main

November 4, 2021

[1]:
```
import torch
import torchvision
from torchvision import transforms, datasets
import matplotlib.pyplot as plt
import numpy as np
from torch.utils.data import Subset
from torch.utils.data import DataLoader
import random
import torch.nn as nn
import torch.nn.functional as F
from torch.optim import Adam
from sklearn.metrics import classification_report
from matplotlib import pyplot as plt
import time
from sklearn.metrics import confusion_matrix
from torch.utils.tensorboard import SummaryWriter
from torch.autograd import Variable
import cv2
%reload_ext tensorboard
```

```
c:\users\hp631\appdata\local\programs\python\python39\lib\site-
packages\setuptools\distutils_patch.py:25: UserWarning: Distutils was imported
before Setuptools. This usage is discouraged and may exhibit undesirable
behaviors or errors. Please use Setuptools' objects directly or at least import
Setuptools first.
  warnings.warn(
```

[2]:
```
def dataset_builder(size_img):

    #reading the train and test datasets and converting them to tensor
    #transforms.ToTensor will scale the images to 0-1 range
    #resize images using transforms.resize
    data_transform = transforms.Compose([
            transforms.Resize((size_img,size_img)),
            transforms.ToTensor()])
    train_dataset = datasets.ImageFolder(root='../data/train/img',
                                         transform=data_transform)
    train_dataloader = DataLoader(train_dataset,batch_size=len(train_dataset),
```

```
                                    shuffle=True,
                                    num_workers=4)

    test_val_dataset=datasets.ImageFolder(root='../data/test/img',
                                          transform=data_transform)
    #splitting test dataset into test+val dataset such that each class in test␣
↪and validation dataset has
    # 500 and 300 images respectively. Also, Shuffling is done before the sets␣
↪are created.
    dataset_size = len(test_val_dataset)
    val_idx=[]
    test_idx=[]
    temp=[]
    for i in range(0, 8000, 800):
        temp=random.sample(range(i,i+800),800)
        val_idx+=temp[0:300]
        test_idx+=temp[300:800]
    val_dataset=Subset(test_val_dataset, val_idx)
    test_dataset=Subset(test_val_dataset,test_idx)

    #normalizing the datasets. first calculate mean and std for each channel in␣
↪each training image and take
    #the average of those values. Then normalize train,test,val datasets using␣
↪these found values for training dataset.
    inputs, classes = next(iter(train_dataloader))
    #print("inputs",len(inputs))
    mean_r=0
    std_r=0
    mean_g=0
    std_g=0
    mean_b=0
    std_b=0
    for img in inputs:
        mean_r+=(img[0].mean()).item()
        std_r+=(img[0].std()).item()
        mean_g+=(img[1].mean()).item()
        std_g+=(img[1].std()).item()
        mean_b+=(img[2].mean()).item()
        std_b+=(img[2].std()).item()
    mean_R=mean_r/(len(inputs))
    mean_G=mean_g/(len(inputs))
    mean_B=mean_b/(len(inputs))
    std_R=std_r/(len(inputs))
    std_G=std_g/(len(inputs))
    std_B=std_b/(len(inputs))
```

```
    #now normalize the datasets using these values of mean and std for each␣
↪channel
    data_transform_normalize = transforms.Compose([
            transforms.Resize((size_img,size_img)), transforms.ToTensor(),
        transforms.Normalize((mean_R,mean_G,mean_B),(std_R,std_G,std_B))])
    train_dataset_normalized = datasets.ImageFolder(root='../data/train/img',
                                            transform=data_transform_normalize)

    val_dataset.dataset.transform = transforms.Compose([transforms.
↪Resize((size_img,size_img)), transforms.ToTensor(),
        transforms.Normalize((mean_R,mean_G,mean_B),(std_R,std_G,std_B))])

    test_dataset.dataset.transform = transforms.Compose([transforms.
↪Resize((size_img,size_img)), transforms.ToTensor(),
        transforms.Normalize((mean_R,mean_G,mean_B),(std_R,std_G,std_B))])

    return train_dataset_normalized, val_dataset, test_dataset
```

[3]:
```
class LeNet5(nn.Module):
    def __init__(self):
        super().__init__()

        #initialize first layer:conv->relu-->maxpooling
        self.conv1=nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5,␣
↪stride=1, padding=0)
        self.relu1=nn.ReLU()
        self.pool1=nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        #initialize second layer:conv->relu-->maxpooling
        self.conv2=nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5,␣
↪stride=1, padding=0)
        self.relu2=nn.ReLU()
        self.pool2=nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        #initialize third layer:FC layer-->relu
        self.fc1=nn.Linear(in_features=16*5*5, out_features=120)
        self.relu3=nn.ReLU()

        #initialize fourth layer:FC layer-->relu
        self.fc2=nn.Linear(in_features=120, out_features=84)
        self.relu4=nn.ReLU()

        #initialize fifth layer:FC layer-->logSoftMax
        self.fc3=nn.Linear(in_features=84, out_features=10)
        self.logSoftmax = nn.LogSoftmax(dim=1)
```

```
    def forward(self,x):
        #passing the input through first set of layers
        x=self.conv1(x)
        x=self.relu1(x)
        x=self.pool1(x)

        #passing output from first layer through second set of layers
        x=self.conv2(x)
        x=self.relu2(x)
        x=self.pool2(x)

        #passing output from second layer through third set of layers
        x=torch.flatten(x,1)
        x=self.fc1(x)
        x=self.relu3(x)

        #passing output from third layer through fourth set of layers
        x=self.fc2(x)
        x=self.relu4(x)

        #passing output from fourth layer through fifth set of layers
        x=self.fc3(x)
        output = self.logSoftmax(x)
        return output
```

[36]:
```
class LeNet5_BN(nn.Module):
    def __init__(self):
        super().__init__()

        #initialize first layer:conv->BatchNormalization-->relu-->maxpooling
        self.conv1=nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5,␣
↪stride=1, padding=0)
        self.bn1=nn.BatchNorm2d(6)
        self.relu1=nn.ReLU()
        self.pool1=nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        #initialize second layer:conv->BatchNormalization-->relu-->maxpooling
        self.conv2=nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5,␣
↪stride=1, padding=0)
        self.bn2=nn.BatchNorm2d(16)
        self.relu2=nn.ReLU()
        self.pool2=nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        #initialize third layer:FC layer-->BatchNormalization-->relu
        self.fc1=nn.Linear(in_features=16*5*5, out_features=120)
        self.bn3=nn.BatchNorm1d(120)
```

```
        self.relu3=nn.ReLU()

        #initialize fourth layer:FC layer-->BatchNormalization-->relu
        self.fc2=nn.Linear(in_features=120, out_features=84)
        self.bn4=nn.BatchNorm1d(84)
        self.relu4=nn.ReLU()

        #initialize fifth layer:FC layer-->logSoftMax
        self.fc3=nn.Linear(in_features=84, out_features=10)
        self.logSoftmax = nn.LogSoftmax(dim=1)

    def forward(self,x):
        #passing the input through first set of layers
        x=self.conv1(x)
        x=self.bn1(x)
        x=self.relu1(x)
        x=self.pool1(x)

        #passing output from first layer through second set of layers
        x=self.conv2(x)
        x=self.bn2(x)
        x=self.relu2(x)
        x=self.pool2(x)

        #passing output from second layer through third set of layers
        x=torch.flatten(x,1)
        x=self.fc1(x)
        x=self.bn3(x)
        x=self.relu3(x)

        #passing output from third layer through fourth set of layers
        x=self.fc2(x)
        x=self.bn4(x)
        x=self.relu4(x)

        #passing output from fourth layer through fifth set of layers
        x=self.fc3(x)
        output = self.logSoftmax(x)
        return output
```

[4]:
```
train_dataset, val_dataset, test_dataset=dataset_builder(32)
print("train size",len(train_dataset))
print("test size",len(test_dataset))
print("val size",len(val_dataset))
```

```
train size 5000
test size 5000
val size 3000
```

[30]:
```
def␣
↪experiment(train_data,test_data,val_data,model_version,epochs,batchsize,learning_rate):
↪

    # initializing the train, validation, and test data loaders with given␣
↪batchsize
    trainDataLoader = DataLoader(train_data, shuffle=True,batch_size=batchsize)
    valDataLoader = DataLoader(val_data, batch_size=batchsize)
    testDataLoader = DataLoader(test_data, batch_size=batchsize)

    # steps per epoch for train and validation set
    trainSteps = len(trainDataLoader.dataset) // batchsize
    valSteps = len(valDataLoader.dataset) // batchsize

    #initialize the model
    print("//...Initializing {} model...//".format(model_version))
    #model= LeNet5()
    model=model_version()

    #defining optimizer and cross-entropy loss function
    optimizer=Adam(model.parameters(),lr=learning_rate)
    criterion=nn.CrossEntropyLoss()

    #Learning rate will reduce by factor of 50% after every step size of␣
↪20epochs
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=20,␣
↪gamma=0.5)

    #empty dict to store losses and accuracy score
    myDict={"train_loss":[],"train_acc":[],"val_loss":[],"val_acc":[]}

    #initialize training the model
    print("//....Training initiated...//")
    start=time.time()

    tb = SummaryWriter()
    for epo in range(0,epochs):
        model.train()
        trainLoss=0
        trainCorrect=0
        valLoss=0
        valCorrect=0
        for (x,y) in trainDataLoader:
            optimizer.zero_grad()
            ytrain_pred=model(x)
            loss=criterion(ytrain_pred,y)
            loss.backward()
```

```
            optimizer.step()
            trainLoss+=loss
            correct_train=(ytrain_pred.argmax(1)==y).type(torch.float).sum().
↪item()
            trainCorrect=correct_train+trainCorrect
        with torch.no_grad():
            model.eval()
            for (x,y) in valDataLoader:
                yval_pred=model(x)
                loss=criterion(yval_pred,y)
                valLoss+=loss
                correct_val=(yval_pred.argmax(1)==y).type(torch.float).sum().
↪item()
                valCorrect=correct_val+valCorrect
        scheduler.step()

        #calculating loss and accuracy for each epoch for both train and val␣
↪sets
        trainLoss_avg=trainLoss/trainSteps
        valLoss_avg=valLoss/valSteps
        train_acc=trainCorrect/(len(trainDataLoader.dataset))
        val_acc=valCorrect/(len(valDataLoader.dataset))

        tb.add_scalar("train/Loss", trainLoss, epo)
        tb.add_scalar("train/Correct", trainCorrect, epo)
        tb.add_scalar("train/Accuracy", train_acc, epo)

        tb.add_scalar("val/Loss", valLoss, epo)
        tb.add_scalar("val/Correct", valCorrect, epo)
        tb.add_scalar("val/Accuracy", val_acc, epo)

        myDict["train_loss"].append(trainLoss_avg.cpu().detach().numpy())
        myDict["val_loss"].append(valLoss_avg.cpu().detach().numpy())
        myDict["train_acc"].append(train_acc)
        myDict["val_acc"].append(val_acc)

        print("//... epoch: {}/{}".format(epo + 1, epochs))
        print("Training loss: {:.4f}, Train accuracy: {:.2f}".
↪format(trainLoss_avg, train_acc))
        print("Validation loss: {:.4f}, Validation accuracy: {:.2f}\n".
↪format(valLoss_avg, val_acc))

    end=time.time()
    print("//...training the network took {:.2f} sec...//".format(end-start))

    #Evaluation of trained model on test set
    with torch.no_grad():
```

```
        model.eval()
        ytest_pred=[]
        ytrue=[]
        for (x,y) in testDataLoader:
            y_pred=model(x)
            ytest_pred.extend(y_pred.argmax(axis=1).numpy())
            ytrue.extend(y.numpy())

    #print("y_pred",ytest_pred)
    #print("y_true",ytrue)

    class_names = train_data.classes
    label_name = {'1': 'airplane', '2':'bird', '3':'car', '4':'cat', '5':
↪'deer', '6':'dog', '7':'horse', '8':'monkey', '9':'ship', '10':'truck'}
    target_label=[]
    for i in class_names:
        target_label.append(label_name[i])
    #print("clas_names",class_names)
    #print("target_label",target_label)

    print(classification_report(ytrue,ytest_pred,target_names=target_label))
    print("//...Confusion Matrix...//")
    print(confusion_matrix(ytrue, ytest_pred, labels=[x for x in range(10)]))

    #plotting loss and acc
    plt.style.use("ggplot")
    plt.figure(2)
    plt.plot(myDict["train_loss"], label="train_loss")
    plt.plot(myDict["val_loss"], label="val_loss")
    plt.plot(myDict["train_acc"], label="train_acc")
    plt.plot(myDict["val_acc"], label="val_acc")
    plt.title("Training Loss and Accuracy: {} model".format(type(model).
↪__name__))
    plt.xlabel("# of Epochs")
    plt.ylabel("Loss/Accuracy")
    plt.legend()
    tb.close()
```

[39]:
```
def␣
↪experiment_L2(train_data,test_data,val_data,model_version,epochs,batchsize,learning_rate):
↪

    # initializing the train, validation, and test data loaders with given␣
↪batchsize
    trainDataLoader = DataLoader(train_data, shuffle=True,batch_size=batchsize)
    valDataLoader = DataLoader(val_data, batch_size=batchsize)
    testDataLoader = DataLoader(test_data, batch_size=batchsize)
```

```
    # steps per epoch for train and validation set
    trainSteps = len(trainDataLoader.dataset) // batchsize
    valSteps = len(valDataLoader.dataset) // batchsize

    #initialize the model
    print("//...Initializing {} model...//".format(model_version))
    #model= LeNet5()
    model=model_version()

    #defining optimizer and cross-entropy loss function. Also Adding L2 loss by␣
↪specifying weight_decay
    optimizer=Adam(model.parameters(),lr=learning_rate,weight_decay=1e-5)
    criterion=nn.CrossEntropyLoss()

    #Learning rate will reduce by factor of 50% after every step size of␣
↪20epochs
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=20,␣
↪gamma=0.5)

    #empty dict to store losses and accuracy score
    myDict={"train_loss":[],"train_acc":[],"val_loss":[],"val_acc":[]}

    #initialize training the model
    print("//....Training initiated...//")
    start=time.time()

    tb = SummaryWriter()
    for epo in range(0,epochs):
        model.train()
        trainLoss=0
        trainCorrect=0
        valLoss=0
        valCorrect=0
        for (x,y) in trainDataLoader:
            optimizer.zero_grad()
            ytrain_pred=model(x)
            loss=criterion(ytrain_pred,y)
            loss.backward()
            optimizer.step()
            trainLoss+=loss
            correct_train=(ytrain_pred.argmax(1)==y).type(torch.float).sum().
↪item()
            trainCorrect=correct_train+trainCorrect
        with torch.no_grad():
            model.eval()
            for (x,y) in valDataLoader:
```

```
                yval_pred=model(x)
                loss=criterion(yval_pred,y)
                valLoss+=loss
                correct_val=(yval_pred.argmax(1)==y).type(torch.float).sum().
↪item()
                valCorrect=correct_val+valCorrect
        scheduler.step()

        #calculating loss and accuracy for each epoch for both train and val␣
↪sets
        trainLoss_avg=trainLoss/trainSteps
        valLoss_avg=valLoss/valSteps
        train_acc=trainCorrect/(len(trainDataLoader.dataset))
        val_acc=valCorrect/(len(valDataLoader.dataset))

        tb.add_scalar("train/Loss", trainLoss, epo)
        tb.add_scalar("train/Correct", trainCorrect, epo)
        tb.add_scalar("train/Accuracy", train_acc, epo)

        tb.add_scalar("val/Loss", valLoss, epo)
        tb.add_scalar("val/Correct", valCorrect, epo)
        tb.add_scalar("val/Accuracy", val_acc, epo)

        myDict["train_loss"].append(trainLoss_avg.cpu().detach().numpy())
        myDict["val_loss"].append(valLoss_avg.cpu().detach().numpy())
        myDict["train_acc"].append(train_acc)
        myDict["val_acc"].append(val_acc)

        print("//... epoch: {}/{}".format(epo + 1, epochs))
        print("Training loss: {:.4f}, Train accuracy: {:.2f}".
↪format(trainLoss_avg, train_acc))
        print("Validation loss: {:.4f}, Validation accuracy: {:.2f}\n".
↪format(valLoss_avg, val_acc))

    end=time.time()
    print("//...training the network took {:.2f} sec...//".format(end-start))

    #Evaluation of trained model on test set
    with torch.no_grad():
        model.eval()
        ytest_pred=[]
        ytrue=[]
        for (x,y) in testDataLoader:
            y_pred=model(x)
            ytest_pred.extend(y_pred.argmax(axis=1).numpy())
            ytrue.extend(y.numpy())
```

```
    #print("y_pred",ytest_pred)
    #print("y_true",ytrue)

    class_names = train_data.classes
    label_name = {'1': 'airplane', '2':'bird', '3':'car', '4':'cat', '5':
↪'deer', '6':'dog', '7':'horse', '8':'monkey', '9':'ship', '10':'truck'}
    target_label=[]
    for i in class_names:
        target_label.append(label_name[i])
    #print("clas_names",class_names)
    #print("target_label",target_label)

    print(classification_report(ytrue,ytest_pred,target_names=target_label))
    print("//...Confusion Matrix...//")
    print(confusion_matrix(ytrue, ytest_pred, labels=[x for x in range(10)]))

    #plotting loss and acc
    plt.style.use("ggplot")
    plt.figure(2)
    plt.plot(myDict["train_loss"], label="train_loss")
    plt.plot(myDict["val_loss"], label="val_loss")
    plt.plot(myDict["train_acc"], label="train_acc")
    plt.plot(myDict["val_acc"], label="val_acc")
    plt.title("Training Loss and Accuracy: {} model".format(type(model).
↪__name__))
    plt.xlabel("# of Epochs")
    plt.ylabel("Loss/Accuracy")
    plt.legend()
    tb.close()
```

[34]:
```
experiment(train_dataset,test_dataset,val_dataset,LeNet5,100,128,0.001)
```

```
//…Initializing <class '__main__.LeNet5'> model…//
//…Training initiated…//
//… epoch: 1/100
Training loss: 2.1893, Train accuracy: 0.22
Validation loss: 1.9880, Validation accuracy: 0.30

//… epoch: 2/100
Training loss: 1.9029, Train accuracy: 0.31
Validation loss: 1.8018, Validation accuracy: 0.35

//… epoch: 3/100
Training loss: 1.7870, Train accuracy: 0.35
Validation loss: 1.7513, Validation accuracy: 0.37

//… epoch: 4/100
Training loss: 1.7053, Train accuracy: 0.38
```

```
Validation loss: 1.6910, Validation accuracy: 0.38

//… epoch: 5/100
Training loss: 1.6349, Train accuracy: 0.40
Validation loss: 1.6592, Validation accuracy: 0.38

//… epoch: 6/100
Training loss: 1.5930, Train accuracy: 0.42
Validation loss: 1.5974, Validation accuracy: 0.42

//… epoch: 7/100
Training loss: 1.5472, Train accuracy: 0.43
Validation loss: 1.5909, Validation accuracy: 0.42

//… epoch: 8/100
Training loss: 1.4899, Train accuracy: 0.44
Validation loss: 1.5511, Validation accuracy: 0.44

//… epoch: 9/100
Training loss: 1.4532, Train accuracy: 0.49
Validation loss: 1.5359, Validation accuracy: 0.45

//… epoch: 10/100
Training loss: 1.4136, Train accuracy: 0.48
Validation loss: 1.5211, Validation accuracy: 0.45

//… epoch: 11/100
Training loss: 1.3705, Train accuracy: 0.50
Validation loss: 1.4971, Validation accuracy: 0.45

//… epoch: 12/100
Training loss: 1.3319, Train accuracy: 0.53
Validation loss: 1.4775, Validation accuracy: 0.47

//… epoch: 13/100
Training loss: 1.2696, Train accuracy: 0.54
Validation loss: 1.5481, Validation accuracy: 0.45

//… epoch: 14/100
Training loss: 1.2559, Train accuracy: 0.55
Validation loss: 1.4819, Validation accuracy: 0.47

//… epoch: 15/100
Training loss: 1.2275, Train accuracy: 0.56
Validation loss: 1.5157, Validation accuracy: 0.47

//… epoch: 16/100
Training loss: 1.2112, Train accuracy: 0.57
```

```
Validation loss: 1.4603, Validation accuracy: 0.49

//… epoch: 17/100
Training loss: 1.1306, Train accuracy: 0.60
Validation loss: 1.4551, Validation accuracy: 0.49

//… epoch: 18/100
Training loss: 1.0986, Train accuracy: 0.61
Validation loss: 1.5240, Validation accuracy: 0.47

//… epoch: 19/100
Training loss: 1.1025, Train accuracy: 0.62
Validation loss: 1.5698, Validation accuracy: 0.46

//… epoch: 20/100
Training loss: 1.1226, Train accuracy: 0.60
Validation loss: 1.5016, Validation accuracy: 0.48

//… epoch: 21/100
Training loss: 0.9601, Train accuracy: 0.66
Validation loss: 1.5028, Validation accuracy: 0.49

//… epoch: 22/100
Training loss: 0.9367, Train accuracy: 0.68
Validation loss: 1.5055, Validation accuracy: 0.50

//… epoch: 23/100
Training loss: 0.9085, Train accuracy: 0.69
Validation loss: 1.5155, Validation accuracy: 0.49

//… epoch: 24/100
Training loss: 0.8792, Train accuracy: 0.70
Validation loss: 1.5492, Validation accuracy: 0.50

//… epoch: 25/100
Training loss: 0.8604, Train accuracy: 0.71
Validation loss: 1.5350, Validation accuracy: 0.50

//… epoch: 26/100
Training loss: 0.8242, Train accuracy: 0.71
Validation loss: 1.5570, Validation accuracy: 0.49

//… epoch: 27/100
Training loss: 0.8049, Train accuracy: 0.72
Validation loss: 1.6138, Validation accuracy: 0.49

//… epoch: 28/100
Training loss: 0.7918, Train accuracy: 0.73
```

```
Validation loss: 1.6222, Validation accuracy: 0.48

//… epoch: 29/100
Training loss: 0.7628, Train accuracy: 0.74
Validation loss: 1.6111, Validation accuracy: 0.49

//… epoch: 30/100
Training loss: 0.7312, Train accuracy: 0.75
Validation loss: 1.6759, Validation accuracy: 0.49

//… epoch: 31/100
Training loss: 0.7362, Train accuracy: 0.75
Validation loss: 1.6729, Validation accuracy: 0.49

//… epoch: 32/100
Training loss: 0.6854, Train accuracy: 0.77
Validation loss: 1.6935, Validation accuracy: 0.49

//… epoch: 33/100
Training loss: 0.6496, Train accuracy: 0.78
Validation loss: 1.7520, Validation accuracy: 0.49

//… epoch: 34/100
Training loss: 0.6538, Train accuracy: 0.79
Validation loss: 1.7642, Validation accuracy: 0.49

//… epoch: 35/100
Training loss: 0.6180, Train accuracy: 0.79
Validation loss: 1.7563, Validation accuracy: 0.49

//… epoch: 36/100
Training loss: 0.5800, Train accuracy: 0.81
Validation loss: 1.8331, Validation accuracy: 0.49

//… epoch: 37/100
Training loss: 0.5553, Train accuracy: 0.82
Validation loss: 1.8434, Validation accuracy: 0.49

//… epoch: 38/100
Training loss: 0.5415, Train accuracy: 0.83
Validation loss: 1.9122, Validation accuracy: 0.49

//… epoch: 39/100
Training loss: 0.5334, Train accuracy: 0.83
Validation loss: 1.9637, Validation accuracy: 0.49

//… epoch: 40/100
Training loss: 0.4907, Train accuracy: 0.84
```

```
Validation loss: 1.9797, Validation accuracy: 0.48

//… epoch: 41/100
Training loss: 0.4518, Train accuracy: 0.86
Validation loss: 1.9894, Validation accuracy: 0.48

//… epoch: 42/100
Training loss: 0.4427, Train accuracy: 0.86
Validation loss: 2.0035, Validation accuracy: 0.48

//… epoch: 43/100
Training loss: 0.4200, Train accuracy: 0.88
Validation loss: 2.0682, Validation accuracy: 0.48

//… epoch: 44/100
Training loss: 0.4019, Train accuracy: 0.88
Validation loss: 2.0815, Validation accuracy: 0.48

//… epoch: 45/100
Training loss: 0.3942, Train accuracy: 0.89
Validation loss: 2.1038, Validation accuracy: 0.48

//… epoch: 46/100
Training loss: 0.3817, Train accuracy: 0.89
Validation loss: 2.1252, Validation accuracy: 0.48

//… epoch: 47/100
Training loss: 0.3768, Train accuracy: 0.89
Validation loss: 2.1500, Validation accuracy: 0.48

//… epoch: 48/100
Training loss: 0.3652, Train accuracy: 0.89
Validation loss: 2.1895, Validation accuracy: 0.48

//… epoch: 49/100
Training loss: 0.3408, Train accuracy: 0.91
Validation loss: 2.2270, Validation accuracy: 0.48

//… epoch: 50/100
Training loss: 0.3400, Train accuracy: 0.91
Validation loss: 2.2517, Validation accuracy: 0.48

//… epoch: 51/100
Training loss: 0.3448, Train accuracy: 0.90
Validation loss: 2.2656, Validation accuracy: 0.47

//… epoch: 52/100
Training loss: 0.3339, Train accuracy: 0.91
```

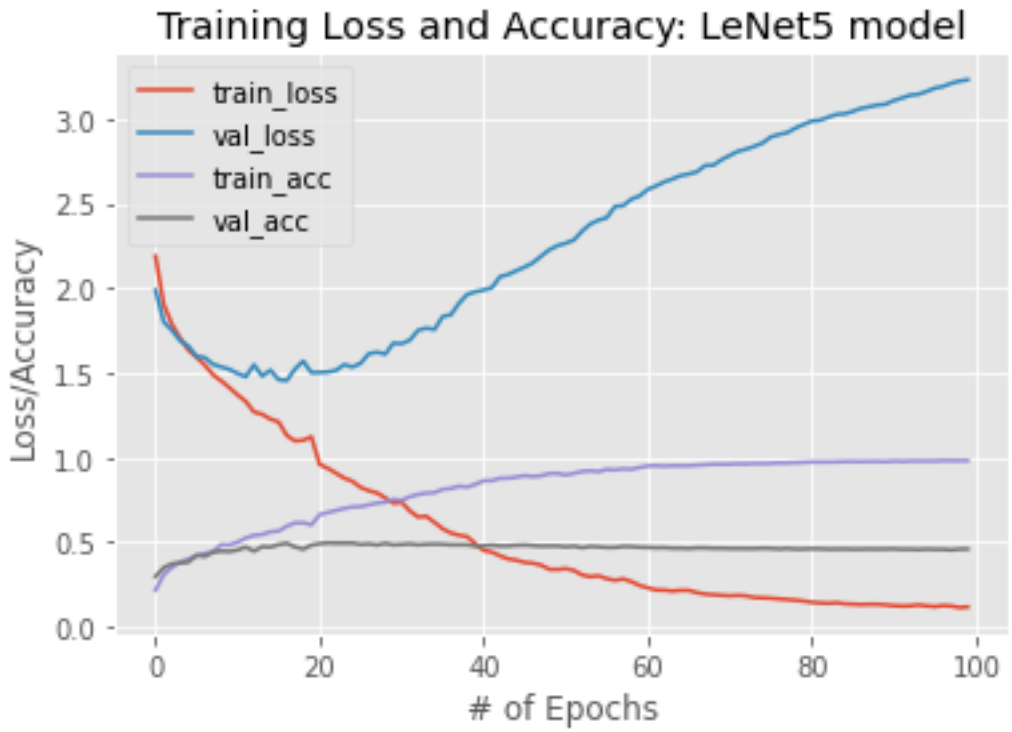
```
Validation loss: 2.2848, Validation accuracy: 0.48

//… epoch: 53/100
Training loss: 0.3075, Train accuracy: 0.92
Validation loss: 2.3376, Validation accuracy: 0.47

//… epoch: 54/100
Training loss: 0.2970, Train accuracy: 0.92
Validation loss: 2.3795, Validation accuracy: 0.48

//… epoch: 55/100
Training loss: 0.3021, Train accuracy: 0.92
Validation loss: 2.4018, Validation accuracy: 0.47

//… epoch: 56/100
Training loss: 0.2858, Train accuracy: 0.93
Validation loss: 2.4166, Validation accuracy: 0.47

//… epoch: 57/100
Training loss: 0.2754, Train accuracy: 0.93
Validation loss: 2.4823, Validation accuracy: 0.47

//… epoch: 58/100
Training loss: 0.2837, Train accuracy: 0.93
Validation loss: 2.4877, Validation accuracy: 0.48

//… epoch: 59/100
Training loss: 0.2685, Train accuracy: 0.93
Validation loss: 2.5258, Validation accuracy: 0.47

//… epoch: 60/100
Training loss: 0.2455, Train accuracy: 0.94
Validation loss: 2.5445, Validation accuracy: 0.47

//… epoch: 61/100
Training loss: 0.2314, Train accuracy: 0.95
Validation loss: 2.5834, Validation accuracy: 0.47

//… epoch: 62/100
Training loss: 0.2190, Train accuracy: 0.95
Validation loss: 2.6038, Validation accuracy: 0.47

//… epoch: 63/100
Training loss: 0.2181, Train accuracy: 0.95
Validation loss: 2.6277, Validation accuracy: 0.47

//… epoch: 64/100
Training loss: 0.2124, Train accuracy: 0.95
```

```
Validation loss: 2.6448, Validation accuracy: 0.47

//… epoch: 65/100
Training loss: 0.2171, Train accuracy: 0.95
Validation loss: 2.6649, Validation accuracy: 0.47

//… epoch: 66/100
Training loss: 0.2161, Train accuracy: 0.95
Validation loss: 2.6741, Validation accuracy: 0.46

//… epoch: 67/100
Training loss: 0.2028, Train accuracy: 0.96
Validation loss: 2.6877, Validation accuracy: 0.47

//… epoch: 68/100
Training loss: 0.1943, Train accuracy: 0.96
Validation loss: 2.7245, Validation accuracy: 0.47

//… epoch: 69/100
Training loss: 0.1906, Train accuracy: 0.96
Validation loss: 2.7265, Validation accuracy: 0.46

//… epoch: 70/100
Training loss: 0.1875, Train accuracy: 0.96
Validation loss: 2.7588, Validation accuracy: 0.47

//… epoch: 71/100
Training loss: 0.1843, Train accuracy: 0.96
Validation loss: 2.7839, Validation accuracy: 0.46

//… epoch: 72/100
Training loss: 0.1871, Train accuracy: 0.96
Validation loss: 2.8084, Validation accuracy: 0.46

//… epoch: 73/100
Training loss: 0.1838, Train accuracy: 0.96
Validation loss: 2.8208, Validation accuracy: 0.46

//… epoch: 74/100
Training loss: 0.1744, Train accuracy: 0.97
Validation loss: 2.8378, Validation accuracy: 0.46

//… epoch: 75/100
Training loss: 0.1725, Train accuracy: 0.96
Validation loss: 2.8574, Validation accuracy: 0.47

//… epoch: 76/100
Training loss: 0.1699, Train accuracy: 0.97
```

```
Validation loss: 2.8930, Validation accuracy: 0.46

//… epoch: 77/100
Training loss: 0.1656, Train accuracy: 0.97
Validation loss: 2.9076, Validation accuracy: 0.46

//… epoch: 78/100
Training loss: 0.1624, Train accuracy: 0.97
Validation loss: 2.9186, Validation accuracy: 0.46

//… epoch: 79/100
Training loss: 0.1583, Train accuracy: 0.97
Validation loss: 2.9455, Validation accuracy: 0.46

//… epoch: 80/100
Training loss: 0.1536, Train accuracy: 0.97
Validation loss: 2.9672, Validation accuracy: 0.46

//… epoch: 81/100
Training loss: 0.1459, Train accuracy: 0.97
Validation loss: 2.9869, Validation accuracy: 0.46

//… epoch: 82/100
Training loss: 0.1426, Train accuracy: 0.97
Validation loss: 2.9919, Validation accuracy: 0.46

//… epoch: 83/100
Training loss: 0.1400, Train accuracy: 0.97
Validation loss: 3.0093, Validation accuracy: 0.46

//… epoch: 84/100
Training loss: 0.1431, Train accuracy: 0.98
Validation loss: 3.0260, Validation accuracy: 0.46

//… epoch: 85/100
Training loss: 0.1371, Train accuracy: 0.98
Validation loss: 3.0300, Validation accuracy: 0.46

//… epoch: 86/100
Training loss: 0.1349, Train accuracy: 0.98
Validation loss: 3.0438, Validation accuracy: 0.46

//… epoch: 87/100
Training loss: 0.1324, Train accuracy: 0.98
Validation loss: 3.0618, Validation accuracy: 0.46

//… epoch: 88/100
Training loss: 0.1341, Train accuracy: 0.98
```

```
Validation loss: 3.0713, Validation accuracy: 0.46

//… epoch: 89/100
Training loss: 0.1341, Train accuracy: 0.98
Validation loss: 3.0813, Validation accuracy: 0.46

//… epoch: 90/100
Training loss: 0.1306, Train accuracy: 0.98
Validation loss: 3.0864, Validation accuracy: 0.46

//… epoch: 91/100
Training loss: 0.1262, Train accuracy: 0.98
Validation loss: 3.1080, Validation accuracy: 0.46

//… epoch: 92/100
Training loss: 0.1241, Train accuracy: 0.98
Validation loss: 3.1230, Validation accuracy: 0.46

//… epoch: 93/100
Training loss: 0.1251, Train accuracy: 0.98
Validation loss: 3.1401, Validation accuracy: 0.46

//… epoch: 94/100
Training loss: 0.1305, Train accuracy: 0.98
Validation loss: 3.1447, Validation accuracy: 0.46

//… epoch: 95/100
Training loss: 0.1246, Train accuracy: 0.98
Validation loss: 3.1621, Validation accuracy: 0.46

//… epoch: 96/100
Training loss: 0.1207, Train accuracy: 0.98
Validation loss: 3.1814, Validation accuracy: 0.46

//… epoch: 97/100
Training loss: 0.1268, Train accuracy: 0.98
Validation loss: 3.1911, Validation accuracy: 0.46

//… epoch: 98/100
Training loss: 0.1239, Train accuracy: 0.98
Validation loss: 3.2100, Validation accuracy: 0.46

//… epoch: 99/100
Training loss: 0.1148, Train accuracy: 0.98
Validation loss: 3.2228, Validation accuracy: 0.46

//… epoch: 100/100
Training loss: 0.1184, Train accuracy: 0.98
```

```
Validation loss: 3.2306, Validation accuracy: 0.46

//…training the network took 879.43 sec…//
              precision    recall  f1-score   support

    airplane       0.64      0.65      0.64       500
       truck       0.53      0.53      0.53       500
        bird       0.35      0.33      0.34       500
         car       0.64      0.55      0.59       500
         cat       0.30      0.32      0.31       500
        deer       0.40      0.42      0.41       500
         dog       0.27      0.26      0.27       500
       horse       0.50      0.48      0.49       500
      monkey       0.33      0.36      0.34       500
        ship       0.64      0.67      0.65       500

    accuracy                           0.46      5000
   macro avg       0.46      0.46      0.46      5000
weighted avg       0.46      0.46      0.46      5000

//…Confusion Matrix…//
[[323  35  17  18  11  22  10   3   8  53]
 [ 35 264  12  78  10   8  13   9  12  59]
 [ 26  10 167   9  72  62  45  23  76  10]
 [ 24  96  16 273  17   9   8   6   8  43]
 [  9  13  56  10 162  55  64  38  78  15]
 [ 12   7  57   4  56 212  48  48  53   3]
 [ 10   5  55   4  81  57 130  60  94   4]
 [  4  14  30   3  33  50  85 240  40   1]
 [  9   7  65   2  72  40  73  49 180   3]
 [ 56  43   5  24  18  12   4   1   4 333]]
```



[38]:
```
experiment(train_dataset,test_dataset,val_dataset,LeNet5_BN,100,128,0.001)
```

```
//…Initializing <class '__main__.LeNet5_BN'> model…//
//…Training initiated…//
//… epoch: 1/100
Training loss: 2.0004, Train accuracy: 0.29
Validation loss: 1.8269, Validation accuracy: 0.38

//… epoch: 2/100
Training loss: 1.6341, Train accuracy: 0.43
Validation loss: 1.6214, Validation accuracy: 0.44

//… epoch: 3/100
Training loss: 1.4449, Train accuracy: 0.50
Validation loss: 1.5258, Validation accuracy: 0.44

//… epoch: 4/100
Training loss: 1.2963, Train accuracy: 0.55
Validation loss: 1.4364, Validation accuracy: 0.48

//… epoch: 5/100
Training loss: 1.1437, Train accuracy: 0.62
Validation loss: 1.4283, Validation accuracy: 0.48
```

```
//… epoch: 6/100
Training loss: 1.0576, Train accuracy: 0.65
Validation loss: 1.4734, Validation accuracy: 0.47

//… epoch: 7/100
Training loss: 0.9535, Train accuracy: 0.68
Validation loss: 1.4347, Validation accuracy: 0.50

//… epoch: 8/100
Training loss: 0.8492, Train accuracy: 0.74
Validation loss: 1.6364, Validation accuracy: 0.46

//… epoch: 9/100
Training loss: 0.8075, Train accuracy: 0.74
Validation loss: 1.5303, Validation accuracy: 0.48

//… epoch: 10/100
Training loss: 0.7378, Train accuracy: 0.77
Validation loss: 1.5850, Validation accuracy: 0.48

//… epoch: 11/100
Training loss: 0.6188, Train accuracy: 0.82
Validation loss: 1.5557, Validation accuracy: 0.49

//… epoch: 12/100
Training loss: 0.5384, Train accuracy: 0.84
Validation loss: 1.7014, Validation accuracy: 0.47

//… epoch: 13/100
Training loss: 0.5202, Train accuracy: 0.85
Validation loss: 1.7364, Validation accuracy: 0.46

//… epoch: 14/100
Training loss: 0.4248, Train accuracy: 0.88
Validation loss: 1.7711, Validation accuracy: 0.47

//… epoch: 15/100
Training loss: 0.3563, Train accuracy: 0.91
Validation loss: 1.8509, Validation accuracy: 0.47

//… epoch: 16/100
Training loss: 0.2912, Train accuracy: 0.94
Validation loss: 1.9018, Validation accuracy: 0.47

//… epoch: 17/100
Training loss: 0.2945, Train accuracy: 0.93
Validation loss: 1.9657, Validation accuracy: 0.47
```

```
//… epoch: 18/100
Training loss: 0.2489, Train accuracy: 0.95
Validation loss: 1.9951, Validation accuracy: 0.47

//… epoch: 19/100
Training loss: 0.3013, Train accuracy: 0.92
Validation loss: 2.0768, Validation accuracy: 0.47

//… epoch: 20/100
Training loss: 0.2279, Train accuracy: 0.94
Validation loss: 2.0927, Validation accuracy: 0.47

//… epoch: 21/100
Training loss: 0.1859, Train accuracy: 0.97
Validation loss: 2.0807, Validation accuracy: 0.47

//… epoch: 22/100
Training loss: 0.1787, Train accuracy: 0.96
Validation loss: 2.1159, Validation accuracy: 0.47

//… epoch: 23/100
Training loss: 0.1343, Train accuracy: 0.98
Validation loss: 2.1636, Validation accuracy: 0.47

//… epoch: 24/100
Training loss: 0.1090, Train accuracy: 0.99
Validation loss: 2.2064, Validation accuracy: 0.47

//… epoch: 25/100
Training loss: 0.1024, Train accuracy: 0.99
Validation loss: 2.2353, Validation accuracy: 0.47

//… epoch: 26/100
Training loss: 0.1211, Train accuracy: 0.99
Validation loss: 2.2390, Validation accuracy: 0.47

//… epoch: 27/100
Training loss: 0.1306, Train accuracy: 0.98
Validation loss: 2.3102, Validation accuracy: 0.47

//… epoch: 28/100
Training loss: 0.1263, Train accuracy: 0.98
Validation loss: 2.3004, Validation accuracy: 0.47

//… epoch: 29/100
Training loss: 0.1426, Train accuracy: 0.98
Validation loss: 2.3217, Validation accuracy: 0.46
```

```
//… epoch: 30/100
Training loss: 0.1165, Train accuracy: 0.98
Validation loss: 2.4132, Validation accuracy: 0.46

//… epoch: 31/100
Training loss: 0.1316, Train accuracy: 0.97
Validation loss: 2.3990, Validation accuracy: 0.46

//… epoch: 32/100
Training loss: 0.0909, Train accuracy: 0.99
Validation loss: 2.3791, Validation accuracy: 0.46

//… epoch: 33/100
Training loss: 0.0634, Train accuracy: 1.00
Validation loss: 2.4255, Validation accuracy: 0.46

//… epoch: 34/100
Training loss: 0.0757, Train accuracy: 0.99
Validation loss: 2.4597, Validation accuracy: 0.46

//… epoch: 35/100
Training loss: 0.1175, Train accuracy: 0.99
Validation loss: 2.4815, Validation accuracy: 0.47

//… epoch: 36/100
Training loss: 0.0988, Train accuracy: 0.98
Validation loss: 2.5337, Validation accuracy: 0.46

//… epoch: 37/100
Training loss: 0.0676, Train accuracy: 1.00
Validation loss: 2.5713, Validation accuracy: 0.46

//… epoch: 38/100
Training loss: 0.1229, Train accuracy: 0.98
Validation loss: 2.5859, Validation accuracy: 0.46

//… epoch: 39/100
Training loss: 0.1001, Train accuracy: 0.98
Validation loss: 2.5449, Validation accuracy: 0.47

//… epoch: 40/100
Training loss: 0.0849, Train accuracy: 0.99
Validation loss: 2.6349, Validation accuracy: 0.46

//… epoch: 41/100
Training loss: 0.0811, Train accuracy: 0.98
Validation loss: 2.5659, Validation accuracy: 0.46
```

```
//… epoch: 42/100
Training loss: 0.0562, Train accuracy: 1.00
Validation loss: 2.6020, Validation accuracy: 0.46

//… epoch: 43/100
Training loss: 0.0543, Train accuracy: 1.00
Validation loss: 2.5712, Validation accuracy: 0.48

//… epoch: 44/100
Training loss: 0.0609, Train accuracy: 1.00
Validation loss: 2.6131, Validation accuracy: 0.47

//… epoch: 45/100
Training loss: 0.0849, Train accuracy: 0.99
Validation loss: 2.6647, Validation accuracy: 0.46

//… epoch: 46/100
Training loss: 0.0645, Train accuracy: 0.99
Validation loss: 2.6266, Validation accuracy: 0.46

//… epoch: 47/100
Training loss: 0.0574, Train accuracy: 1.00
Validation loss: 2.6173, Validation accuracy: 0.47

//… epoch: 48/100
Training loss: 0.0490, Train accuracy: 1.00
Validation loss: 2.6104, Validation accuracy: 0.47

//… epoch: 49/100
Training loss: 0.0470, Train accuracy: 1.00
Validation loss: 2.6562, Validation accuracy: 0.47

//… epoch: 50/100
Training loss: 0.0608, Train accuracy: 0.99
Validation loss: 2.6214, Validation accuracy: 0.47

//… epoch: 51/100
Training loss: 0.0466, Train accuracy: 1.00
Validation loss: 2.6553, Validation accuracy: 0.47

//… epoch: 52/100
Training loss: 0.0435, Train accuracy: 1.00
Validation loss: 2.6477, Validation accuracy: 0.46

//… epoch: 53/100
Training loss: 0.0464, Train accuracy: 0.99
Validation loss: 2.6555, Validation accuracy: 0.47
```

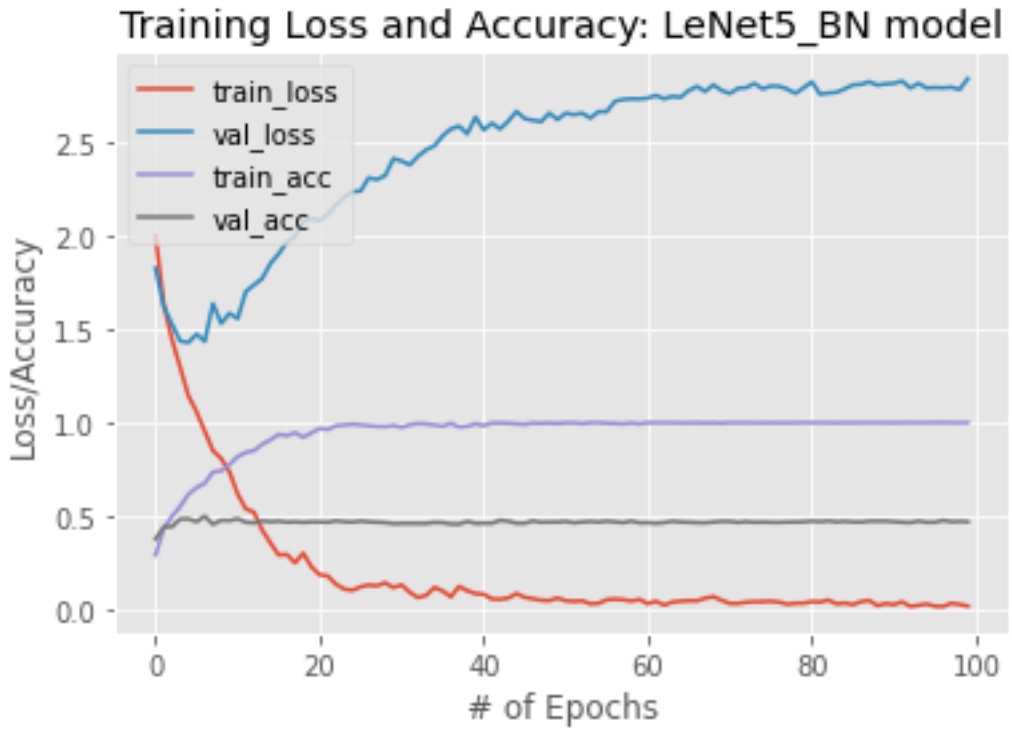
```
//… epoch: 54/100
Training loss: 0.0301, Train accuracy: 1.00
Validation loss: 2.6291, Validation accuracy: 0.47

//… epoch: 55/100
Training loss: 0.0324, Train accuracy: 1.00
Validation loss: 2.6636, Validation accuracy: 0.47

//… epoch: 56/100
Training loss: 0.0518, Train accuracy: 1.00
Validation loss: 2.6636, Validation accuracy: 0.47

//… epoch: 57/100
Training loss: 0.0555, Train accuracy: 1.00
Validation loss: 2.7193, Validation accuracy: 0.47

//… epoch: 58/100
Training loss: 0.0523, Train accuracy: 0.99
Validation loss: 2.7290, Validation accuracy: 0.46

//… epoch: 59/100
Training loss: 0.0444, Train accuracy: 1.00
Validation loss: 2.7328, Validation accuracy: 0.47

//… epoch: 60/100
Training loss: 0.0535, Train accuracy: 0.99
Validation loss: 2.7316, Validation accuracy: 0.46

//… epoch: 61/100
Training loss: 0.0308, Train accuracy: 1.00
Validation loss: 2.7362, Validation accuracy: 0.46

//… epoch: 62/100
Training loss: 0.0434, Train accuracy: 1.00
Validation loss: 2.7505, Validation accuracy: 0.46

//… epoch: 63/100
Training loss: 0.0233, Train accuracy: 1.00
Validation loss: 2.7327, Validation accuracy: 0.46

//… epoch: 64/100
Training loss: 0.0395, Train accuracy: 1.00
Validation loss: 2.7456, Validation accuracy: 0.47

//… epoch: 65/100
Training loss: 0.0441, Train accuracy: 1.00
Validation loss: 2.7402, Validation accuracy: 0.47
```

```
//… epoch: 66/100
Training loss: 0.0469, Train accuracy: 1.00
Validation loss: 2.7773, Validation accuracy: 0.47

//… epoch: 67/100
Training loss: 0.0466, Train accuracy: 1.00
Validation loss: 2.7989, Validation accuracy: 0.46

//… epoch: 68/100
Training loss: 0.0602, Train accuracy: 1.00
Validation loss: 2.7723, Validation accuracy: 0.46

//… epoch: 69/100
Training loss: 0.0687, Train accuracy: 1.00
Validation loss: 2.8072, Validation accuracy: 0.47

//… epoch: 70/100
Training loss: 0.0490, Train accuracy: 1.00
Validation loss: 2.7774, Validation accuracy: 0.47

//… epoch: 71/100
Training loss: 0.0318, Train accuracy: 1.00
Validation loss: 2.7608, Validation accuracy: 0.47

//… epoch: 72/100
Training loss: 0.0318, Train accuracy: 1.00
Validation loss: 2.7886, Validation accuracy: 0.47

//… epoch: 73/100
Training loss: 0.0387, Train accuracy: 1.00
Validation loss: 2.7920, Validation accuracy: 0.47

//… epoch: 74/100
Training loss: 0.0413, Train accuracy: 1.00
Validation loss: 2.8145, Validation accuracy: 0.47

//… epoch: 75/100
Training loss: 0.0418, Train accuracy: 1.00
Validation loss: 2.7859, Validation accuracy: 0.47

//… epoch: 76/100
Training loss: 0.0438, Train accuracy: 1.00
Validation loss: 2.8010, Validation accuracy: 0.47

//… epoch: 77/100
Training loss: 0.0391, Train accuracy: 1.00
Validation loss: 2.7997, Validation accuracy: 0.47
```

```
//… epoch: 78/100
Training loss: 0.0281, Train accuracy: 1.00
Validation loss: 2.7846, Validation accuracy: 0.47

//… epoch: 79/100
Training loss: 0.0326, Train accuracy: 1.00
Validation loss: 2.7615, Validation accuracy: 0.47

//… epoch: 80/100
Training loss: 0.0337, Train accuracy: 1.00
Validation loss: 2.7913, Validation accuracy: 0.47

//… epoch: 81/100
Training loss: 0.0418, Train accuracy: 1.00
Validation loss: 2.8234, Validation accuracy: 0.47

//… epoch: 82/100
Training loss: 0.0398, Train accuracy: 1.00
Validation loss: 2.7577, Validation accuracy: 0.47

//… epoch: 83/100
Training loss: 0.0506, Train accuracy: 1.00
Validation loss: 2.7633, Validation accuracy: 0.47

//… epoch: 84/100
Training loss: 0.0299, Train accuracy: 1.00
Validation loss: 2.7671, Validation accuracy: 0.47

//… epoch: 85/100
Training loss: 0.0338, Train accuracy: 1.00
Validation loss: 2.7853, Validation accuracy: 0.47

//… epoch: 86/100
Training loss: 0.0258, Train accuracy: 1.00
Validation loss: 2.8067, Validation accuracy: 0.47

//… epoch: 87/100
Training loss: 0.0417, Train accuracy: 1.00
Validation loss: 2.8098, Validation accuracy: 0.47

//… epoch: 88/100
Training loss: 0.0486, Train accuracy: 1.00
Validation loss: 2.8239, Validation accuracy: 0.47

//… epoch: 89/100
Training loss: 0.0199, Train accuracy: 1.00
Validation loss: 2.8054, Validation accuracy: 0.47
```

```
//… epoch: 90/100
Training loss: 0.0317, Train accuracy: 1.00
Validation loss: 2.8123, Validation accuracy: 0.47

//… epoch: 91/100
Training loss: 0.0259, Train accuracy: 1.00
Validation loss: 2.8150, Validation accuracy: 0.47

//… epoch: 92/100
Training loss: 0.0406, Train accuracy: 1.00
Validation loss: 2.8277, Validation accuracy: 0.47

//… epoch: 93/100
Training loss: 0.0160, Train accuracy: 1.00
Validation loss: 2.7902, Validation accuracy: 0.46

//… epoch: 94/100
Training loss: 0.0222, Train accuracy: 1.00
Validation loss: 2.8152, Validation accuracy: 0.47

//… epoch: 95/100
Training loss: 0.0293, Train accuracy: 1.00
Validation loss: 2.7899, Validation accuracy: 0.47

//… epoch: 96/100
Training loss: 0.0173, Train accuracy: 1.00
Validation loss: 2.7930, Validation accuracy: 0.47

//… epoch: 97/100
Training loss: 0.0152, Train accuracy: 1.00
Validation loss: 2.7906, Validation accuracy: 0.48

//… epoch: 98/100
Training loss: 0.0332, Train accuracy: 1.00
Validation loss: 2.7950, Validation accuracy: 0.47

//… epoch: 99/100
Training loss: 0.0272, Train accuracy: 1.00
Validation loss: 2.7816, Validation accuracy: 0.47

//… epoch: 100/100
Training loss: 0.0178, Train accuracy: 1.00
Validation loss: 2.8399, Validation accuracy: 0.47

//…training the network took 919.00 sec…//
              precision    recall  f1-score   support

    airplane       0.69      0.71      0.70       500
```

```
       truck       0.53      0.45      0.49       500
        bird       0.38      0.31      0.34       500
         car       0.67      0.63      0.65       500
         cat       0.28      0.28      0.28       500
        deer       0.40      0.42      0.41       500
         dog       0.26      0.29      0.28       500
       horse       0.45      0.51      0.48       500
      monkey       0.33      0.34      0.33       500
        ship       0.64      0.68      0.66       500

    accuracy                           0.46      5000
   macro avg       0.46      0.46      0.46      5000
weighted avg       0.46      0.46      0.46      5000

//…Confusion Matrix…//
[[353  25  15  17   6   9  10  10   5  50]
 [ 28 224  11  86  23  20  15  16   7  70]
 [ 24   5 154   7  60  63  66  31  73  17]
 [ 16  73  18 313  18   4  13  10  11  24]
 [  9  11  42  15 141  82  72  44  72  12]
 [ 11   5  46   4  65 212  40  60  50   7]
 [  7  10  49   3  74  55 147  82  68   5]
 [  4  14  30   4  33  31  83 255  44   2]
 [  6   6  27   3  70  49 112  56 168   3]
 [ 52  46   9  12  13   8   6   7   8 339]]
```



[42]:
```
experiment_L2(train_dataset,test_dataset,val_dataset,LeNet5,100,128,0.001)
```

```
//…Initializing <class '__main__.LeNet5'> model…//
//…Training initiated…//
//… epoch: 1/100
Training loss: 2.1840, Train accuracy: 0.21
Validation loss: 2.0263, Validation accuracy: 0.28

//… epoch: 2/100
Training loss: 1.8906, Train accuracy: 0.31
Validation loss: 1.8038, Validation accuracy: 0.35

//… epoch: 3/100
Training loss: 1.7384, Train accuracy: 0.37
Validation loss: 1.7840, Validation accuracy: 0.36

//… epoch: 4/100
Training loss: 1.6290, Train accuracy: 0.40
Validation loss: 1.6395, Validation accuracy: 0.40

//… epoch: 5/100
Training loss: 1.5634, Train accuracy: 0.43
Validation loss: 1.6028, Validation accuracy: 0.41

//… epoch: 6/100
Training loss: 1.5148, Train accuracy: 0.45
Validation loss: 1.5749, Validation accuracy: 0.41

//… epoch: 7/100
Training loss: 1.4913, Train accuracy: 0.45
Validation loss: 1.5488, Validation accuracy: 0.44

//… epoch: 8/100
Training loss: 1.4514, Train accuracy: 0.46
Validation loss: 1.5401, Validation accuracy: 0.44

//… epoch: 9/100
Training loss: 1.4151, Train accuracy: 0.48
Validation loss: 1.5380, Validation accuracy: 0.44

//… epoch: 10/100
Training loss: 1.3781, Train accuracy: 0.51
Validation loss: 1.5531, Validation accuracy: 0.43

//… epoch: 11/100
```

```
Training loss: 1.3777, Train accuracy: 0.50
Validation loss: 1.4995, Validation accuracy: 0.45

//… epoch: 12/100
Training loss: 1.3007, Train accuracy: 0.53
Validation loss: 1.5467, Validation accuracy: 0.46

//… epoch: 13/100
Training loss: 1.3060, Train accuracy: 0.53
Validation loss: 1.5058, Validation accuracy: 0.45

//… epoch: 14/100
Training loss: 1.2497, Train accuracy: 0.54
Validation loss: 1.5459, Validation accuracy: 0.46

//… epoch: 15/100
Training loss: 1.2066, Train accuracy: 0.55
Validation loss: 1.5125, Validation accuracy: 0.47

//… epoch: 16/100
Training loss: 1.1822, Train accuracy: 0.58
Validation loss: 1.5244, Validation accuracy: 0.48

//… epoch: 17/100
Training loss: 1.1506, Train accuracy: 0.59
Validation loss: 1.5289, Validation accuracy: 0.47

//… epoch: 18/100
Training loss: 1.1144, Train accuracy: 0.61
Validation loss: 1.4936, Validation accuracy: 0.49

//… epoch: 19/100
Training loss: 1.0703, Train accuracy: 0.61
Validation loss: 1.4977, Validation accuracy: 0.48

//… epoch: 20/100
Training loss: 1.0458, Train accuracy: 0.63
Validation loss: 1.5291, Validation accuracy: 0.47

//… epoch: 21/100
Training loss: 0.9673, Train accuracy: 0.66
Validation loss: 1.5597, Validation accuracy: 0.48

//… epoch: 22/100
Training loss: 0.9562, Train accuracy: 0.66
Validation loss: 1.5317, Validation accuracy: 0.48

//… epoch: 23/100
```

```
Training loss: 0.9183, Train accuracy: 0.68
Validation loss: 1.5437, Validation accuracy: 0.48

//… epoch: 24/100
Training loss: 0.8843, Train accuracy: 0.68
Validation loss: 1.5576, Validation accuracy: 0.49

//… epoch: 25/100
Training loss: 0.8693, Train accuracy: 0.70
Validation loss: 1.5598, Validation accuracy: 0.48

//… epoch: 26/100
Training loss: 0.8598, Train accuracy: 0.71
Validation loss: 1.6038, Validation accuracy: 0.48

//… epoch: 27/100
Training loss: 0.8292, Train accuracy: 0.71
Validation loss: 1.6029, Validation accuracy: 0.48

//… epoch: 28/100
Training loss: 0.8310, Train accuracy: 0.72
Validation loss: 1.6075, Validation accuracy: 0.48

//… epoch: 29/100
Training loss: 0.7866, Train accuracy: 0.73
Validation loss: 1.6268, Validation accuracy: 0.48

//… epoch: 30/100
Training loss: 0.7474, Train accuracy: 0.74
Validation loss: 1.6721, Validation accuracy: 0.48

//… epoch: 31/100
Training loss: 0.7692, Train accuracy: 0.74
Validation loss: 1.6731, Validation accuracy: 0.48

//… epoch: 32/100
Training loss: 0.7193, Train accuracy: 0.75
Validation loss: 1.7210, Validation accuracy: 0.48

//… epoch: 33/100
Training loss: 0.7156, Train accuracy: 0.76
Validation loss: 1.6966, Validation accuracy: 0.48

//… epoch: 34/100
Training loss: 0.6906, Train accuracy: 0.76
Validation loss: 1.7905, Validation accuracy: 0.48

//… epoch: 35/100
```

```
Training loss: 0.6743, Train accuracy: 0.77
Validation loss: 1.7951, Validation accuracy: 0.47

//… epoch: 36/100
Training loss: 0.6257, Train accuracy: 0.79
Validation loss: 1.7878, Validation accuracy: 0.48

//… epoch: 37/100
Training loss: 0.6168, Train accuracy: 0.79
Validation loss: 1.8189, Validation accuracy: 0.47

//… epoch: 38/100
Training loss: 0.6073, Train accuracy: 0.80
Validation loss: 1.8444, Validation accuracy: 0.48

//… epoch: 39/100
Training loss: 0.5825, Train accuracy: 0.80
Validation loss: 1.8840, Validation accuracy: 0.48

//… epoch: 40/100
Training loss: 0.5741, Train accuracy: 0.81
Validation loss: 1.9016, Validation accuracy: 0.48

//… epoch: 41/100
Training loss: 0.5195, Train accuracy: 0.83
Validation loss: 1.9121, Validation accuracy: 0.48

//… epoch: 42/100
Training loss: 0.4959, Train accuracy: 0.84
Validation loss: 1.9309, Validation accuracy: 0.48

//… epoch: 43/100
Training loss: 0.4945, Train accuracy: 0.85
Validation loss: 1.9457, Validation accuracy: 0.47

//… epoch: 44/100
Training loss: 0.4820, Train accuracy: 0.85
Validation loss: 1.9631, Validation accuracy: 0.48

//… epoch: 45/100
Training loss: 0.4655, Train accuracy: 0.85
Validation loss: 2.0087, Validation accuracy: 0.48

//… epoch: 46/100
Training loss: 0.4502, Train accuracy: 0.86
Validation loss: 2.0580, Validation accuracy: 0.48

//… epoch: 47/100
```

```
Training loss: 0.4613, Train accuracy: 0.86
Validation loss: 2.0415, Validation accuracy: 0.47

//… epoch: 48/100
Training loss: 0.4410, Train accuracy: 0.87
Validation loss: 2.0853, Validation accuracy: 0.47

//… epoch: 49/100
Training loss: 0.4307, Train accuracy: 0.87
Validation loss: 2.0808, Validation accuracy: 0.48

//… epoch: 50/100
Training loss: 0.4149, Train accuracy: 0.87
Validation loss: 2.1275, Validation accuracy: 0.48

//… epoch: 51/100
Training loss: 0.4072, Train accuracy: 0.88
Validation loss: 2.1489, Validation accuracy: 0.47

//… epoch: 52/100
Training loss: 0.3978, Train accuracy: 0.88
Validation loss: 2.1600, Validation accuracy: 0.48

//… epoch: 53/100
Training loss: 0.3877, Train accuracy: 0.89
Validation loss: 2.1768, Validation accuracy: 0.48

//… epoch: 54/100
Training loss: 0.3852, Train accuracy: 0.88
Validation loss: 2.1956, Validation accuracy: 0.48

//… epoch: 55/100
Training loss: 0.3642, Train accuracy: 0.89
Validation loss: 2.2482, Validation accuracy: 0.48

//… epoch: 56/100
Training loss: 0.3564, Train accuracy: 0.89
Validation loss: 2.2775, Validation accuracy: 0.47

//… epoch: 57/100
Training loss: 0.3534, Train accuracy: 0.89
Validation loss: 2.2768, Validation accuracy: 0.48

//… epoch: 58/100
Training loss: 0.3403, Train accuracy: 0.90
Validation loss: 2.3356, Validation accuracy: 0.47

//… epoch: 59/100
```

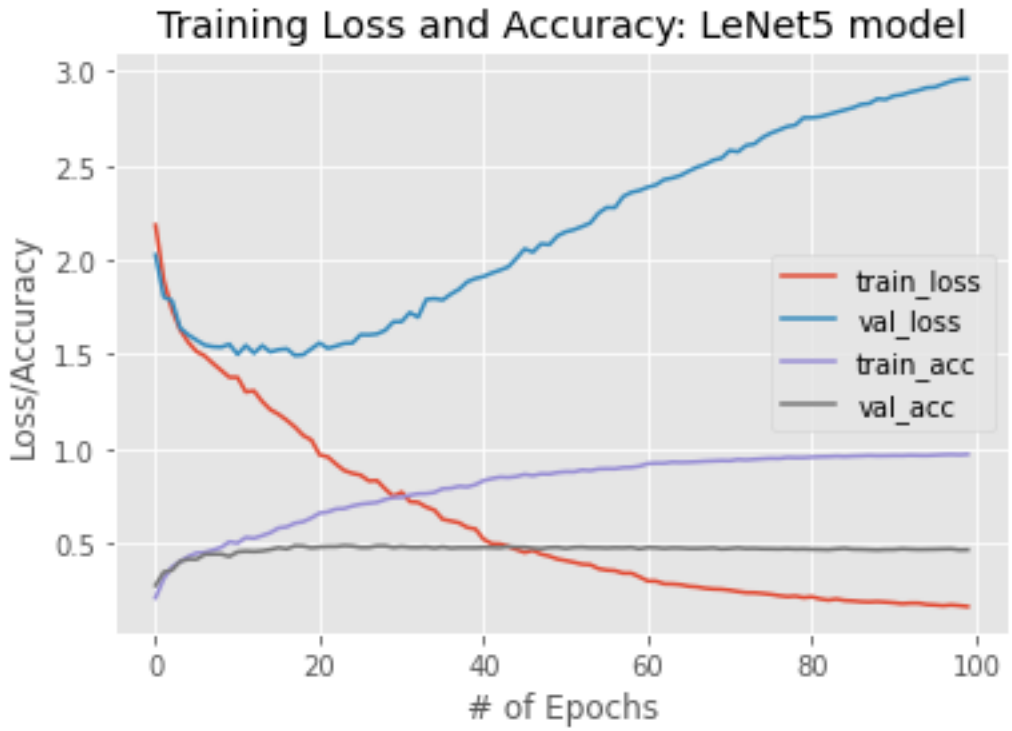
```
Training loss: 0.3404, Train accuracy: 0.90
Validation loss: 2.3579, Validation accuracy: 0.48

//… epoch: 60/100
Training loss: 0.3206, Train accuracy: 0.91
Validation loss: 2.3681, Validation accuracy: 0.47

//… epoch: 61/100
Training loss: 0.2990, Train accuracy: 0.92
Validation loss: 2.3852, Validation accuracy: 0.48

//… epoch: 62/100
Training loss: 0.2980, Train accuracy: 0.92
Validation loss: 2.3974, Validation accuracy: 0.48

//… epoch: 63/100
Training loss: 0.2837, Train accuracy: 0.92
Validation loss: 2.4253, Validation accuracy: 0.47

//… epoch: 64/100
Training loss: 0.2827, Train accuracy: 0.93
Validation loss: 2.4338, Validation accuracy: 0.47

//… epoch: 65/100
Training loss: 0.2792, Train accuracy: 0.93
Validation loss: 2.4472, Validation accuracy: 0.47

//… epoch: 66/100
Training loss: 0.2709, Train accuracy: 0.93
Validation loss: 2.4706, Validation accuracy: 0.47

//… epoch: 67/100
Training loss: 0.2676, Train accuracy: 0.93
Validation loss: 2.4908, Validation accuracy: 0.47

//… epoch: 68/100
Training loss: 0.2603, Train accuracy: 0.93
Validation loss: 2.5064, Validation accuracy: 0.47

//… epoch: 69/100
Training loss: 0.2564, Train accuracy: 0.93
Validation loss: 2.5284, Validation accuracy: 0.47

//… epoch: 70/100
Training loss: 0.2550, Train accuracy: 0.94
Validation loss: 2.5394, Validation accuracy: 0.47

//… epoch: 71/100
```

```
Training loss: 0.2499, Train accuracy: 0.94
Validation loss: 2.5794, Validation accuracy: 0.47

//… epoch: 72/100
Training loss: 0.2430, Train accuracy: 0.94
Validation loss: 2.5728, Validation accuracy: 0.47

//… epoch: 73/100
Training loss: 0.2372, Train accuracy: 0.94
Validation loss: 2.6065, Validation accuracy: 0.47

//… epoch: 74/100
Training loss: 0.2367, Train accuracy: 0.94
Validation loss: 2.6145, Validation accuracy: 0.47

//… epoch: 75/100
Training loss: 0.2330, Train accuracy: 0.95
Validation loss: 2.6485, Validation accuracy: 0.47

//… epoch: 76/100
Training loss: 0.2280, Train accuracy: 0.95
Validation loss: 2.6715, Validation accuracy: 0.47

//… epoch: 77/100
Training loss: 0.2212, Train accuracy: 0.95
Validation loss: 2.6883, Validation accuracy: 0.47

//… epoch: 78/100
Training loss: 0.2163, Train accuracy: 0.95
Validation loss: 2.7061, Validation accuracy: 0.47

//… epoch: 79/100
Training loss: 0.2188, Train accuracy: 0.95
Validation loss: 2.7145, Validation accuracy: 0.47

//… epoch: 80/100
Training loss: 0.2111, Train accuracy: 0.95
Validation loss: 2.7535, Validation accuracy: 0.47

//… epoch: 81/100
Training loss: 0.2154, Train accuracy: 0.96
Validation loss: 2.7536, Validation accuracy: 0.47

//… epoch: 82/100
Training loss: 0.2042, Train accuracy: 0.96
Validation loss: 2.7587, Validation accuracy: 0.47

//… epoch: 83/100
```

```
Training loss: 0.1977, Train accuracy: 0.96
Validation loss: 2.7704, Validation accuracy: 0.47

//… epoch: 84/100
Training loss: 0.2032, Train accuracy: 0.96
Validation loss: 2.7826, Validation accuracy: 0.47

//… epoch: 85/100
Training loss: 0.1957, Train accuracy: 0.96
Validation loss: 2.7943, Validation accuracy: 0.47

//… epoch: 86/100
Training loss: 0.1932, Train accuracy: 0.96
Validation loss: 2.8058, Validation accuracy: 0.47

//… epoch: 87/100
Training loss: 0.1894, Train accuracy: 0.96
Validation loss: 2.8236, Validation accuracy: 0.47

//… epoch: 88/100
Training loss: 0.1879, Train accuracy: 0.96
Validation loss: 2.8286, Validation accuracy: 0.47

//… epoch: 89/100
Training loss: 0.1896, Train accuracy: 0.96
Validation loss: 2.8534, Validation accuracy: 0.46

//… epoch: 90/100
Training loss: 0.1869, Train accuracy: 0.96
Validation loss: 2.8501, Validation accuracy: 0.47

//… epoch: 91/100
Training loss: 0.1834, Train accuracy: 0.96
Validation loss: 2.8692, Validation accuracy: 0.47

//… epoch: 92/100
Training loss: 0.1785, Train accuracy: 0.96
Validation loss: 2.8754, Validation accuracy: 0.47

//… epoch: 93/100
Training loss: 0.1813, Train accuracy: 0.97
Validation loss: 2.8891, Validation accuracy: 0.47

//… epoch: 94/100
Training loss: 0.1812, Train accuracy: 0.96
Validation loss: 2.8988, Validation accuracy: 0.47

//… epoch: 95/100
```

```
Training loss: 0.1742, Train accuracy: 0.96
Validation loss: 2.9122, Validation accuracy: 0.47

//… epoch: 96/100
Training loss: 0.1719, Train accuracy: 0.97
Validation loss: 2.9151, Validation accuracy: 0.47

//… epoch: 97/100
Training loss: 0.1682, Train accuracy: 0.97
Validation loss: 2.9314, Validation accuracy: 0.47

//… epoch: 98/100
Training loss: 0.1720, Train accuracy: 0.97
Validation loss: 2.9471, Validation accuracy: 0.47

//… epoch: 99/100
Training loss: 0.1682, Train accuracy: 0.97
Validation loss: 2.9573, Validation accuracy: 0.46

//… epoch: 100/100
Training loss: 0.1633, Train accuracy: 0.97
Validation loss: 2.9596, Validation accuracy: 0.46

//…training the network took 896.53 sec…//
              precision    recall  f1-score   support

    airplane       0.65      0.66      0.66       500
       truck       0.50      0.52      0.51       500
        bird       0.36      0.37      0.36       500
         car       0.59      0.61      0.60       500
         cat       0.28      0.28      0.28       500
        deer       0.39      0.35      0.37       500
         dog       0.28      0.28      0.28       500
       horse       0.48      0.49      0.49       500
      monkey       0.35      0.36      0.36       500
        ship       0.63      0.59      0.61       500

    accuracy                           0.45      5000
   macro avg       0.45      0.45      0.45      5000
weighted avg       0.45      0.45      0.45      5000

//…Confusion Matrix…//
[[331  33  20  23   5  18  10   7   3  50]
 [ 26 261  16  97  16   7   5  10   7  55]
 [ 28   5 185  21  59  57  45  23  68   9]
 [ 26 105   8 303   9   8   6   5   7  23]
 [  4  20  48   8 139  59  85  36  83  18]
 [ 15   8  67  11  64 173  55  52  46   9]
```

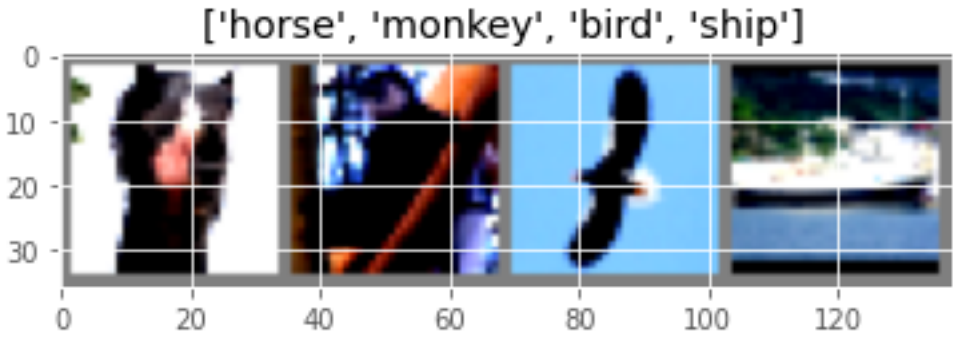
```
[  5   6  57   7  86  44 142  73  75   5]
[  4  12  38   4  34  35  85 246  41   1]
[  2   4  67  11  66  39  76  51 181   3]
[ 65  67   9  28  14   7   6   6   3 295]]
```



[87]:
```
def imshow(img,title_grid=None):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    label_name = {'1': 'airplane', '2':'bird', '3':'car', '4':'cat', '5':
 ↪'deer', '6':'dog', '7':'horse', '8':'monkey', '9':'ship', '10':'truck'}
    tc=[]
    if title_grid is not None:
        for x in title_grid:
            a=label_name[x]
            tc.append(a)
        plt.title(tc)
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

classes=train_dataset.classes
testDataLoader_disp = DataLoader(test_dataset, batch_size=4,shuffle=True)
images_test,lab_test=next(iter(testDataLoader_disp))
title_lab=((classes[lab_test[j]]) for j in range(4))
```

```
title_lab1=(list(title_lab))

imshow(torchvision.utils.make_grid(images_test),title_lab1)
```

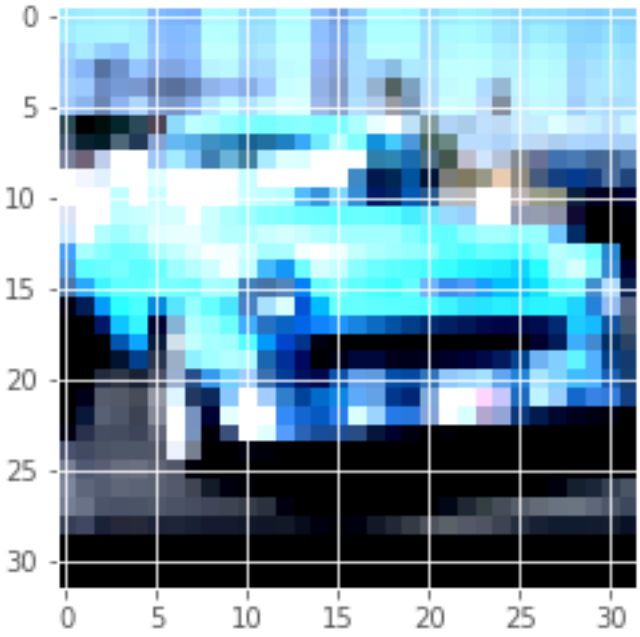
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



[115]:
```
classes=train_dataset.classes
testDataLoader_disp = DataLoader(test_dataset, batch_size=1,shuffle=True)
model=LeNet5()
model.eval()
with torch.no_grad():
    image, label= next(iter(testDataLoader_disp))
    gtLabel = classes[label.numpy()[0]]
    pred = model(image)
    idx = pred.argmax(axis=1).cpu().numpy()[0]
    predLabel = classes[idx]
    if gtLabel != predLabel:
        print("ground truth label: {}, predicted label: {}".format(gtLabel,␣
 ↪predLabel))
        imshow(torchvision.utils.make_grid(image),title_grid=None)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
ground truth label: 3, predicted label: 9
```


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