Homework 4 - Deep Learning (ECE 60146)

Souradip Pal
pal43@purdue.edu
PUID 0034772329

February 20, 2023

1 Introduction

In this homework, the programming tasks give an overview of how to create our own Convolutional Neural Networks in PyTorch and use it for the classification of images. It introduces the widely used MS-COCO **(Microsoft Common Objects in COntext)** dataset for image classification and gives a holistic picture of how to use it in the training and validation of those neural networks. In this assignment, three Convolutional Neural Networks were created, each having different architecture consisting of convolutional, max_pooling, linear layers, etc. The networks were trained and validated using a subset of images present in the COCO dataset(2014 version) which were downloaded with the help of **COCO API**, a library for managing the dataset. Finally, the performance of the networks was compared by computing the confusion matrix to understand which images were correctly or incorrectly labeled for each of the classes. Some ideas related to the logic of downloading the COCO dataset, running the training and validation loops, and calculating the confusion matrix were taken from the source code of the **DLStudio** module and from the previous year's solutions for completing this assignment.

2 Methodology

Initially, to get familiar with the code for training and testing classification networks, the script *playing_with_cifar10.py* from the **ExperimentsWithCIFAR** example directory in the **DLStudio** module was run. It illustrated how the neural networks were constructed with different types of layers like convolutional layers with activation functions, pooling layers, and finally, linear layers for getting the outputs before calculating the loss. It also gives an idea of how Dataloaders were created with which the classification network was trained and validated. A similar approach was taken for the programming tasks where a custom Dataset and a Dataloader were created which were then used for the training and validation of three CNN models. For comparing the model, a method was created to calculate the confusion matrix to measure the accuracy of the models and how the models performed in classifying each of the classes. The following section gives a detailed description of each of the approaches and the results obtained from the experiments performed on the CNN models.

3 Implementation and Results

3.1 Task 3: Programming Tasks

3.1.1 Creating Image Classification Dataset

- In order to use the **COCO API** for managing the MS-COCO dataset, the python version of the API called *pycocotools* was downloaded.
- Since the 2014 version of the COCO dataset was used for this homework, the 2014 Train/Val annotation zip file was downloaded from the MS-COCO website. It contained a JSON file named *instances_train2014.json* which contained details of the images present in the *train2014.zip* file including the image ids, image URLs, image categories, etc.
- Here, only the following five categories : [‘airplane’, ‘bus’, ‘cat’, ‘dog’, ‘pizza’] were selected, and the first 1500 image URLs for training and the last 500 image URLs for validation were taken from each category. The images are then downloaded using those URLs using the *requests* python library and saved in separate directories named after each of the categories. Moreover, the images were downsampled to a smaller size of 64×64 using the *PIL* library before saving. Shown in Fig. 1 and Fig. 2 are samples of the images from the training and validation sets for each of the five classes.
- A custom *CocoDataset* class was created from *torch.utils.data.Dataset* class to load the images from each of the class directories after applying the necessary transforms. Finally, a *Dataloader* was created to wrap the dataset for processing the images in batches of 128 for training and validation.

3.1.2 Image Classification using CNNs – Training and Validation

- **CNN Task 1**: In this task, a convolutional neural network *Net1* was created as per the instructions in the assignment. *Net1* is a shallow neural network having two convolutional layers followed by max-pooling layers with two linear layers at the end. Each of the convolutional layers has a kernel size of 3. *Net1* doesn’t contain any padding and operates only in ‘valid’ mode. The input and output dimensions for the linear layers were calculated based on the output of the previous CNN layers and the dimensions of the input image. The dimension of the output for a convolutional layer is calculated using the formula $\lfloor \frac{W-F+2P}{S} \rfloor + 1$ where W denotes the height or width, F denotes the kernel size, P denotes the padding and S denotes the stride. Thus the dimension of the linear layer is calculated is follows:

$$\begin{aligned}(B, 3, 64, 64) &\xrightarrow[in\_ch=3, out\_ch=16]{conv2D(kernel\_size=3)} (B, 16, 62, 62)\\ &\xrightarrow[stride=2]{maxpool2D(kernel\_size=2)} (B, 16, 31, 31)\\ &\xrightarrow[in\_ch=16, out\_ch=32]{conv2D(kernel\_size=3)} (B, 32, 29, 29)\\ &\xrightarrow[stride=2]{maxpool2D(kernel\_size=2)} (B, 32, 14, 14)\end{aligned}$$

  Here B denotes the batch size. Hence the value of **XXXX** is calculated as $32 \times 14 \times 14 = 6272$.

  Using the training routine provided, the network was trained for 20 epochs with optimizer as Adam having parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and learning rate 0.001. In order to validate the network, a validation routine was created where the predicted labels were obtained via. model

Figure 1: Sample of training images from the COCO dataset

evaluation. These predictions were then used to calculate the confusion matrix. Fig. 3 shows the training loss and the confusion matrix for *Net1*.

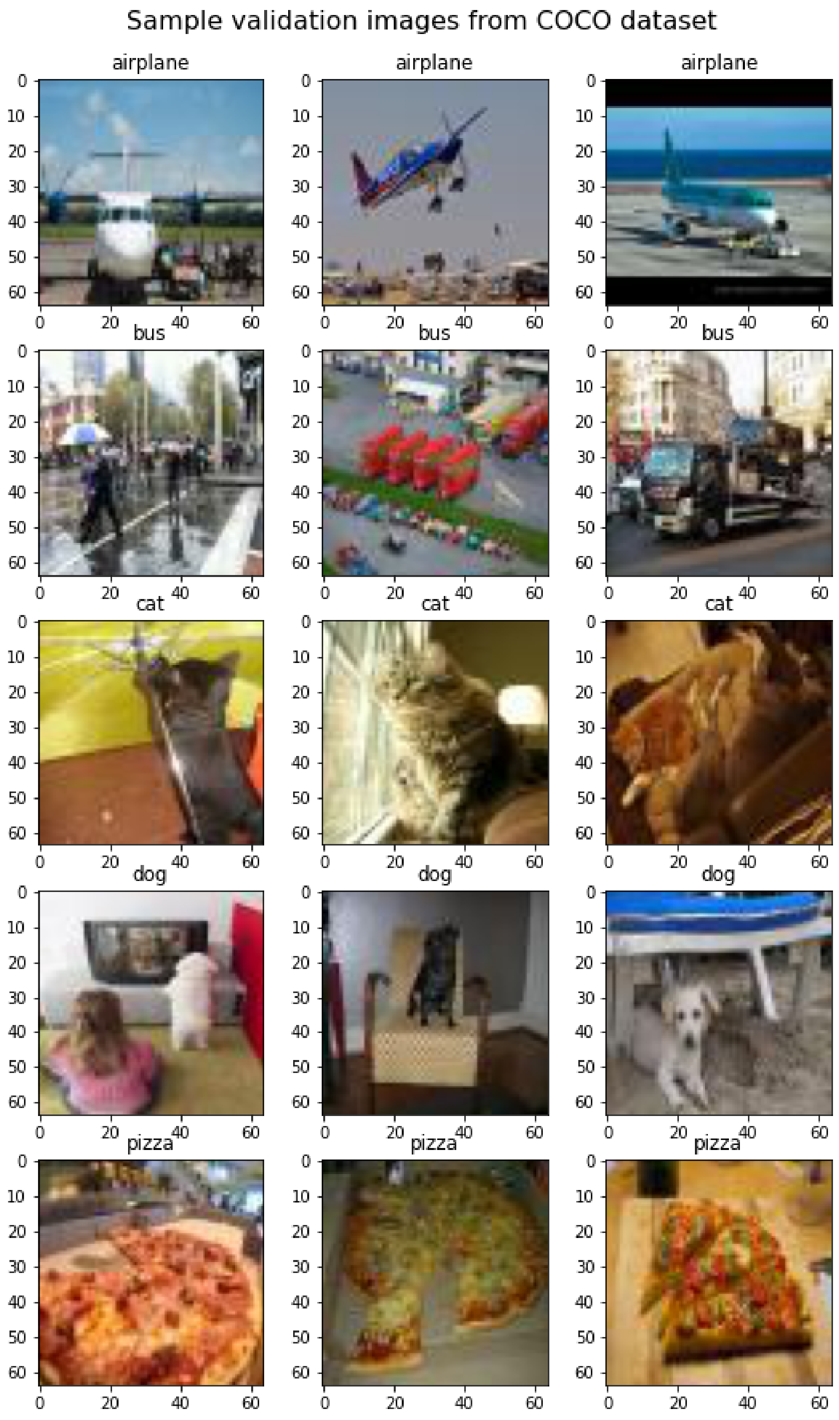
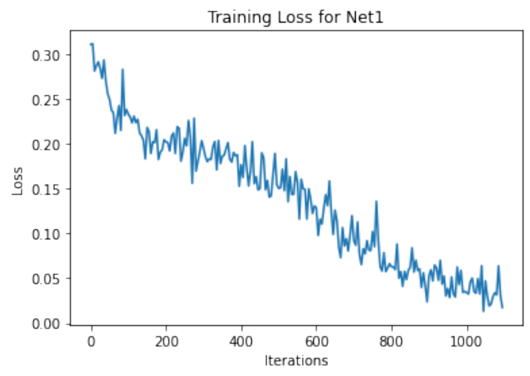
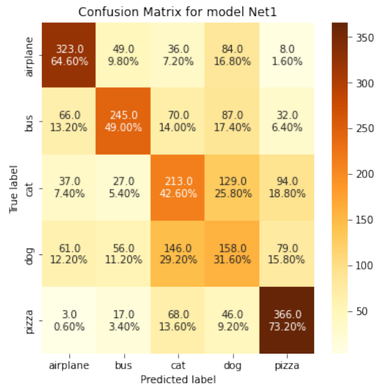


Figure 2: Sample of validation images from the COCO dataset

- **CNN Task 2**: Another convolutional neural network *Net2* was created similar to *Net1*. However, *Net2* contains convolutional layers with a padding of 1 to maintain similar input and output image



(a) Training loss vs Iterations for *Net1*



True \ Predicted	airplane	bus	cat	dog	pizza
airplane	323.0 64.60%	49.0 9.80%	36.0 7.20%	84.0 16.80%	8.0 1.60%
bus	66.0 13.20%	245.0 49.00%	70.0 14.00%	87.0 17.40%	32.0 6.40%
cat	37.0 7.40%	27.0 5.40%	213.0 42.60%	129.0 25.80%	94.0 18.80%
dog	61.0 12.20%	56.0 11.20%	146.0 29.20%	158.0 31.60%	79.0 15.80%
pizza	3.0 0.60%	17.0 3.40%	68.0 13.60%	46.0 9.20%	366.0 73.20%

(b) Confusion Matrix for *Net1* validations. Accuracy = 52.2%

Figure 3: Training loss and the confusion matrix for image classifier *Net1*

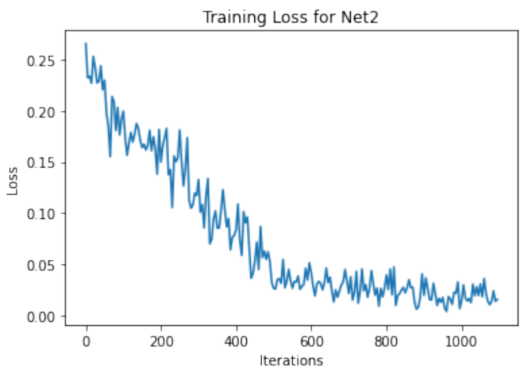
shapes. This was achieved by setting the 'padding' parameter in each *torch.nn.Conv2d* layer to 1. The input and output dimensions for the linear layers were calculated similarly as done with *Net1*. The dimension is calculated as follows:

$$\begin{aligned}(B,3,64,64) &\xrightarrow[in\_ch=3,out\_ch=16]{conv2D(kernel\_size=3,padding=1)} (B,16,64,64)\\ &\xrightarrow[stride=2]{maxpool2D(kernel\_size=2)} (B,16,32,32)\\ &\xrightarrow[in\_ch=16,out\_ch=32]{conv2D(kernel\_size=3,padding=1)} (B,32,32,32)\\ &\xrightarrow[stride=2]{maxpool2D(kernel\_size=2)} (B,32,16,16)\end{aligned}$$

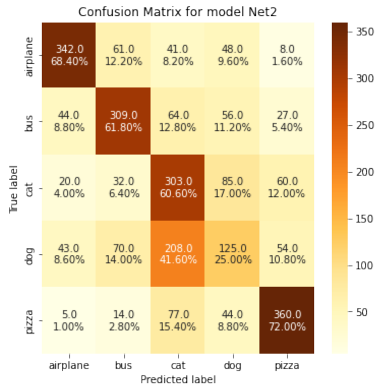
Here B denotes the batch size. Hence the value of **XXXX** for *Net2* is calculated as $32 \times 16 \times 16 = 8192$.

Using the same training routine earlier, the network was trained for 20 epochs with optimizer as Adam having parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and learning rate 0.001. In order to validate the network, the validation routine was employed to obtain the predicted labels via. model evaluation. These predictions were then used to calculate the confusion matrix. Fig. 4 shows the training loss and the confusion matrix for *Net2*.

- **CNN Task 3**: A new deep neural network *Net3* was created from *Net1* by adding 10 intermediate CNN layers with padding of 1 between the first two CNN layers and the last two linear layers, each having 32 output channels and 32 input channels. Each of the convolutional layers has a kernel size of 3. Since the kernel size of 3 and padding of 1 were used in the additional CNN layers, the shape of the output remained the same as that of the input. Thus the value of **XXXX** for *Net3* is 6272 same as *Net1*. Using the same training routine earlier, the network was trained for 20 epochs with optimizer as Adam having parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and learning rate 0.001. In order to validate the network, the validation routine was employed to obtain the predicted labels via. model evaluation. These predictions were then used to calculate the confusion matrix. Fig. 5 shows the training loss and the confusion matrix for *Net3*.

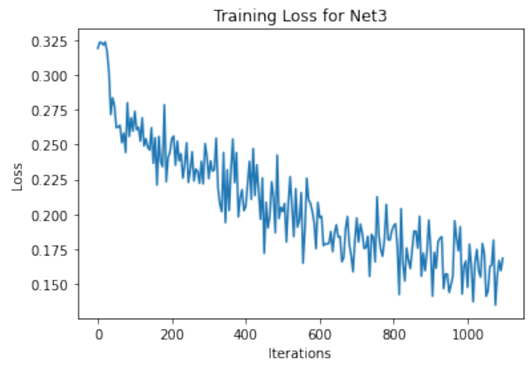


(a) Training loss vs Iterations for *Net2*

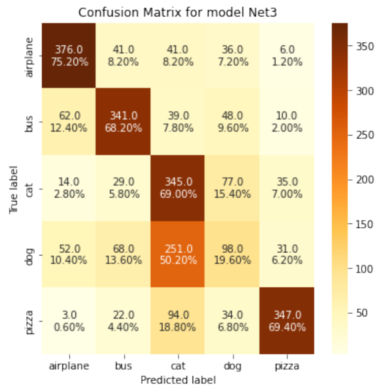


(b) Confusion Matrix for *Net2* validations. Accuracy = 57.56%

Figure 4: Training loss and the confusion matrix for image classifier *Net2*



(a) Training loss vs Iterations for *Net3*



(b) Confusion Matrix for *Net3* validations. Accuracy = 60.28%

Figure 5: Training loss and the confusion matrix for image classifier *Net3*

- **Results & Comparison**: Fig 6 shows the performance comparison between the different CNN architectures used for classifying a subset of the COCO image dataset. The validation accuracy of *Net1*, *Net2* and *Net3* are 52.2%, 57.56% and 60.28% respectively. The following section contains the answers to the questions that are provided in the assignment.

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

     Yes. Adding padding to the convolutional layers makes slight differences in the classification performance in some of the classes. Padding generally retains information from the image boundaries. Thus some categories like bus and airplane in which the objects in the images are generally larger and extend more towards the boundaries, the CNN layer architectures with padding was able to give better accuracy than the one without padding.

  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 the vanishing**

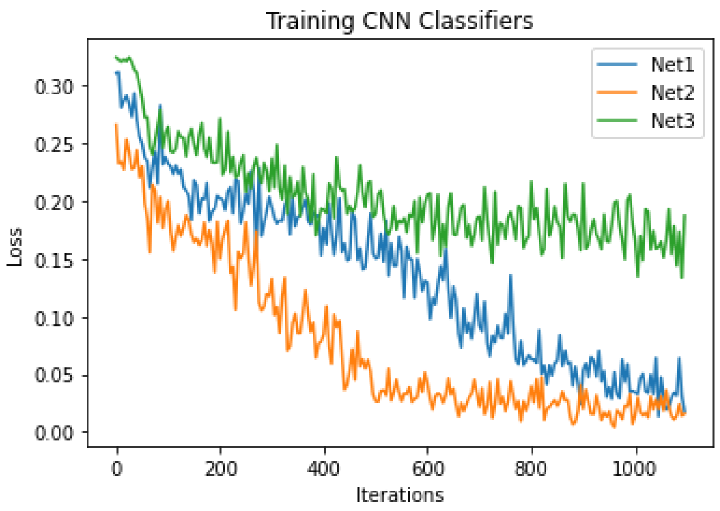


Figure 6: Training loss vs Iterations for three CNN architectures *Net1*, *Net2* and *Net3*

gradient. Do you observe something like that in Net3?

As can be observed in Fig. 6, the training loss for Net3 decreases much more slowly in comparison to the other networks. This is attributed to the naive chaining of a large number of convolutional layers which may lead to the vanishing gradient problem. Also, due to the large of learnable parameters in those layers, training is effectively slower.

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

   The image classifier *Net3* performed the best among the three networks during validation. Its validation accuracy is 60.28% whereas the validation accuracy of *Net1* and *Net2* are 52.2% and 57.56% respectively. This is due to the fact that *Net3* is a much deeper network than the other two CNN models and has more trainable parameters. Due to the chained intermediate convolution layers, it is able to extract better and richer image features and hence generalize better to distinguish between the categories.

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

   The category **dog** is the most difficult to classify. As can be seen from the confusion matrix of the three CNN models, a large percentage of dog images are has been predicted as cats. This is because there are a lot of overlapping features between cat and dog images which makes it harder for these networks to detect any significant distinguishable features between these two categories.

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

   One thing which can improve the classification score is the use of Dropout layers after the convolutional layers to reduce overfitting so that the network can generalize more.

4 Conclusion

In conclusion, Convolutional Neural Networks perform well in Image classification. However, the architecture of the CNN layers should be designed carefully to get better accuracy and faster training time. Although increasing the number of convolutional layers may result in higher classification accuracy, it may lead to slowness in training due to the problem of vanishing gradient with the increase in the number of trainable parameters. Also, padding is a great technique to chain a sequence of convolutional layers without worrying too much about the shapes of the input and output tensors. Hence, well-designed network architectures containing all these layer elements along with appropriate hyper-parameters are essential in making the training much more efficient and stable and getting higher classification accuracy.

5 Source Code

```
# -*- coding: utf-8 -*-
"""hw4_SouradipPal.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1CfvaokHX3YYlyXDl0n5FRWjhBEj4OWqY
"""

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

# Commented out IPython magic to ensure Python compatibility.
# %cd /content/drive/MyDrive/Purdue/ECE60146/HW4

!wget -O DLStudio-2.2.2.tar.gz \
    https://engineering.purdue.edu/kak/distDLS/DLStudio-2.2.2.tar.gz?download

!tar -xvf DLStudio-2.2.2.tar.gz

# Commented out IPython magic to ensure Python compatibility.
# %cd DLStudio-2.2.2

!pip install pymsgbox

!python setup.py install

# %load_ext autoreload
# %autoreload 2

!pip install pycocotools

import os
import torch
import random
import numpy as np
import requests
import matplotlib.pyplot as plt

from tqdm import tqdm
from PIL import Image
from pycocotools.coco import COCO

seed = 0
random.seed(seed)
np.random.seed(seed)

from DLStudio import *

!python Examples/playing_with_cifar10.py

!mkdir /content/drive/MyDrive/Purdue/ECE60146/HW4/coco
!wget --no-check-certificate http://images.cocodataset.org/annotations/annotations_trainval2014.zip \
    -O /content/drive/MyDrive/Purdue/ECE60146/HW4/coco/annotations_trainval2014.zip

!unzip /content/drive/MyDrive/Purdue/ECE60146/HW4/coco/annotations_trainval2014.zip -d /content/drive/MyDrive/Purdue/ECE60146/HW4/coco/

!mkdir /content/drive/MyDrive/Purdue/ECE60146/HW4/coco/train2014
!mkdir /content/drive/MyDrive/Purdue/ECE60146/HW4/coco/val2014
!mkdir /content/drive/MyDrive/Purdue/ECE60146/HW4/saved_models

# Image Downloader class to download COCO images
class ImageDownloader():
    def __init__(self, root_dir, annotation_path, classes, imgs_per_class):
        self.root_dir = root_dir
        self.annotation_path = annotation_path
        self.classes = classes
        self.images_per_class = imgs_per_class
        self.class_dir = {}
        self.coco = COCO(self.annotation_path)

    # Create directories same as category names to save images
    def create_dir(self):
        for c in self.classes:
            dir = os.path.join(self.root_dir, c)
            self.class_dir[c] = dir
            if not os.path.exists(dir):
                os.makedirs(dir)

    # Download images
    def download_images(self, download = True, val = False):
        img_paths = {}
        for c in tqdm(self.classes):
            class_id = self.coco.getCatIds(c)
            img_id = self.coco.getImgIds(catIds=class_id)
            imgs = self.coco.loadImgs(img_id)
            img_paths[c] = []

            indices = np.arange(0, self.images_per_class)
            if val == True:
                indices = np.arange(len(imgs) - self.images_per_class, len(imgs))

            for i in indices:
                img_path = os.path.join(self.root_dir, c, imgs[i]['file_name'])
                if download:
                    done = self.download_image(img_path, imgs[i]['coco_url'])
                    if done:
                        self.resize_image(img_path)
                        img_paths[c].append(img_path)
                else:
                    img_paths[c].append(img_path)

        return img_paths

    # Download image from URL using requests
    def download_image(self, path, url):
        try:
            img_data = requests.get(url).content
            with open(path, 'wb') as f:
                f.write(img_data)
            return True
        except Exception as e:
            print(f"Caught exception: {e}")
        return False

    # Resize image
    def resize_image(self, path, size = (64, 64)):
        im = Image.open(path)
        if im.mode != "RGB":
            im = im.convert(mode="RGB")
        im_resized = im.resize(size, Image.BOX)
        im_resized.save(path)

classes = ['airplane', 'bus', 'cat', 'dog', 'pizza']
train_imgs_per_class = 1500
val_imgs_per_class = 500

# Download training images
train_downloader = ImageDownloader('/content/drive/MyDrive/Purdue/ECE60146/HW4/coco/train2014',
                '/content/drive/MyDrive/Purdue/ECE60146/HW4/coco/annotations/instances_train2014.json',
                classes, train_imgs_per_class)
train_downloader.create_dir()
train_img_paths = train_downloader.download_images(download = False)

# Download validation images
val_downloader = ImageDownloader('/content/drive/MyDrive/Purdue/ECE60146/HW4/coco/val2014',
                '/content/drive/MyDrive/Purdue/ECE60146/HW4/coco/annotations/instances_train2014.json',
                classes, val_imgs_per_class)
val_downloader.create_dir()
val_img_paths = val_downloader.download_images(download = False, val = True)

# Plotting sample training images
fig, axes = plt.subplots(5, 3, figsize=(9, 15))

for i, cls in enumerate(classes):
    for j, path in enumerate(train_img_paths[cls][:3]):
        im = Image.open(path)
        axes[i][j].imshow(im)
        axes[i][j].set_title(cls)

fig.suptitle('Sample training images from COCO dataset', fontsize=16, y=0.92)
plt.show()

# Plotting sample validation images
fig, axes = plt.subplots(5, 3, figsize=(9, 15))

for i, cls in enumerate(classes):
    for j, path in enumerate(val_img_paths[cls][:3]):
        im = Image.open(path)
        axes[i][j].imshow(im)
        axes[i][j].set_title(cls)

fig.suptitle('Sample validation images from COCO dataset', fontsize=16, y=0.92)
plt.show()

import os
import torch

# Custom dataset class for COCO
class CocoDataset(torch.utils.data.Dataset):
    def __init__(self, root, transforms=None):
        super().__init__()
        self.root_dir = root
        self.classes = os.listdir(self.root_dir)
        self.transforms = transforms
        self.img_paths = []
        self.img_labels = []
        self.class_to_idx = {'airplane':0, 'bus':1, 'cat':2, 'dog':3, 'pizza': 4}
        self.idx_to_class = {i:c for c, i in self.class_to_idx.items()}

        for cls in self.classes:
            cls_dir = os.path.join(self.root_dir, cls)
            paths = os.listdir(cls_dir)
            self.img_paths+= [os.path.join(cls_dir, path) for path in paths]
            self.img_labels+=[self.class_to_idx[cls]]*len(paths)

    def __len__(self):
        # Return the total number of images
        return len(self.img_paths)

    def __getitem__(self, index):
        index = index % len(self.img_paths)
        img_path = self.img_paths[index]
        img_label = self.img_labels[index]
        img = Image.open(img_path)
        img_transformed = self.transforms(img)
        return img_transformed, img_label

import torchvision.transforms as tvt

reshape_size = 64
transforms = tvt.Compose([
    tvt.ToTensor(),
    tvt.Resize((reshape_size, reshape_size)),
    tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

# Create custom training dataset
train_dataset = CocoDataset('/content/drive/MyDrive/Purdue/ECE60146/HW4/coco/train2014/', transforms=transforms)

len(train_dataset)

# Create custom validation dataset
val_dataset = CocoDataset('/content/drive/MyDrive/Purdue/ECE60146/HW4/coco/val2014/', transforms=transforms)

len(val_dataset)

# Create custom training/validation dataloader
train_data_loader = torch.utils.data.DataLoader(train_dataset, batch_size=128, shuffle=True, num_workers=2)
val_data_loader = torch.utils.data.DataLoader(val_dataset, batch_size=128, shuffle=False, num_workers=2)

# Check dataloader
train_loader_iter = iter(train_data_loader)
img, target = next(train_loader_iter)

print('img has length: ', len(img))
print('target has length: ', len(target))
print(img[0].shape)

# Routin to train a neural network classifier
def train_net(device, net, optimizer, criterion, data_loader, model_name,
              epochs = 10, display_interval = 100):
    net = net.to(device)
    loss_running_record = []

    for epoch in range(epochs):
        running_loss = 0.0
        for i, data in enumerate(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.item()
            if (i+1) % display_interval == 0:
                avg_loss = running_loss / display_interval
                print ("[epoch : %d, batch : %5d] loss : %.3f" % (epoch + 1, i + 1, avg_loss))
                loss_running_record.append(avg_loss)
            running_loss = 0.0

    checkpoint_path = os.path.join('/content/drive/MyDrive/Purdue/ECE60146/HW4/saved_models',
                                   f'{model_name}.pth')
    torch.save(net.state_dict(), checkpoint_path)

    return loss_running_record

# Plotting training loss
def plot_loss(loss, display_interval, model_name):
    plt.figure()
    plt.plot(np.arange(len(loss))*display_interval, loss);
    plt.title(f'Training Loss for {model_name}')
    plt.xlabel('Iterations')
    plt.ylabel('Loss')
    plt.show()

# Routine to validate a neural network classifier
def validate_net(device, net, data_loader, model_path = None):
    if model_path is not None:
        net.load_state_dict(torch.load(model_path))
    net = net.to(device)
    net.eval()
    running_loss = 0.0

    device_cpu = torch.device('cpu')

    iters = 0
    imgs = []
    all_labels = []
    all_pred = []

    with torch.no_grad():
        for i, data in enumerate(data_loader):
            inputs, labels = data
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            running_loss += loss
            iters += 1
            pred_labels = torch.argmax(outputs.data, axis = 1)
            all_labels += list(labels.to(device_cpu).numpy())
            all_pred += list(pred_labels.to(device_cpu).numpy())
            imgs += list(inputs.to(device_cpu))

        print(f"Validation Loss: {running_loss/iters}")

    return imgs, all_labels, all_pred

# Function to calculate confusion matrix
def calc_confusion_matrix(num_classes, actual, predicted):
    conf_mat = np.zeros((num_classes, num_classes))
    for a, p in zip(actual, predicted):
        conf_mat[a][p]+=1
    return conf_mat

import seaborn as sns

# Plotting confusion matrix
def plot_conf_mat(conf_mat, classes, model_name):
    labels = []
    num_classes = len(classes)
    labels_per_class =  np.sum(conf_mat) / num_classes
    for row in range(num_classes):
        rows = []
        for col in range(num_classes):
            count = conf_mat[row][col]
            percent = "%.2f%%" % (count / labels_per_class * 100)
            label = str(count) + '\n' + str(percent)
            rows.append(label)
        labels.append(rows)
    labels = np.asarray(labels)

    plt.figure(figsize=(6, 6))
    sns.heatmap(conf_mat, annot=labels, fmt="", cmap="YlOrBr", cbar=True,
                xticklabels=classes, yticklabels=classes)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.title(f'Confusion Matrix for model {model_name}')
    plt.show()

# Define Net1 architecture
import torch.nn as nn
import torch.nn.functional as F

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(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

device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

# Initialize Net1
```

```
net = HW4Net()
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net.parameters() , lr=1e-3, betas=(0.9, 0.999))
epochs = 20
display_interval = 5

# Train Net1
net1_losses = train_net(device, net, optimizer=optimizer, criterion=criterion,
                        data_loader = train_data_loader, model_name = 'net1',
                        epochs=epochs, display_interval = display_interval)

# Plotting Net1 training loss
plot_loss(net1_losses, display_interval, 'Net1')

# Validate Net1
save_path = '/content/drive/MyDrive/Purdue/ECE60146/HW4/saved_models/net1.pth'
imgs, actual, predicted = validate_net(device, net, val_data_loader, model_path = save_path)

# Calculate Net1 confusion matrix and accuracy
conf_mat = calc_confusion_matrix(5, actual, predicted)
print(conf_mat)
accuracy = np.trace(conf_mat) / float(np.sum(conf_mat))
print(accuracy)

# Plotting Net1 confusion matrix
plot_conf_mat(conf_mat, classes, 'Net1')

# Define Net2
class Net2(nn.Module):
    def __init__(self):
        super(Net2, 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(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

# Initialize Net2
net2 = Net2()
net2 = net2.to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net2.parameters() , lr=1e-3, betas=(0.9, 0.999))
epochs = 20
display_interval = 5

# Train Net1
net2_losses = train_net(device, net2, optimizer=optimizer, criterion=criterion,
                        data_loader = train_data_loader, model_name = 'net2',
                        epochs=epochs, display_interval = display_interval)

# Plotting Net2 training loss
plot_loss(net2_losses, display_interval, 'Net2')

save_path = '/content/drive/MyDrive/Purdue/ECE60146/HW4/saved_models/net2.pth'
imgs, actual, predicted = validate_net(device, net2, val_data_loader, model_path = save_path)

conf_mat = calc_confusion_matrix(5, actual, predicted)
print(conf_mat)
accuracy = np.trace(conf_mat) / float(np.sum(conf_mat))
print(accuracy)

plot_conf_mat(conf_mat, classes, 'Net2')

class Net3(nn.Module):
    def __init__(self):
        super(Net3, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, 3)
        self.conv3 = nn.Conv2d(32, 32, 3, padding = 1)
        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)))
        for i in range(10):
            x = F.relu(self.conv3(x))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

net3 = Net3()
net3 = net3.to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net3.parameters() , lr=1e-3, betas=(0.9, 0.999))
epochs = 20
display_interval = 5

net3_losses = train_net(device, net3, optimizer=optimizer, criterion=criterion,
                        data_loader = train_data_loader, model_name = 'net3',
                        epochs=epochs, display_interval = display_interval)

plot_loss(net3_losses, display_interval, 'Net3')

save_path = '/content/drive/MyDrive/Purdue/ECE60146/HW4/saved_models/net3.pth'
imgs, actual, predicted = validate_net(device, net3, val_data_loader, model_path = save_path)

conf_mat = calc_confusion_matrix(5, actual, predicted)
print(conf_mat)
accuracy = np.trace(conf_mat) / float(np.sum(conf_mat))
print(accuracy)

plot_conf_mat(conf_mat, classes, 'Net3')

# Plotting training loss for Net1, Net2 and Net3
plt.figure()
x = np.arange(len(net1_losses))*display_interval
plt.plot(x, net1_losses, label='Net1')
plt.plot(x, net2_losses, label='Net2')
plt.plot(x, net3_losses, label='Net3')
plt.title('Training CNN Classifiers')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.show()
```