015/015 | Batch: 100/1000 | Loss: 0.6864
Epoch: 015/015 | Batch: 200/1000 | Loss: 0.4992
Epoch: 015/015 | Batch: 300/1000 | Loss: 0.3462
Epoch: 015/015 | Batch: 400/1000 | Loss: 0.4666
Epoch: 015/015 | Batch: 500/1000 | Loss: 0.2510
Epoch: 015/015 | Batch: 600/1000 | Loss: 0.3214
Epoch: 015/015 | Batch: 700/1000 | Loss: 0.5002
Epoch: 015/015 | Batch: 800/1000 | Loss: 0.4640
Epoch: 015/015 | Batch: 900/1000 | Loss: 0.4818
Epoch: 015/015 | Batch: 1000/1000 | Loss: 0.4726
Epoch: 015 Learning Rate 0.00023437 -> 0.00023437
Train Accuracy: 86.996
Validation Accuracy: 87.420
Time elapsed so far: 10.91 min
Total Train Time: 10.91 min
```

In [ ]:
```python
model = model.eval()
ts_acc = accuracy(model, testloader, device)
print(f"Test Accuracy: {ts_acc:0.3f}")
```

```
Test Accuracy: 81.000
```

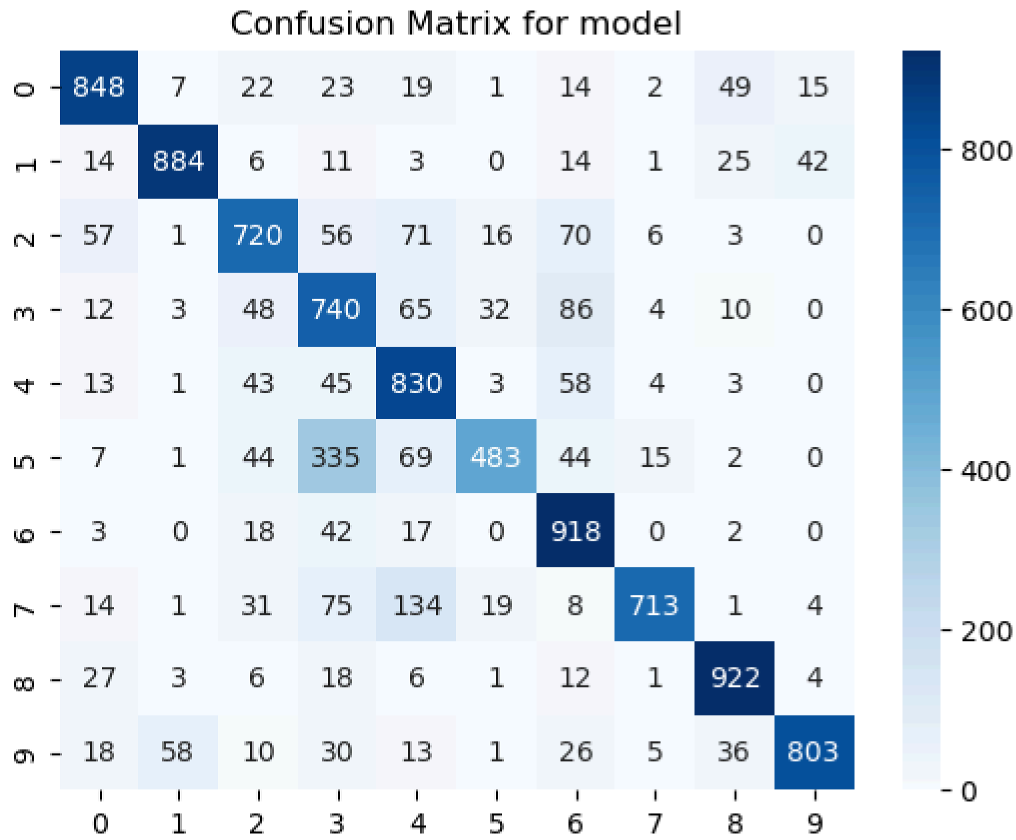
In [ ]:
```python
# torch.save(model.state_dict(), 'model_state_dict.pth')
```

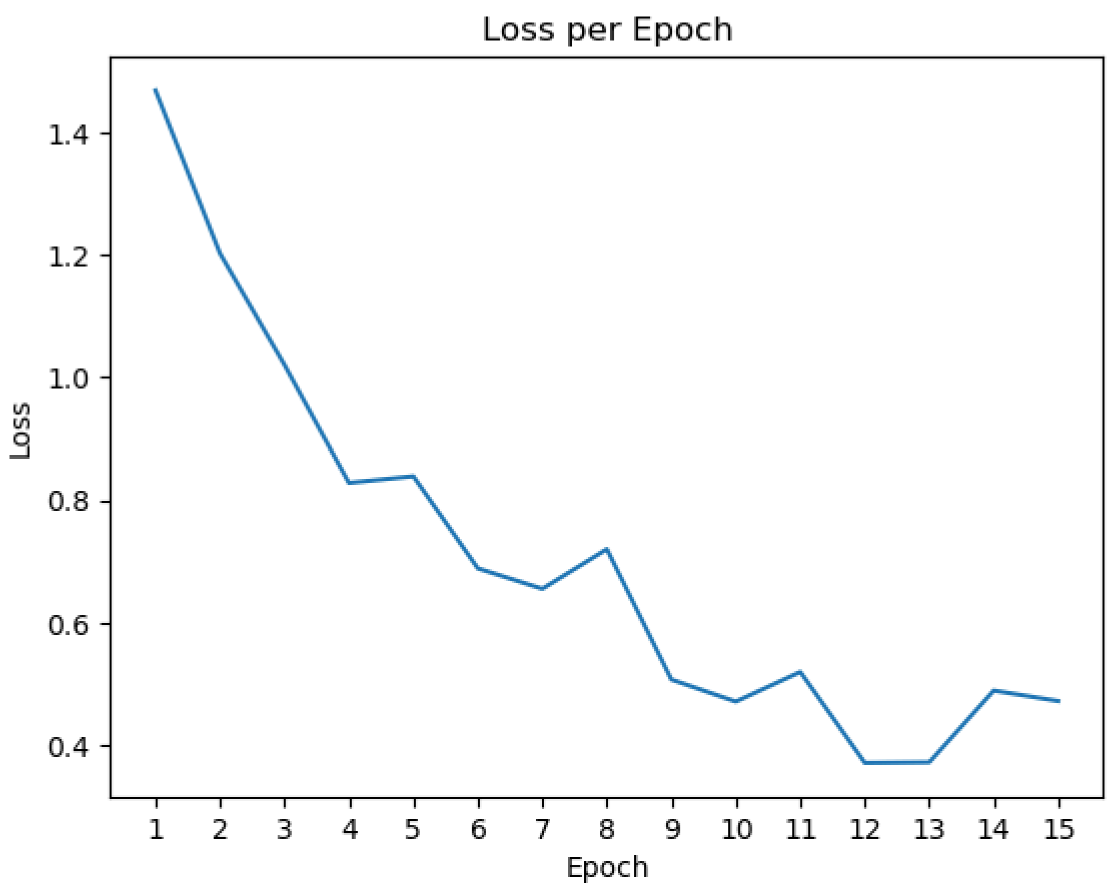
Evaluation:

In [ ]:
```
model.eval()

#Tensors for ypred and ytrue
ypred = torch.tensor([], dtype=torch.long, device=device)
ytrue = torch.tensor([], dtype=torch.long, device=device)

with torch.no_grad():
    for imgs, labels in testloader:
        imgs = imgs.to(device)
        labels = labels.to(device)

        logits = model(imgs)
        _, label_pred = torch.max(logits, axis=1)

        ypred = torch.cat((ypred, label_pred))
        ytrue = torch.cat((ytrue, labels))
ypred = ypred.cpu().numpy()
ytrue = ytrue.cpu().numpy()
```

Visualizations:

In [ ]:
```
cm = confusion_matrix(ytrue,ypred,labels=[*range(10)])
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.title("Confusion Matrix for model")
plt.show()

#Line plot for loss
plt.plot([*range(1,num_epochs+1)],[x.cpu().detach().numpy() for x in train_loss])
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.xticks([*range(1,num_epochs+1)])
plt.title("Loss per Epoch")
plt.show()
```


Confusion Matrix for model
848 7 22 23 19 1 14 2 49 15
14 884 6 11 3 0 14 1 25 42
57 1 720 56 71 16 70 6 3 0
12 3 48 740 65 32 86 4 10 0
13 1 43 45 830 3 58 4 3 0
7 1 44 335 69 483 44 15 2 0
3 0 18 42 17 0 918 0 2 0
14 1 31 75 134 19 8 713 1 4
27 3 6 18 6 1 12 1 922 4
18 58 10 30 13 1 26 5 36 803
0 1 2 3 4 5 6 7 8 9
0
200
400
600
800


Loss per Epoch
Loss
Epoch
0.4
0.6
0.8
1.0
1.2
1.4
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15