SPRING 2023 ECE 60146 – Homework 4

Sahithi Kodali – 34789866
kodali1@purdue.edu

<u>RESULTS:</u>

3.1 Creating own Image classification dataset

A dataset with a total of 7.5k training images and 2.5k validation images has been created, which includes 1500 training images and 500 validating images for each of the five classes: [‘airplane’, ‘bus’, ‘cat’, ‘dog’, ‘pizza’].

A sample of three images from each class in the dataset have been plotted as shown in Figure 1.

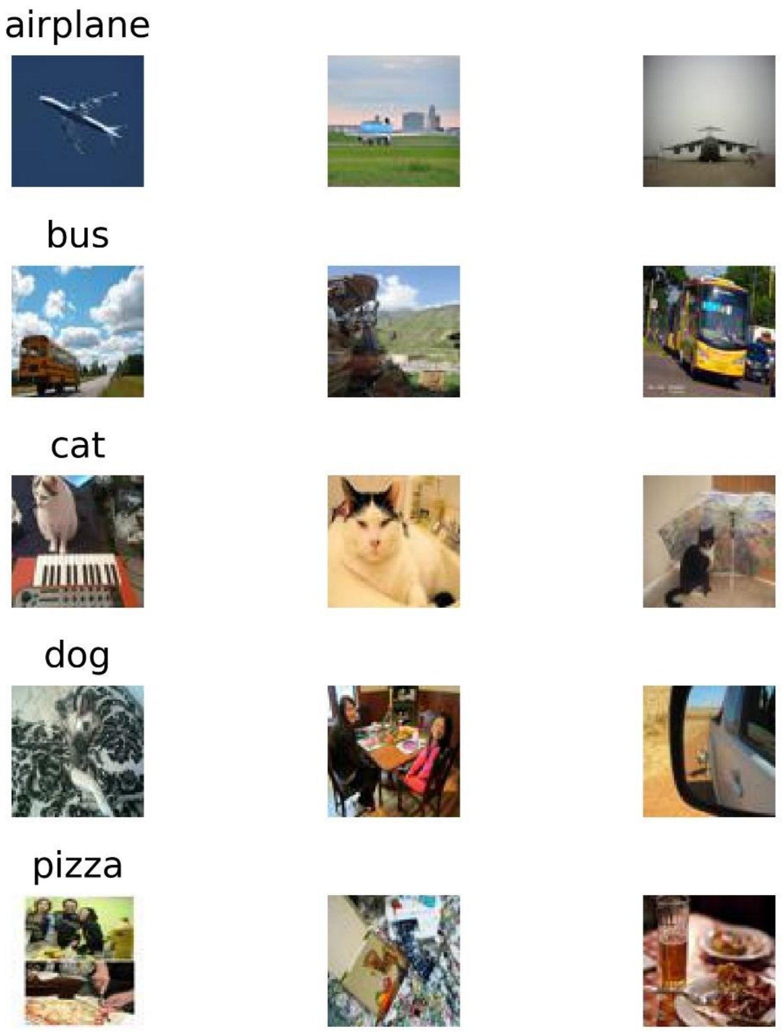

Figure 1: Plot of three images from each of the five classes

3.2 Image classification using CNNs

An own torch.utils.data.Dataset and DataLoader are implemented based on the own dataset requirements. The three networks Net1, Net2 and Net3 are created based on the instructions given in HW4 and trained using the training data. The hyperparameters used for training are 10 epochs, 32 batch size and 2 number of workers. The training loss for every 10 batches has been stored and the loss vs iterations for all the networks are plotted in Figure-2.

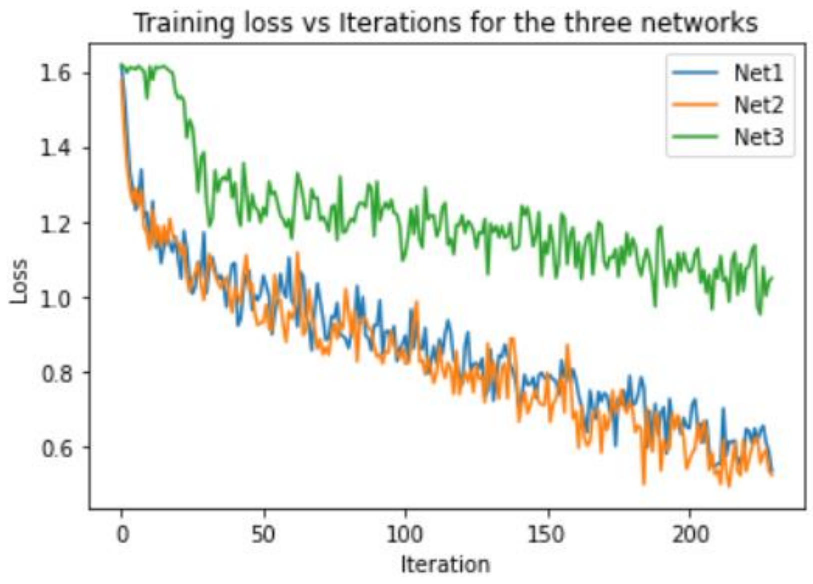

Figure 2: Loss vs iterations for the networks Net1, Net2, Net3

Further, the model is tested on the validation data to check for the performance. The accuracy along with a confusion matrix obtained using Net1, Net2, Net3 for the validation dataset are shown in Figures 3,4,5 respectively.

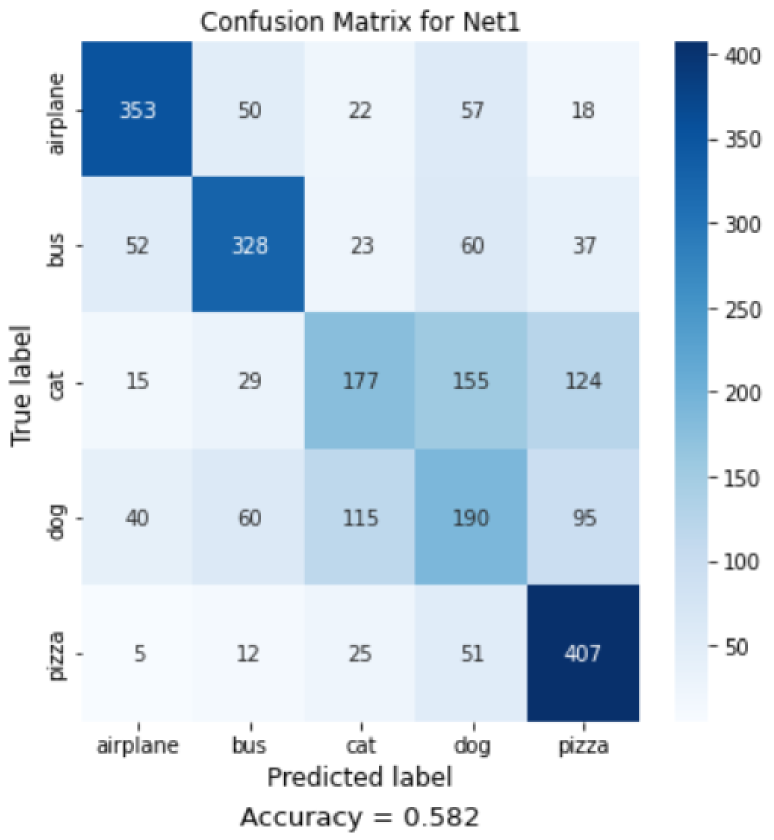

Figure 3: Validation confusion matrix using Net1 with accuracy of 0.582

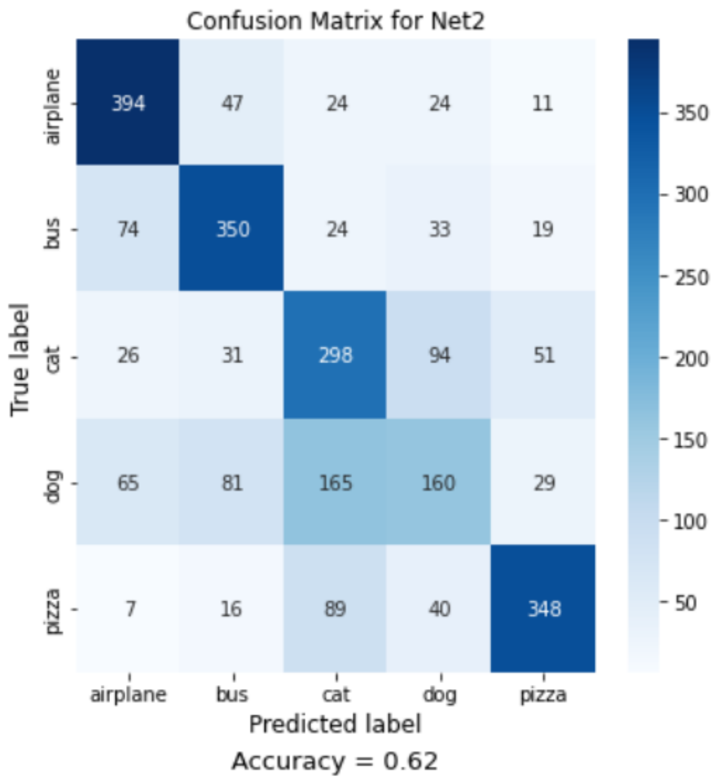

Figure 4: Validation confusion matrix using Net2 with accuracy of 0.62

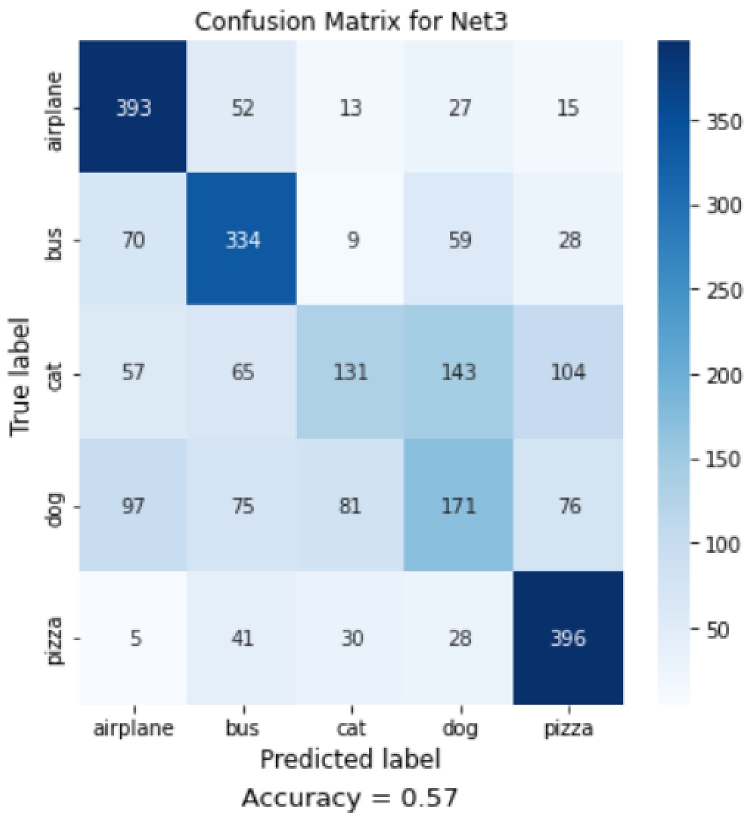

Figure 5: Validation confusion matrix using Net3 with accuracy of 0.57

<u>Answers to Questions given in HW:</u>

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

   Yes. Padding to the convolutional layers did improve the classification performance slightly in this case. However, usually we can observe the classification performance to be higher if the hyperparameters are fine-tuned for the task. This is because padding helps in preserving the edges of the image without any loss of information that is on the edges of the images, whereas, without padding the CNN layer the output feature maps obtained are smaller than the input image with each convolution performed which leads to getting rid of information on the edge pixels that might be important.

   Hence, we can say that padding the convolutional layers does improve the classification performance. In this task, we can observe the accuracy obtained using non-padded CNN is 0.58 and using padding is 0.62. The accuracy can still be improved using padding by experimenting with the hyperparameters in this task.

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, we can observe the vanishing gradient problem in Net3 where we simply chain 10 similar convolution layers to the network. We can observe that the learning has been very slow and the loss is similar as the iterations increase which is possibility due to vanishing gradients. These vanishing gradients occur when the gradients of loss function w.r.t learnable parameters become very small such that learning becomes slow/stagnant.

   The use of activation layer and batch wise training will decrease this issue to an extent, and by setting the hypermeters accordingly, however in this task we can observe the difficulty in training the deep network in Net3.

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

   The CNN in Net2 with 2 convolution layers including the padding attained good classification results of all the three networks. Though the accuracy is similar in the three cases, we can observe a clear decrease in loss using Net2 with an accuracy of 0.62 (as compared to 0.582 for Net1 and 0.57 for Net3) which is higher. Hence, we can say that Net2 is the best performer.

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

We can observe that the cat and dog classes are more difficult to correctly differentiate compared to other classes in the confusion matrix. We can also observe a quite number of dogs to be predicted as cats and vice-versa. This could be due to the similarity of shape, color and position and characteristic similarities between dog and cat such as the fur color, physical factors, and the way of sitting/standing which makes the task of classification difficult.

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

   There are many ways to make the classification performance better. One such way is to augment data. Augmenting data is the processing of applying transformation to the images in the dataset (like flipping, rotations, color transformations) to allow the model to improve its generalization over the dataset, thus improving the performance. Further, we can also experiment with ways like fine-tuning the hyperparameters, regularization etc. to improve the classification performance.

SOURCE CODE:

```
# -*- coding: utf-8 -*-
"""HW4_ece60146.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1eI8qZPW6LV-ZDZa2TnAA6rZ-KhQeG2yT
"""

# Commented out IPython magic to ensure Python compatibility.
#Change directories, unzip DLStudio and install
# %cd /content/drive/MyDrive/Purdue

# Commented out IPython magic to ensure Python compatibility.
!tar -xzf DLStudio-2.2.2.tar.gz
# %cd /content/drive/MyDrive/Purdue/DLStudio-2.2.2

!pip install .

# Commented out IPython magic to ensure Python compatibility.
#execute the paying with CIFAR10 code to check the working of the networks
# %cd Examples

pip install pymsgbox

# Commented out IPython magic to ensure Python compatibility.
# %run 'playing_with_cifar10.py'
```

```
#install COCO API and download data
!pip install pycocotools

!wget http://images.cocodataset.org/zips/train2014.zip -P
/content/drive/MyDrive/Purdue

!wget http://images.cocodataset.org/annotations/annotations_trainval2014.zip -P
/content/drive/MyDrive/Purdue

#!unzip /content/drive/MyDrive/Purdue/train2014.zip -d
/content/drive/MyDrive/Purdue

# Commented out IPython magic to ensure Python compatibility.
#import libraries required
# %matplotlib inline
from pycocotools.coco import COCO
import numpy as np
import matplotlib.pyplot as plt
import skimage.io as io
import random
import os
from shutil import copyfile

#get the annotation file and call an instance of it
annot_file =
'/content/drive/MyDrive/Purdue/annotations_trainval2014/annotations/instances_tra
in2014.json'
coco = COCO(annot_file)

#check the categories
catgs = coco.loadCats(coco.getCatIds())
print(catgs)

#obtain the image ids of the 5 classes
airplane_ids = coco.getImgIds(catIds = coco.getCatIds(['airplane']))
bus_ids = coco.getImgIds(catIds = coco.getCatIds(['bus']))
cat_ids = coco.getImgIds(catIds = coco.getCatIds(['cat']))
dog_ids = coco.getImgIds(catIds = coco.getCatIds(['dog']))
pizza_ids = coco.getImgIds(catIds = coco.getCatIds(['pizza']))

#check for the length of the data in each class
len(airplane_ids)
len(bus_ids)
len(cat_ids)
len(dog_ids)
len(pizza_ids)

#plotting an image to check for correctness of classes (from demo doc)
img = coco.loadImgs(dog_ids[np.random.randint(0,len(dog_ids))])[0]
I = io.imread(img['coco_url'])
plt.axis('off')
plt.imshow(I)
plt.show()

# load and display instance annotations (from demo doc)
plt.imshow(I); plt.axis('off')
annIds = coco.getAnnIds(imgIds=img['id'], catIds=coco.getCatIds(['dog']),
iscrowd=None)
anns = coco.loadAnns(annIds)
coco.showAnns(anns)

#function to saparate training and validation data
def get_train_val_data(ids):
  random.shuffle(ids)
  train_ids = ids[:1500]
  val_ids = ids[1500:2000]
  return train_ids, val_ids

#get the data for each class
airplane_train_ids, airplane_val_ids  = get_train_val_data(airplane_ids)
bus_train_ids, bus_val_ids = get_train_val_data(bus_ids)
cat_train_ids, cat_val_ids = get_train_val_data(cat_ids)
dog_train_ids, dog_val_ids = get_train_val_data(dog_ids)
pizza_train_ids, pizza_val_ids = get_train_val_data(pizza_ids)

#install libraries
from PIL import Image
import urllib.request

#give directories path
train_path = '/content/drive/MyDrive/Purdue/custom_data/train_data'
valid_path = '/content/drive/MyDrive/Purdue/custom_data/valid_data'

#function to save the training and validation images based on classes
def save_data(train_ids, val_ids, cls):

    #for training data
    for id in train_ids:
      img = coco.loadImgs(id)[0]
      #extract image from source path
      src_path = os.path.join('/content/drive/MyDrive/Purdue/train_14data',
img['file_name'])
      name, ext = os.path.splitext(img['file_name'])
      urllib.request.urlretrieve(img['coco_url'], src_path)

      #save image at destination path
      dest_path = os.path.join(train_path, cls)
      if not os.path.exists(dest_path):
          os.mkdir(dest_path)

      #resize to 64x64
      I = Image.open(src_path)
      im_resized = I.resize((64, 64))
      im_resized.save(os.path.join(dest_path, f"{name}.jpg"))

    print('Saved training data for ', cls)

    #for validation data
    for id in val_ids:
      img = coco.loadImgs(id)[0]

      #extract image from source path
      src_path = os.path.join('/content/drive/MyDrive/Purdue/train_14data',
img['file_name'])
      name, ext = os.path.splitext(img['file_name'])
      urllib.request.urlretrieve(img['coco_url'], src_path)

      #save image at destination path
      dest_path = os.path.join(valid_path, cls)
      if not os.path.exists(dest_path):
          os.mkdir(dest_path)

      #resize to 64x64
      I = Image.open(src_path)
      im_resized = I.resize((64, 64))
      im_resized.save(os.path.join(dest_path, f"{name}.jpg"))

    print('Saved validation data for ', cls)

#Save the data for each class
save_data(airplane_train_ids, airplane_val_ids, 'airplane')
save_data(bus_train_ids, bus_val_ids, 'bus')
save_data(cat_train_ids, cat_val_ids, 'cat')
save_data(dog_train_ids, dog_val_ids, 'dog')
save_data(pizza_train_ids, pizza_val_ids, 'pizza')

#plot three images from each class

classes = ['airplane', 'bus', 'cat', 'dog', 'pizza']

fig, axs = plt.subplots(len(classes), 3, figsize = (5,5), dpi = 450)

for i, cls in enumerate(classes):
  dir_path = f'/content/drive/MyDrive/Purdue/custom_data/train_data/{cls}/'

  three_imgs = os.listdir(dir_path)[:3]

  for j, img in enumerate(three_imgs):
    img_loc = os.path.join(dir_path, img)
    I = Image.open(img_loc)
    axs[i,j].imshow(I)
    axs[i,j].axis('off')
    if j == 0:
      axs[i,j].set_title(cls)

plt.tight_layout()
# plt.show()
fig.savefig('/content/drive/MyDrive/Purdue/three_imgs_plot2.png')

#import libraries
import torch
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from PIL import Image
import pandas as pd
import torchvision.transforms as tvt
import torch.nn as nn
import torch.nn.functional as F

#check for GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

#create dataset and dataloader
class MyDataset(Dataset):

    #intializations
    def __init__(self, root_dir, transform = None):
        super().__init__()
        self.root_dir = root_dir
        self.transform = transform
        self.classes = []
        self.image_paths = []
        self.labels = []

        for cls_folder in os.listdir(self.root_dir):
          cls_dir = os.path.join(self.root_dir, cls_folder)

          #add class name
          if os.path.isdir(cls_dir):
            self.classes.append(cls_folder)

          #add image paths and corresponding class as a label
          for img_name in os.listdir(cls_dir):
            if img_name.endswith('.jpg'):
              self.image_paths.append(os.path.join(cls_dir,img_name))
              self.labels.append(len(self.classes)-1)

    #compute length of dataset
    def __len__(self):
        return len(self.image_paths)

    #apply transformations for the image chosen by index
    def __getitem__(self, index):
        img = Image.open(self.image_paths[index]).convert('RGB')
        label = self.labels[index]

        if self.transform:
            img = self.transform(img)

        return img, label

#initialize the dataset and dataloader and apply transformations as required

transform = tvt.Compose([tvt.ToTensor()])

train_dataset = MyDataset('/content/drive/MyDrive/Purdue/custom_data/train_data',
transform = transform)
val_dataset = MyDataset('/content/drive/MyDrive/Purdue/custom_data/valid_data',
transform = transform)

#check for the data length
print(len(train_dataset))
print(len(val_dataset))
```

```
#initialize batch and num workers
batch_size = 32
num_workers = 2

#create dataloader
train_data_loader = DataLoader(train_dataset, batch_size = batch_size, shuffle =
True, num_workers=num_workers)
val_data_loader = DataLoader(val_dataset, batch_size = batch_size, shuffle =
True, num_workers=num_workers)

#Net1 CNN (code as given in HW )

class HW4Net(nn.Module):
  def __init__(self):
    super(HW4Net, self) . __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(32*14*14, 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

# Commented out IPython magic to ensure Python compatibility.
#training (code as given in HW )
net = HW4Net()
net = net.to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net.parameters() , lr=1e-3 , betas = (0.9 , 0.99))
epochs = 10
HW4Net_train_loss = []

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)

    outputs = net(inputs)

    loss = criterion(outputs, labels)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    running_loss += loss.item()
    if (i+1) % 10 == 0:
      print ("[ epoch : %d, batch : %5d] loss : %.3f" \
#               % ( epoch + 1 , i + 1 , running_loss/10 ) )
      HW4Net_train_loss.append(running_loss/10)
      running_loss = 0.0

#Net2 CNN (code as given in HW with added padding)

class HW4Net2(nn.Module):
  def __init__(self):
    super(HW4Net2, self) . __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(32*16*16, 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

# Commented out IPython magic to ensure Python compatibility.
#training with Net2 (code as given in HW )

net2 = HW4Net2()
net2 = net2.to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net2.parameters() , lr=1e-3 , betas = (0.9 , 0.99))
epochs = 10
HW4Net2_train_loss = []
```

```
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)

    outputs = net2(inputs)

    loss = criterion(outputs, labels)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    running_loss += loss.item()
    if (i+1) % 10 == 0:
      print ("[ epoch : %d, batch : %5d] loss : %.3f" \
#               % ( epoch + 1 , i + 1 , running_loss/10 ) )
      HW4Net2_train_loss.append(running_loss/10)
      running_loss = 0.0

#Net3 CNN (code as given in HW with added padding and 10 extra convolution
layers)

class HW4Net3(nn.Module):
  def __init__(self):
    super(HW4Net3, self) . __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(32*16*16, 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

# Commented out IPython magic to ensure Python compatibility.
#training with net3 (code as given in HW )

net3 = HW4Net3()
net3 = net3.to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net3.parameters() , lr=1e-3 , betas = (0.9 , 0.99))
epochs = 10
HW4Net3_train_loss = []

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)

    outputs = net3(inputs)

    loss = criterion(outputs, labels)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    running_loss += loss.item()
    if (i+1) % 10 == 0:
      print ("[ epoch : %d, batch : %5d] loss : %.3f" \
#               % ( epoch + 1 , i + 1 , running_loss/10 ) )
      HW4Net3_train_loss.append(running_loss/10)
      running_loss = 0.0

#plotting the loss vs iterations for all the three networks
plt.plot(HW4Net_train_loss, label = 'Net1')
plt.plot(HW4Net2_train_loss, label = 'Net2')
plt.plot(HW4Net3_train_loss, label = 'Net3')

plt.title('Training loss vs Iterations for the three networks')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()
plt.show()

#checking performance on validation dataset

#import libraries
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns

# make predictions on the test set using net1
net.eval()
y_true = []
y_pred = []

classes = ['airplane', 'bus', 'cat', 'dog', 'pizza']

with torch.no_grad():
    for inputs, labels in val_data_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = net(inputs)

        _, predicted = torch.max(outputs.data, 1)
        y_true += labels.cpu().numpy().tolist()
        y_pred += predicted.cpu().numpy().tolist()

# construct the confusion matrix
cm = confusion_matrix(y_true, y_pred)
acc = accuracy_score(y_true, y_pred)
```

```
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot = True, cmap = 'Blues', xticklabels = classes, yticklabels
= classes, fmt = 'g')
plt.title('Confusion Matrix for Net1')
plt.xlabel('Predicted label', fontsize = 12)
plt.ylabel('True label', fontsize = 12)
plt.text(2.5, 5.7, 'Accuracy = ' + str(acc), fontsize = 13, ha='center',
va='center')
plt.show()

# make predictions on the test set using net2
net2.eval()
y_true = []
y_pred = []

classes = ['airplane', 'bus', 'cat', 'dog', 'pizza']

with torch.no_grad():
    for inputs, labels in val_data_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = net2(inputs)

        _, predicted = torch.max(outputs.data, 1)
        y_true += labels.cpu().numpy().tolist()
        y_pred += predicted.cpu().numpy().tolist()

# construct the confusion matrix
cm = confusion_matrix(y_true, y_pred)
acc = accuracy_score(y_true, y_pred)

plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot = True, cmap = 'Blues', xticklabels = classes, yticklabels
= classes, fmt = 'g')
plt.title('Confusion Matrix for Net2')
plt.xlabel('Predicted label', fontsize = 12)
plt.ylabel('True label', fontsize = 12)
plt.text(2.5, 5.7, 'Accuracy = ' + str(acc), fontsize = 13, ha='center',
va='center')
plt.show()

# make predictions on the test set using net3
net3.eval()
```

```
y_true = []
y_pred = []

classes = ['airplane', 'bus', 'cat', 'dog', 'pizza']

with torch.no_grad():
    for inputs, labels in val_data_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = net3(inputs)

        _, predicted = torch.max(outputs.data, 1)
        y_true += labels.cpu().numpy().tolist()
        y_pred += predicted.cpu().numpy().tolist()

# construct the confusion matrix
cm = confusion_matrix(y_true, y_pred)
acc = accuracy_score(y_true, y_pred)

plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot = True, cmap = 'Blues', xticklabels = classes, yticklabels
= classes, fmt = 'g')
plt.title('Confusion Matrix for Net3')
plt.xlabel('Predicted label', fontsize = 12)
plt.ylabel('True label', fontsize = 12)
plt.text(2.5, 5.7, 'Accuracy = ' + str(acc), fontsize = 13, ha='center',
va='center')
plt.show()
```

****