PURDUE UNIVERSITY

Elmore Family School of Electrical and Computer Engineering

Deep Learning

Homework 4

Adithya Sineesh

Email : asineesh@purdue.edu

Submitted: February 20, 2023

1 Experiment

In this homework, we do the following:

1. Make our own dataset containing 1500 training and 500 validation images each for airplane, bus, cat, dog and pizza classes from the MS-COCO dataset.

2. To train three different models on the above dataset and report their training loss and validation accuracy.

1.1 Dataset Loading

We first download the JSON annotation file. Then using the COCO API, we load 2000 images each for the above mentioned classes and resize them into (64×64) and save 1500 of them into the training dataset and 500 of them into the test dataset.

1.2 Model Training

For all the three models, the optimizer used is Adam, the learning rate is 0.001 and the loss is Cross Entropy. All the three models were trained for 10 epochs.

1.2.1 Task 1

For the first task, we train the Net1 model. It contains two convolutional layers with no padding (valid padding), two pooling layers, and two fully connected layers. The activation funtion used throughout the model is ReLU. The value of $XXXX$ is computed by multiplying the number of channels (32) with the size of the image after the second convolutional layer (14x14), which came out to be 6272. The value of XX is 5 as that is the number of classes.

1.2.2 Task 2

For the second task, we train the Net2 model. It contains two convolutional layers with padding of 1 (same padding) along with two pooling layers and two fully connected layers. The activation funtion used throughout the model is ReLU. The value of $XXXX$ is computed by multiplying the number of channels (32) with the size of the image after the second convolutional layer (16x16), which came out to be 8192. The value of XX is 5 as that is the number of classes.

1.2.3 Task 3

For the third task, we train the Net3 model. It contains twelve convolutional layers with padding of 1 (same padding) along with two pooling layers and two fully connected layers. The activation funtion used throughout the model is ReLU. The value of $XXXX$ is computed by multiplying the number of channels (32) with the size of the image after the last convolutional layer (16x16), which came out to be 8192. The value of XX is 5 as that is the number of classes.

2 Results

2.1 Dataset

Figure 1: The first column contains images from the airplane class, the second column contains images from the bus class,the third column contains images from the cat class, the fourth column contains images from the dog class and the fifth column contains images from the pizza class

2.2 Model Training

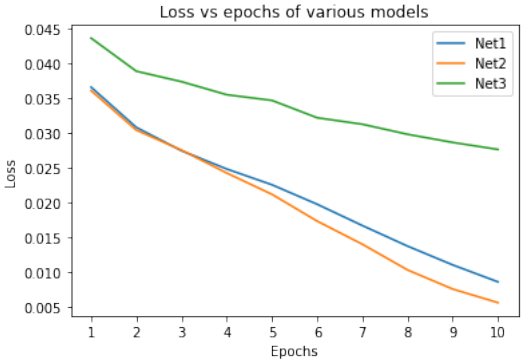


Figure 2: The training loss vs epochs plot

Below, I have plotted the confusion matrices of all the three models for validation. The x-axis contains the ground truth labels while the y-axis contains the predicted labels.

2.2.1 For Net1

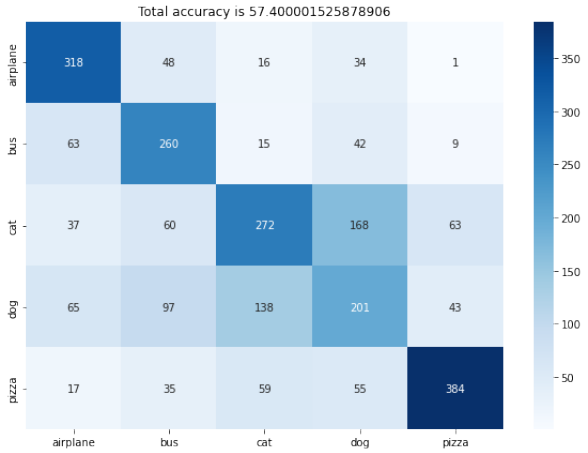


Figure 3: The validation accuracy is 57.40%

2.2.2 For Net2

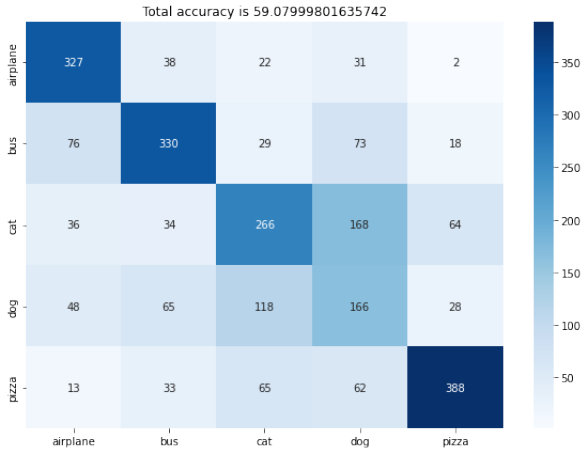


Figure 4: The validation accuracy is 59.08%

2.2.3 For Net3

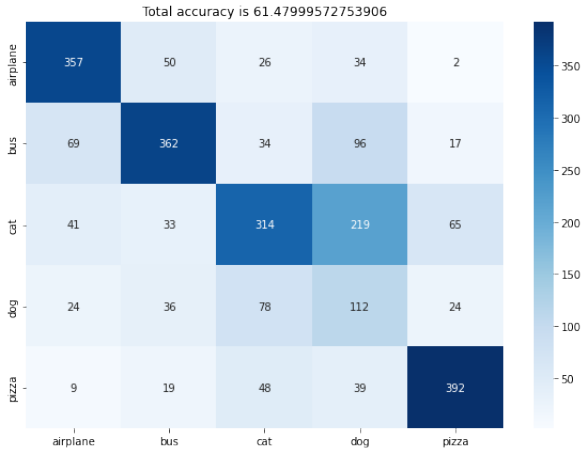


Figure 5: The validation accuracy is 61.48%

3 Conclusion

1. **Does adding padding to the convolutional layers make a difference in classification performance?**

   Yes, it improves the classification performance of the model. The accuracy of Net1 (without padding) on the validation dataset is 57.40% while the accuracy of Net2 (with padding) on the validation dataset is 59.08%. Padding prevents the shrinkage of images and also allows for the edges of the images to be preserved.

2. **As you may have known, naively chaining a large number of layers can result in difficulties in training. This phenomenon is often referred to as vanishing gradient. Do you observe something like that in Net3 ?**

   Yes, the training loss of Net3 is the slowest to reduce compared to the other two models.

3. **Compare the classification results by all three networks, which CNN do you think is the best performer?**

   Net3 gave the best performance on the validation dataset (61.48% accuracy).

4. **By observing your confusion matrices, which class or classes do you think are more difficult to correctly differentiate and why?**

   All the three models seem to struggle to classify cats and dogs. They seem to misclassify them as each other. This is most likely due to the fact that cats and dogs appear more similar to each other than any of the other classes.

5. **What is one thing that you propose to make the classification performance better?**

   The classification performance can be improved by using skip connections and batch normalization in Net3. The images can also be of a bigger size.

In [1]:

```
#Referred to the COCO API Github repo
#importing the libraries
from pycocotools.coco import COCO
import os
import time
import random
import requests
import matplotlib.pyplot as plt
import numpy as np
import skimage.io as io
from PIL import Image
```

In [2]:

```
coco=COCO("annotations_trainval2014/annotations/instances_train2014.json")
```

```
loading annotations into memory...
Done (t=10.44s)
creating index...
index created!
```

In [3]:

```
#Function to download the dataset
def download_dataset(categories):
    for category in categories:
        print(category)
        catIds = coco.getCatIds(catNms=[category]) #Getting the id of the particular id
        imgIds = coco.getImgIds(catIds=catIds ) #Getting all the ids of the images of th
        img_idxs = random.sample(imgIds, 2200) #getting the image indices of 2000 images
        total_images = 0
        for i in range(len(img_idxs)):
            if total_images == 2000:
                break
            img_url = coco.loadImgs(img_idxs[i])[0] #url of the image
            img = io.imread(img_url['coco_url'])   #The actual image
            im = Image.fromarray(img) #Converting it to PIL object
            if im is None: #To make sure that no NoneType objects are in the dataset
                continue
            im1 = im.resize((64,64))  #Resizing it
            if total_images==0: #Creating the directories
                train = os.path.join("Dataset","train",category)
                test = os.path.join("Dataset", "test",category)
                os.makedirs(train)
                os.makedirs(test)
            if total_images<1500: #Saving first 1500 images to the training dataset
                im1.save(os.path.join(train,str(i)+".jpeg"))
            elif total_images>=1500: #Saving next 500 images to the testing dataset
                im1.save(os.path.join(test,str(i)+".jpeg"))
            total_images = total_images +1
        print(i)
```

In [4]:

```
categories = ['airplane', 'bus', 'cat', 'dog', 'pizza']
download_dataset(categories)
```

```
airplane
2000
bus
2000
cat
2000
dog
2000
pizza
2000
```

In [ ]:

```

```

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
```

In [2]:

```
fig = plt.figure(figsize = (15,6))

ax11 = fig.add_subplot(3,5,1)
ax12 = fig.add_subplot(3,5,6)
ax13 = fig.add_subplot(3,5,11)

ax11.axis('off')
ax12.axis('off')
ax13.axis('off')

ax11.imshow(Image.open('Dataset/train/airplane/1.jpeg'))
ax12.imshow(Image.open('Dataset/train/airplane/2.jpeg'))
ax13.imshow(Image.open('Dataset/train/airplane/3.jpeg'))

ax21 = fig.add_subplot(3,5,2)
ax22 = fig.add_subplot(3,5,7)
ax23 = fig.add_subplot(3,5,12)

ax21.axis('off')
ax22.axis('off')
ax23.axis('off')

ax21.imshow(Image.open('Dataset/train/bus/1.jpeg'))
ax22.imshow(Image.open('Dataset/train/bus/2.jpeg'))
ax23.imshow(Image.open('Dataset/train/bus/3.jpeg'))

ax31 = fig.add_subplot(3,5,3)
ax32 = fig.add_subplot(3,5,8)
ax33 = fig.add_subplot(3,5,13)

ax31.axis('off')
ax32.axis('off')
ax33.axis('off')

ax31.imshow(Image.open('Dataset/train/cat/1.jpeg'))
ax32.imshow(Image.open('Dataset/train/cat/2.jpeg'))
ax33.imshow(Image.open('Dataset/train/cat/3.jpeg'))

ax41 = fig.add_subplot(3,5,4)
ax42 = fig.add_subplot(3,5,9)
ax43 = fig.add_subplot(3,5,14)

ax41.axis('off')
ax42.axis('off')
ax43.axis('off')

ax41.imshow(Image.open('Dataset/train/dog/1.jpeg'))
ax42.imshow(Image.open('Dataset/train/dog/2.jpeg'))
ax43.imshow(Image.open('Dataset/train/dog/3.jpeg'))

ax51 = fig.add_subplot(3,5,5)
ax52 = fig.add_subplot(3,5,10)
ax53 = fig.add_subplot(3,5,15)

ax51.axis('off')
ax52.axis('off')
ax53.axis('off')

ax51.imshow(Image.open('Dataset/train/pizza/1.jpeg'))
```

```
ax52.imshow(Image.open('Dataset/train/pizza/2.jpeg'))
ax53.imshow(Image.open('Dataset/train/pizza/3.jpeg'))
```

Out[2]:

```
<matplotlib.image.AxesImage at 0x1402852ca00>
```

In [ ]:

In [1]:

```
#importing all the libraries
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset
import torch.nn.functional as F
import os
import torchvision.transforms as transforms
from torchvision import datasets,models
from tqdm import tqdm
from PIL import Image
import numpy as np
import time
import matplotlib.pyplot as plt
import seaborn as sn
import pandas as pd
```

In [2]:

```
#My own implementation of the Dataset class
class IndexedDataset(Dataset):

    def __init__(self, dir_path):
        self.dir_path = dir_path

        if os.path.basename(self.dir_path) == 'train': #transforms for the train dataset
          self.transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize([0.5,0.5,0.5],[0.5,0.5,0.5])
        ])
        elif os.path.basename(self.dir_path) == 'test': #transforms for the test dataset
          self.transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize([0.5,0.5,0.5],[0.5,0.5,0.5])
        ])

        image_filenames = []
        for (dirpath, dirnames, filenames) in os.walk(dir_path): #Saving all the image l
            image_filenames += [os.path.join(dirpath, file) for file in filenames]
        self.image_filenames = image_filenames
        self.labels_map = {"airplane" : 0, "bus": 1, "cat" : 2, "dog" : 3, "pizza" : 4}

    def __len__(self):
        return len(self.image_filenames)

    def __getitem__(self, idx):
        img_name = self.image_filenames[idx] #Reading the image
        image = Image.open(img_name).convert('RGB')
        image = self.transform(image)
        return image, self.labels_map[os.path.basename(os.path.dirname(img_name))]
```

In [3]:

```
#The first model as given in the Homework
class HW4Net1(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3,16,3)
        self.pool = nn.MaxPool2d(2,2)
        self.conv2 = nn.Conv2d(16,32,3)
        self.fc1 = nn.Linear(6272,64)
        self.fc2 = nn.Linear(64,5)

    def forward(self,x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(x.shape[0],-1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

In [4]:

```
#The second model as given in the Homework
class HW4Net2(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3,16,3,padding=1)
        self.pool = nn.MaxPool2d(2,2)
        self.conv2 = nn.Conv2d(16,32,3,padding=1)
        self.fc1 = nn.Linear(8192,64)
        self.fc2 = nn.Linear(64,5)

    def forward(self,x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(x.shape[0],-1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

In [5]:

```
#The third model as given in the Homework
class HW4Net3(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3,16,3,padding=1)
        self.pool = nn.MaxPool2d(2,2)
        self.conv2 = nn.Conv2d(16,32,3,padding=1)
        self.conv3 = nn.Conv2d(32,32,3,padding=1)
        self.conv4 = nn.Conv2d(32,32,3,padding=1)
        self.conv5 = nn.Conv2d(32,32,3,padding=1)
        self.conv6 = nn.Conv2d(32,32,3,padding=1)
        self.conv7 = nn.Conv2d(32,32,3,padding=1)
        self.conv8 = nn.Conv2d(32,32,3,padding=1)
        self.conv9 = nn.Conv2d(32,32,3,padding=1)
        self.conv10 = nn.Conv2d(32,32,3,padding=1)
        self.conv11 = nn.Conv2d(32,32,3,padding=1)
        self.conv12 = nn.Conv2d(32,32,3,padding=1)
        self.fc1 = nn.Linear(8192,64)
        self.fc2 = nn.Linear(64,5)

    def forward(self,x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = F.relu(self.conv5(x))
        x = F.relu(self.conv6(x))
        x = F.relu(self.conv7(x))
        x = F.relu(self.conv8(x))
        x = F.relu(self.conv9(x))
        x = F.relu(self.conv10(x))
        x = F.relu(self.conv11(x))
        x = F.relu(self.conv12(x))
        x = x.view(x.shape[0],-1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

In [6]:

```
#Function to train the models. Taken from the Homework
def training(epochs,optimizer,criterion,net,train_data_loader,device):
  train_losses = []
  for epoch in range(epochs):
    running_loss = 0.0
    for i, data in enumerate(train_data_loader):
      inputs, labels = data
      inputs = inputs.to(device)
      labels = labels.to(device)
      optimizer.zero_grad()
      outputs = net(inputs)
      loss = criterion(outputs, labels)
      loss.backward()
      optimizer.step()
      running_loss += loss.cpu().item()
      if (i + 1) % 100 == 0:
        print("[epoch: %d, batch: %5d] loss: %.3f" % (epoch + 1, i + 1, running_loss/(i+
    train_losses.append(running_loss/len(train_dataset)) #Storing the loss per image for
  return net, train_losses
```

In [7]:

```
#Function to compute the confusion matrix
def confusion_matrix(model,test_data_loader):
  matrix = torch.zeros((5,5))
  with torch.no_grad():
    for b, (X_test, y_test) in enumerate(test_data_loader):
      model.eval()
      X_test, y_test = X_test.to(device), y_test.to(device)
      # Apply the model
      y_val = model(X_test)

      # Tally the number of correct predictions
      predicted = torch.max(y_val.data, 1)[1]
      for i in range(len(predicted)):
        matrix[predicted[i].cpu(),y_test[i].cpu()] += 1

  heat = pd.DataFrame(matrix, index = [i for i in ["airplane","bus","cat","dog","pizza"]
                  columns = [i for i in ["airplane","bus","cat","dog","pizza"]])
  heat = heat.astype(int)
  accuracy = (matrix.trace()/matrix.sum())*100
  plt.figure(figsize = (10,7))
  plt.title("Total accuracy is "+str(accuracy.item()))
  sn.heatmap(heat, annot=True,cmap='Blues',fmt='g')
```

In [8]:

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

Out[8]:

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

In [18]:

```
#Loading the dataset into the dataloader
train_dataset = IndexedDataset("/content/drive/MyDrive/Dataset_HW4/train")
test_dataset = IndexedDataset("/content/drive/MyDrive/Dataset_HW4/test")
train_data_loader = DataLoader(train_dataset,batch_size=32, shuffle=True,num_workers=32)
test_data_loader = DataLoader(test_dataset,batch_size=32, shuffle=True,num_workers=32)

#Defining the loss function and the number of epochs
criterion = nn.CrossEntropyLoss()
epochs = 10

#Creating an instance of the models and their respective optimizers
model1 = HW4Net1().to(device)
optimizer1 = torch.optim.Adam(model1.parameters(), lr=0.001, betas = (0.9,0.99))

model2 = HW4Net2().to(device)
optimizer2 = torch.optim.Adam(model2.parameters(), lr=0.001, betas = (0.9,0.99))

model3 = HW4Net3().to(device)
optimizer3 = torch.optim.Adam(model3.parameters(), lr=0.001, betas = (0.9,0.99))
```

```
/usr/local/lib/python3.8/dist-packages/torch/utils/data/dataloader.py:55
4: UserWarning: This DataLoader will create 32 worker processes in total.
Our suggested max number of worker in current system is 2, which is small
er than what this DataLoader is going to create. Please be aware that exc
essive worker creation might get DataLoader running slow or even freeze,
lower the worker number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
```

In [10]:

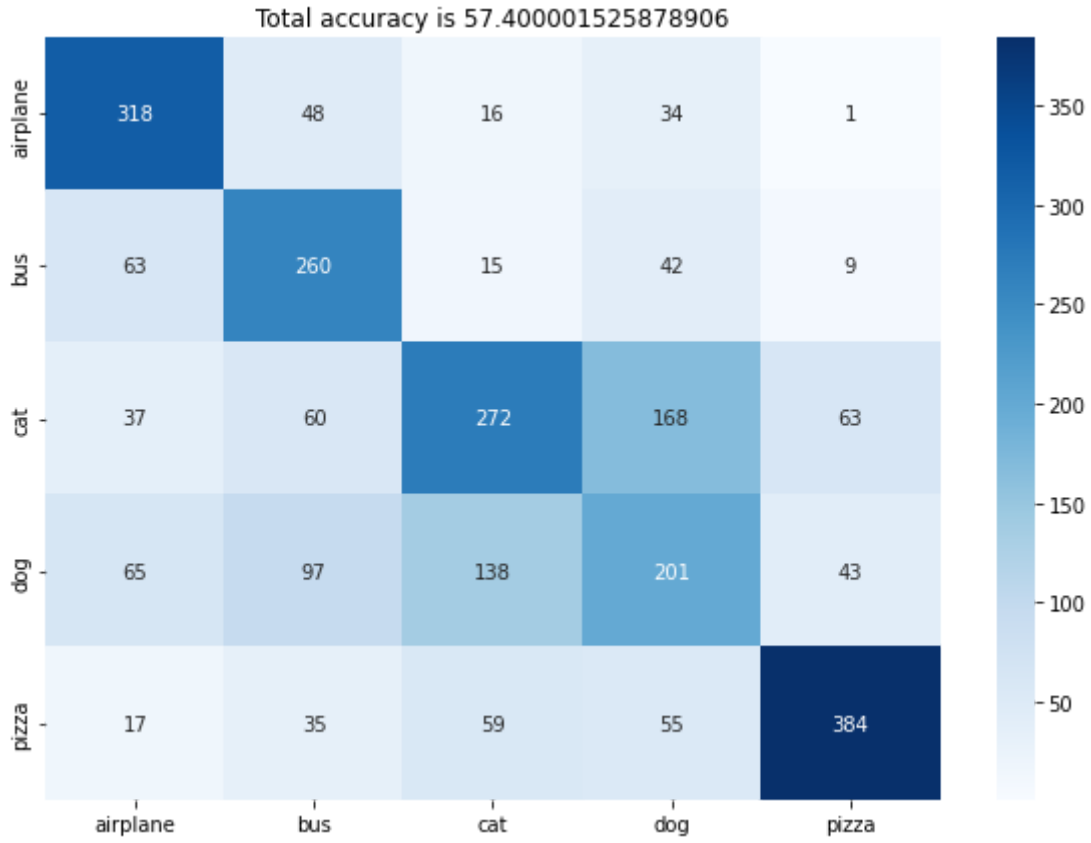
```
#Checking whether the train and test datasets are of the correct size
print(len(train_dataset))
print(len(test_dataset))
```

```
7500
2500
```

In [11]:

```
#Training the first model and plotting its confusion matrix
trained_model1,train_losses1 = training(epochs,optimizer1,criterion,model1,train_data_lo
confusion_matrix(trained_model1,test_data_loader)
```

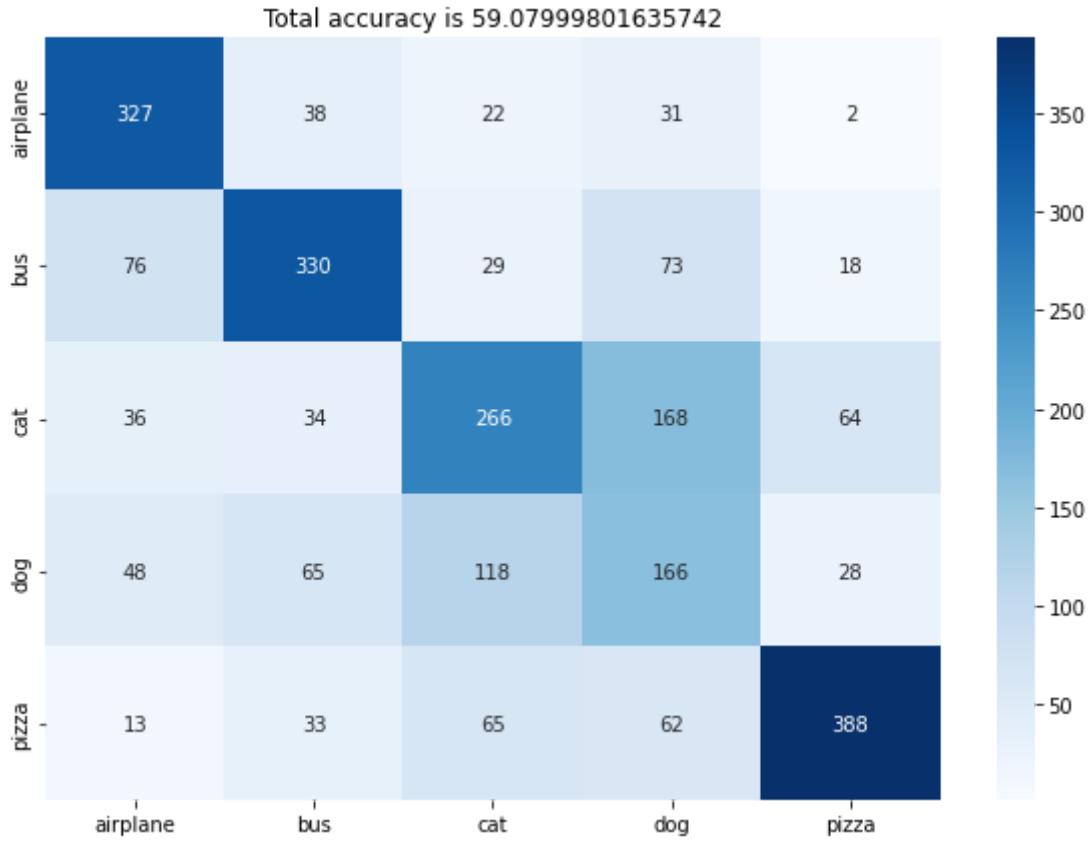
```
[epoch: 1, batch:   100] loss: 1.252
[epoch: 1, batch:   200] loss: 1.189
[epoch: 2, batch:   100] loss: 0.993
[epoch: 2, batch:   200] loss: 0.974
[epoch: 3, batch:   100] loss: 0.884
[epoch: 3, batch:   200] loss: 0.880
[epoch: 4, batch:   100] loss: 0.798
[epoch: 4, batch:   200] loss: 0.793
[epoch: 5, batch:   100] loss: 0.735
[epoch: 5, batch:   200] loss: 0.723
[epoch: 6, batch:   100] loss: 0.607
[epoch: 6, batch:   200] loss: 0.630
[epoch: 7, batch:   100] loss: 0.503
[epoch: 7, batch:   200] loss: 0.529
[epoch: 8, batch:   100] loss: 0.425
[epoch: 8, batch:   200] loss: 0.438
[epoch: 9, batch:   100] loss: 0.341
[epoch: 9, batch:   200] loss: 0.354
[epoch: 10, batch:   100] loss: 0.262
[epoch: 10, batch:   200] loss: 0.272
```



In [12]:

```
#Training the second model and plotting its confusion matrix
trained_model2,train_losses2 = training(epochs,optimizer2,criterion,model2,train_data_lo
confusion_matrix(trained_model2,test_data_loader)
```

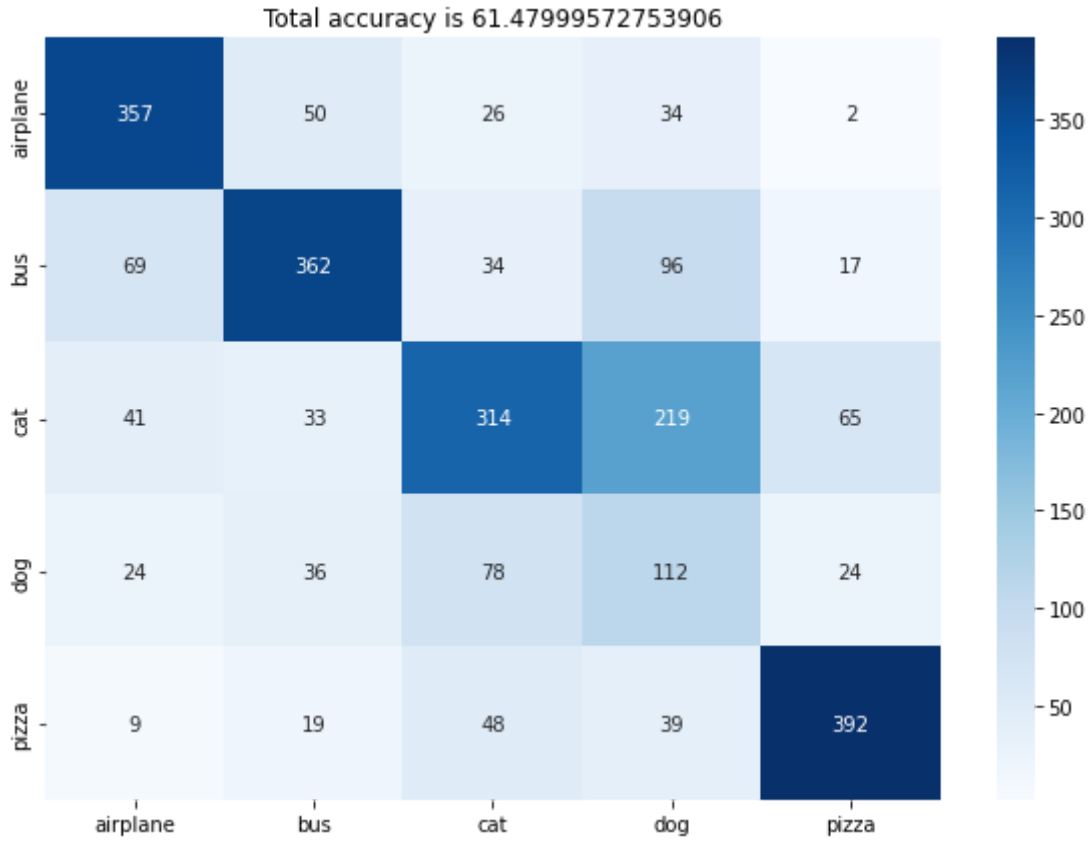
```
[epoch: 1, batch:   100] loss: 1.243
[epoch: 1, batch:   200] loss: 1.167
[epoch: 2, batch:   100] loss: 0.975
[epoch: 2, batch:   200] loss: 0.973
[epoch: 3, batch:   100] loss: 0.907
[epoch: 3, batch:   200] loss: 0.881
[epoch: 4, batch:   100] loss: 0.787
[epoch: 4, batch:   200] loss: 0.781
[epoch: 5, batch:   100] loss: 0.666
[epoch: 5, batch:   200] loss: 0.674
[epoch: 6, batch:   100] loss: 0.552
[epoch: 6, batch:   200] loss: 0.553
[epoch: 7, batch:   100] loss: 0.431
[epoch: 7, batch:   200] loss: 0.443
[epoch: 8, batch:   100] loss: 0.305
[epoch: 8, batch:   200] loss: 0.328
[epoch: 9, batch:   100] loss: 0.219
[epoch: 9, batch:   200] loss: 0.243
[epoch: 10, batch:   100] loss: 0.169
[epoch: 10, batch:   200] loss: 0.178
```



In [19]:

```
#Training the third model and plotting its confusion matrix
trained_model3,train_losses3 = training(epochs,optimizer3,criterion,model3,train_data_lo
confusion_matrix(trained_model3,test_data_loader)
```

```
[epoch: 1, batch:   100] loss: 1.508
[epoch: 1, batch:   200] loss: 1.416
[epoch: 2, batch:   100] loss: 1.224
[epoch: 2, batch:   200] loss: 1.242
[epoch: 3, batch:   100] loss: 1.202
[epoch: 3, batch:   200] loss: 1.206
[epoch: 4, batch:   100] loss: 1.160
[epoch: 4, batch:   200] loss: 1.135
[epoch: 5, batch:   100] loss: 1.104
[epoch: 5, batch:   200] loss: 1.106
[epoch: 6, batch:   100] loss: 1.034
[epoch: 6, batch:   200] loss: 1.027
[epoch: 7, batch:   100] loss: 1.009
[epoch: 7, batch:   200] loss: 0.991
[epoch: 8, batch:   100] loss: 0.948
[epoch: 8, batch:   200] loss: 0.959
[epoch: 9, batch:   100] loss: 0.912
[epoch: 9, batch:   200] loss: 0.911
[epoch: 10, batch:   100] loss: 0.886
[epoch: 10, batch:   200] loss: 0.885
```

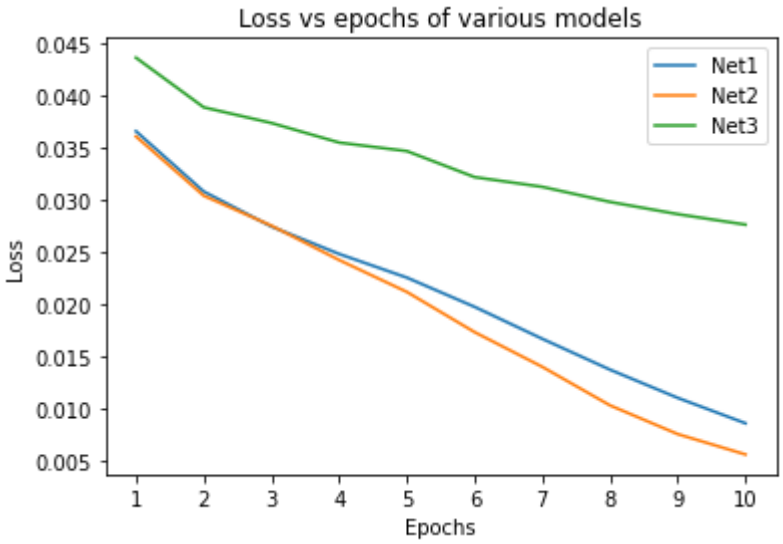


In [20]:

```
#Plotting the training loss vs epochs for all the three models
epochs = np.arange(1,11)
plt.xticks(epochs, epochs)
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs epochs of various models")
plt.plot(epochs,train_losses1,label="Net1")
plt.plot(epochs,train_losses2,label="Net2")
plt.plot(epochs,train_losses3,label="Net3")
plt.legend(loc = "upper right")
```

Out[20]:

```
<matplotlib.legend.Legend at 0x7f31998942e0>
```



In [ ]: