BME 646 / ECE 60146: Homework 4

Andrés C. Castillo J.

castil12@purdue.edu

February 20, 2023

1. Introduction

The focus of this assignment is to introduce us to the MS-COCO dataset, a widely used and well-regarded dataset for many taks like: classification and object detection. Beyond the introduction to the dataset, the assignment also introduces the concepts of convolutional neural networks (CNNs).

The assignment asks us to curate the COCO dataset and select 1500 training images as well as 500 testing images for 5 different classes. It then asks us to implement a classification task on these classes by using three differently architected CNNs and compare their performance.

2. Methodology

In order to curate the MS-COCO Dataset to fit the requirements, I first familiarized myself with the COCO API in the following link: https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocotools/coco.py (https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocotools/coco.py)

After becoming familiar with the API, I opted to downloaded the COCO dataset so it would reside in my machine locally. An advantage of this method is not having to rely on the internet to populate the training and testing datasets by downloading the images directly from their URL.

Once downloaded, I created two sub-folders: Training and Testing. Within these sub-folders I will dynamically create and populate folders with images belonging to each class. Doing it this way, will allow for easier labeling when retrieving images through a custom dataset class.

After curating the dataset, I then passed them through three different CNNs as requested, and obtained a comparison plot for all networks training loss as well as testing accuracy via a confusion matrix.

3. Implementation and Results

3.1 Creating Your Own Image Classification Dataset

As mentioned in the methodology section, my thought was to have the dataset locally and dynamically build from it everytime I ran the program. In order to do this, I created a Training and Testing sub-folder within my CustomCOCO folder.

Using the desired categories from the homework assignment: `['airplane', 'bus', 'cat', 'dog', 'pizza']` I randomly extracted 1500 training images and 500 testing images from each class, and saved them in their respective category folders. The structure can be seen on Figure 1 below.

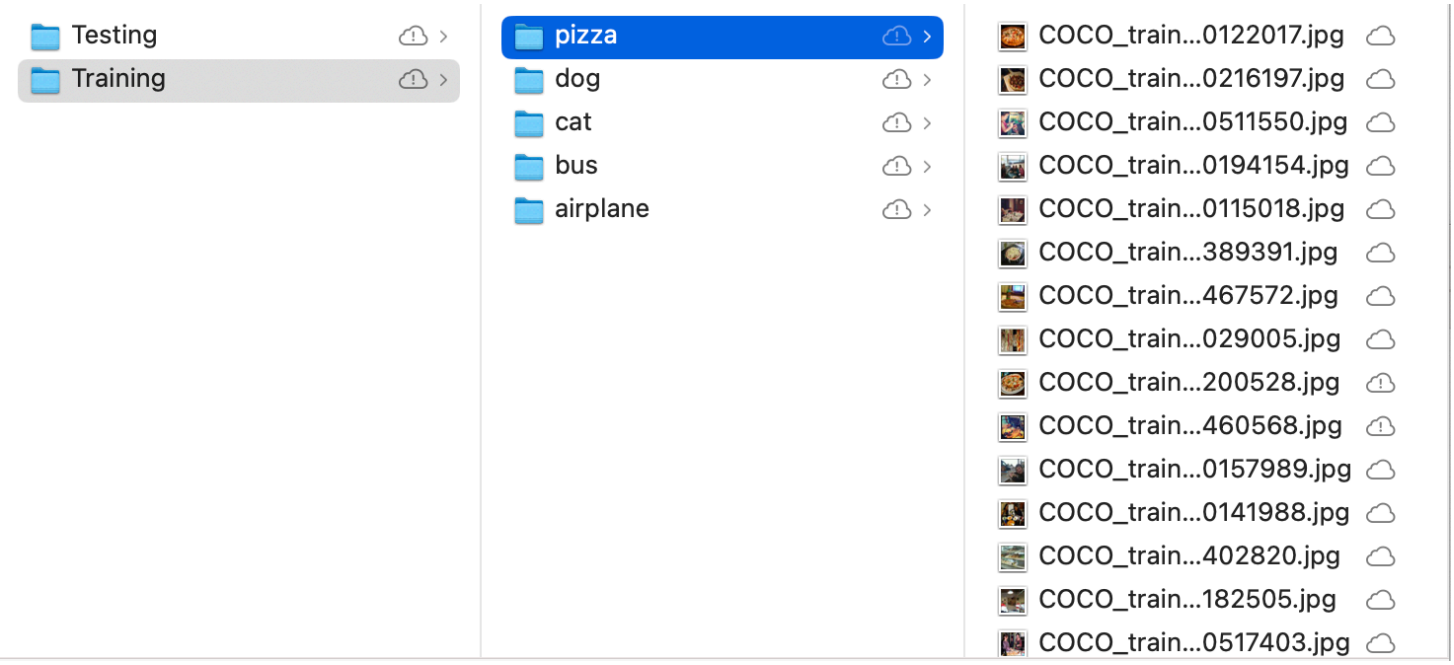

Figure 1. Folder Structure for Curated COCO Dataset.

Images in the COCO dataset can have multiple categories and the instructions were to make sure there were no duplicate images. I made sure to run a master list of all available COCO images, whenever any image was picked, the picked image was removed from the master list, ensuring that whenever new images needed to be picked they would not repeat with previously chosen images. After an image was chosen, it was then re-sized to 64x64 as requested and saved in its specific category. The logic for this can be seen on Figure 2 below.

```
def data_creation(count, coco, desired_cats, training_size, testing_size, new_size, root_images,
                  parent_dir_root_train, parent_dir_root_testing):

    if count == 10000:
        return

    # Master List
    full_img_list_with_bw = coco.getImgIds()
    full_img_list = []
    for image_id in full_img_list_with_bw:
        potential_img = coco.loadImgs(int(image_id))[0]
        img_to_open = Image.open(os.path.join(root_images, potential_img['file_name']))
        img_class = img_to_open.mode
        if img_class == 'RGB':
            full_img_list.append(image_id)

    for category in desired_cats:
        print("Populating Training and Testing folders with {} images".format(category))
        cat_id = coco.getCatIds(catNms=[category])
        cat_img_list = coco.getImgIds(catIds=[cat_id[0]]) # Get images of a specific category
        final_cat_img_list = [id for id in cat_img_list if id in full_img_list] # Only get images that have not been picked
        random_training_images = list(np.random.choice(final_cat_img_list, training_size, replace=False)) # Pick random training
        new_cat_list_no_repeat = [id for id in final_cat_img_list if id not in random_training_images] # Create new list with pickable testing images
        random_testing_images = list(np.random.choice(new_cat_list_no_repeat, testing_size, replace=False)) # Pick random testing
        full_img_list = [id for id in full_img_list if id not in final_cat_img_list] # Remove picked training and testing images
        for img_number in random_training_images:
            im = coco.loadImgs(int(img_number))[0]
            orig_img_train = Image.open(os.path.join(root_images, im['file_name']))
            resized_img_train = orig_img_train.resize(new_size)
            final_img = resized_img_train.save(os.path.join(parent_dir_root_train + "/" + category, im['file_name']))
        for img_number in random_testing_images:
            im = coco.loadImgs(int(img_number))[0]
            orig_img_test = Image.open(os.path.join(root_images, im['file_name']))
            resized_img_test = orig_img_test.resize(new_size)
            # Add if image exists condition here
            final_img = resized_img_test.save(os.path.join(parent_dir_root_testing + "/" + category, im['file_name']))

    print("Finished Populating Dataset")
```

Figure 2. Custom COCO Logic

Once the images popoulated the Training and Testing sub-folders, I focused on creating a custom Dataset class. The class is `CuratedCOCO` and can be seen on Figure 3 below. This class inherits from `torch.utils.data.Dataset` and in here, beyond assigning labels to the images, I also made sure to normalize the images and transform them into tensors by modifying the class' `__init__` and defining a transform, then in the class' `__getitem__` the Image is opened, transformed, a label assigned to it depending on its category and returned.

```
class CuratedCOCO(torch.utils.data.Dataset):

    def __init__(self, root_path, desired_categories):
        self.root_path = root_path
        self.desired_categories = desired_categories
        self.images_and_labels = []

        for categories in self.desired_categories:
            img_file_path = os.path.join(root_path, categories)
            label = self.desired_categories.index(categories) # Index of Categories, need to create a dictionary
            for path in os.listdir(img_file_path):
                info_to_append = [os.path.join(img_file_path,path), label]
                self.images_and_labels.append(info_to_append)
            # Do I need to do augmentations here?

        self.transforms = tvt.Compose([tvt.ToTensor(),
                                       tvt.Normalize((0.5, 0.5, 0.5),
                                                     (0.5, 0.5, 0.5))])

    def __len__(self):
        self.dataset_length = len(self.images_and_labels)
        return self.dataset_length

    def __getitem__(self, index):
        file_name = self.images_and_labels[index][0]
        label = self.images_and_labels[index][1]
        img = Image.open(file_name)
        img = self.transforms(img)

        # Need to return file_name
        return img, label
```

Figure 3. CuratedCOCO class

In order to test the functionality of `CuratedCOCO`, three different images for each of the five classes are plotted. They can be seen on Figure 4 below.

Figure 4. Sample Images from CuratedCOCO

3.2 Image Classification using CNNs - Training and Validation

This section requested the implementation of three different CNNs. I will be discussing each one below:

CNN Task # 1

This tasked asked us to implement the provided network into our program. However, there were some values that were missing and that we needed to calculate. This network consisted of 2 Convolutional Layers, 2 Maxpool Layers and 2 Linear Layers. In order to calculate the missing parameters in order to successfully implement this network. I used PyTorch's official Conv2d and MaxPool2d documentation to see how they calculated their output dimensions. The image can be seen on Figure 5 below.

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Figure 5. Official Formula for Conv2d

These formula, applied to both Conv2d and MaxPool2d, and by using them I was able to determine the missing `xxx` values for the network. For this network the first `xxx` value was determined to be `32 * 14 * 14`. The calculation is as follows after the first convolutional layer:

$$H_{in} = 64, padding = 0, KernelSize = 3, stride = 1$$
$$H_{out} = \lfloor \frac{64+2*(0)-(1)(3-1)-1}{1} + 1 \rfloor$$
$$H_{out} = \lfloor 61 + 1 \rfloor$$
$$H_{out} = 62$$

A similar calculation can be made for W_{out} as the image is square. After calculating the dimension after passing through the convolutional layer, the dimensions need to be calculated for the MaxPool2d layer. These are as follows:

$$H_{in} = 62, padding = 0, KernelSize = 2, stride = 2$$
$$H_{out} = \lfloor \frac{62+2*(0)-(1)(2-1)-1}{2} + 1 \rfloor$$
$$H_{out} = \lfloor 30 + 1 \rfloor$$
$$H_{out} = 31$$

Again, since the image is square, a similar calculation can be used for W_{out}.

Lastly, the dimensions after the second convolutional layer are:

$$H_{in} = 31, padding = 0, KernelSize = 3, stride = 1$$
$$H_{out} = \lfloor \frac{31+2*(0)-(1)(3-1)-1}{1} + 1 \rfloor$$
$$H_{out} = \lfloor 28 + 1 \rfloor$$
$$H_{out} = 29$$

And for the second MaxPool layer:

$$H_{in} = 29, padding = 0, KernelSize = 2, stride = 2$$
$$H_{out} = \lfloor \frac{29+2*(0)-(1)(2-1)-1}{2} + 1 \rfloor$$
$$H_{out} = \lfloor 13.5 + 1 \rfloor$$
$$H_{out} = 14$$

The value above, is the dimension of the image after the last MaxPool layer. This in turn, will be fed to the first linear layer. In order to calculate the first `xxxx` we need to multiply the image dimensions by the number of channels to get the number of parameters. In this case we get:

$$InputFeatures = 32 * 14 * 14$$
$$InputFeatures = 6272$$

As for the last `xxx`, this value is the output of the last linear layer. Since we are trying to classify images into 5 specific categories. This value will be `5`. Figure 6, below shows the values on Net1.

```
class HW4Net(nn.Module):
    def __init__(self, net):
        super(HW4Net, self).__init__()
        self.net = net
        if self.net == 'Net1':
            self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3)
            self.pool = nn.MaxPool2d(kernel_size=2,stride=2)
            self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3)
            # INPUT FEATURES CALCULATED BY 'FLATTENING THE OUTPUT OF CONV2 AND POOL
            # FINAL OUTPUT AFTER THAT LAYER HAS SHAPE BATCHSIZE X 32 X 16 X 16 (CONV2D FORMULA)
            # THEREFORE NUMBER OF INPUT PARAMETERS ARE 32*16*16
            self.fc1 = nn.Linear(in_features=32 * 14 * 14, out_features=64)
            # OUTPUT FEATURES OF LAST LAYER SHOULD BE NUMBER OF DESIRED CATEGORIES TO DISCRIMINATE FROM
            # IN THIS CASE, IT IS 5 CLASSES
            self.fc2 = nn.Linear(in_features=64, out_features=5)
```

Figure 6. Values for Net1

CNN Task # 2

This tasked asked us to implement the CNN network #1, except that each convolutional layer would have a padding of 1. This required having to recalculate the missing `xxx` values for the network as the addition of the padding would result in different dimensions as the image passed through the network. In order to calculate the missing parameters I used PyTorch's official Conv2d and MaxPool2d documentation to see how they calculated their output dimensions.

$$H_{in} = 64, padding = 1, KernelSize = 3, stride = 1$$
$$H_{out} = \lfloor \frac{64+2*(1)-(1)(3-1)-1}{1} + 1 \rfloor$$
$$H_{out} = \lfloor 63 + 1 \rfloor$$
$$H_{out} = 64$$

A similar calculation can be made for W_{out} as the image is square. After calculating the dimension after passing through the convolutional layer, the dimensions need to be calculated for the MaxPool2d layer. These are as follows:

$$H_{in} = 64, padding = 0, KernelSize = 2, stride = 2$$
$$H_{out} = \lfloor \frac{64+2*(0)-(1)(2-1)-1}{2} + 1 \rfloor$$
$$H_{out} = \lfloor 31 + 1 \rfloor$$
$$H_{out} = 32$$

Again, since the image is square, a similar calculation can be used for W_{out}.

Lastly, the dimensions after the second convolutional layer are:

$$H_{in} = 32, padding = 1, KernelSize = 3, stride = 1$$
$$H_{out} = \lfloor \frac{32+2*(1)-(1)(3-1)-1}{1} + 1 \rfloor$$
$$H_{out} = \lfloor 31 + 1 \rfloor$$
$$H_{out} = 32$$

And for the second MaxPool layer:

$$H_{in} = 32, padding = 0, KernelSize = 2, stride = 2$$
$$H_{out} = \lfloor \frac{32+2*(0)-(1)(2-1)-1}{2} + 1 \rfloor$$
$$H_{out} = \lfloor 15 + 1 \rfloor$$
$$H_{out} = 16$$

The value above, is the dimension of the image after the last MaxPool layer. This in turn, will be fed to the first linear layer. In order to calculate the first `xxxx` we need to multiply the image dimensions by the number of channels to get the number of parameters. In this case we get:

$$InputFeatures = 32 * 16 * 16$$
$$InputFeatures = 8192$$

As for the last `xxx`, this value is the output of the last linear layer. It does not change from the previous network. Since we are trying to classify images into 5 specific categories. This value will be `5`. Figure 7 below, shows the values for Net2.

```
elif self.net == 'Net2':
    self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, padding=1)
    self.pool = nn.MaxPool2d(kernel_size=2,stride=2)
    self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding=1)
    # INPUT FEATURES CALCULATED BY 'FLATTENING THE OUTPUT OF CONV2 AND POOL
    # FINAL OUTPUT AFTER THAT LAYER HAS SHAPE BATCHSIZE X 32 X 16 X 16 (CONV2D FORMULA)
    # THEREFORE NUMBER OF INPUT PARAMETERS ARE 32*16*16
    self.fc1 = nn.Linear(in_features=32 * 16 * 16, out_features=64)
    # OUTPUT FEATURES OF LAST LAYER SHOULD BE NUMBER OF DESIRED CATEGORIES TO DISCRIMINATE FROM
    # IN THIS CASE, IT IS 5 CLASSES
    self.fc2 = nn.Linear(in_features=64, out_features=5)
```

Figure 7. Values for Net2

CNN Task # 3

This tasked asked us to chain 10 additional convolutional layers, except that each new convolutional layer would have a padding of 1 and the output would only go through an activation function. This required having to recalculate the missing `xxx` values for the network after passing it through 10 additional convolutions of padding 1. as the addition of the padding would result in different dimensions as the image passed through the network. In order to calculate the missing parameters I used PyTorch's official Conv2d and MaxPool2d documentation to see how they calculated their output dimensions. From CNN Task # 2, we know that after the second convolution and MaxPool layer, the dimension is 16. Using this value with the above formulas we get the following:

$$H_{in} = 16, padding = 1, KernelSize = 3, stride = 1$$
$$H_{out} = \lfloor \frac{16+2*(1)-(1)(3-1)-1}{1} + 1 \rfloor$$
$$H_{out} = \lfloor 15 + 1 \rfloor$$
$$H_{out} = 16$$

A similar calculation can be made for W_{out} as the image is square. After calculating the dimension after passing through the convolutional layer it can be noted that the dimension stays the same. Therefore, if we were to pass this through 10 additional convolutional layers with the same kernel and padding. The final dimension would be `16x16`.

The value above, is the dimension of the image after the tenth convolutional layer. This in turn, will be fed to the first linear layer. In order to calculate the first `xxxx` we need to multiply the image dimensions by the number of channels to get the number of parameters. In this case we get:

$$InputFeatures = 32 * 16 * 16$$
$$InputFeatures = 8192$$

As for the last `xxx`, this value is the output of the last linear layer. It does not change from the previous network. Since we are trying to classify images into 5 specific categories. This value will be `5`. Figure 8 below, shows the values for Net3.

```
elif self.net == 'Net3':
    self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, padding=1)
    self.pool = nn.MaxPool2d(kernel_size=2,stride=2)
    self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding=1)
    self.conv_layers = nn.ModuleList()
    for _ in range(10):
        self.conv_layers.append(nn.Conv2d(in_channels=32,out_channels=32,kernel_size=3,padding=1))
    # INPUT FEATURES CALCULATED BY 'FLATTENING THE OUTPUT OF 10 LAYERS OF CONVOLUTION
    # FINAL OUTPUT AFTER THAT LAYER HAS SHAPE BATCHSIZE X 32 X 16 X 16 (CONV2D FORMULA)
    # THEREFORE NUMBER OF INPUT PARAMETERS ARE 32*16*16
    self.fc1 = nn.Linear(in_features=32 * 16 * 16, out_features=64)
    # OUTPUT FEATURES OF LAST LAYER SHOULD BE NUMBER OF DESIRED CATEGORIES TO DISCRIMINATE FROM
    # IN THIS CASE, IT IS 5 CLASSES
    self.fc2 = nn.Linear(in_features=64, out_features=5)
```

Figure 8. Values for Net3

Once the missing values of `xxx` were determined for each network. The training for each network was done with the following parameters:

- Learning Rate of $1\mathrm{e}{-3}$
- Both training and testing batchsizes were set to 4
- Adam as an optimizer with the following beta values: $\beta_1 = 0.9$, $\beta_2 = 0.99$
- Loss is measured throuh CrossEntropy. The reason this loss is chosen is because it performs well for classification tasks as it provides a probability value for each class. With the determined class being the one with the highest probability.
- Epochs: 15
- Note that losses are stored (and displayed) every 100 batches

After training the networks with these parameters, the following training loss plot and confusion matrices were obtained.

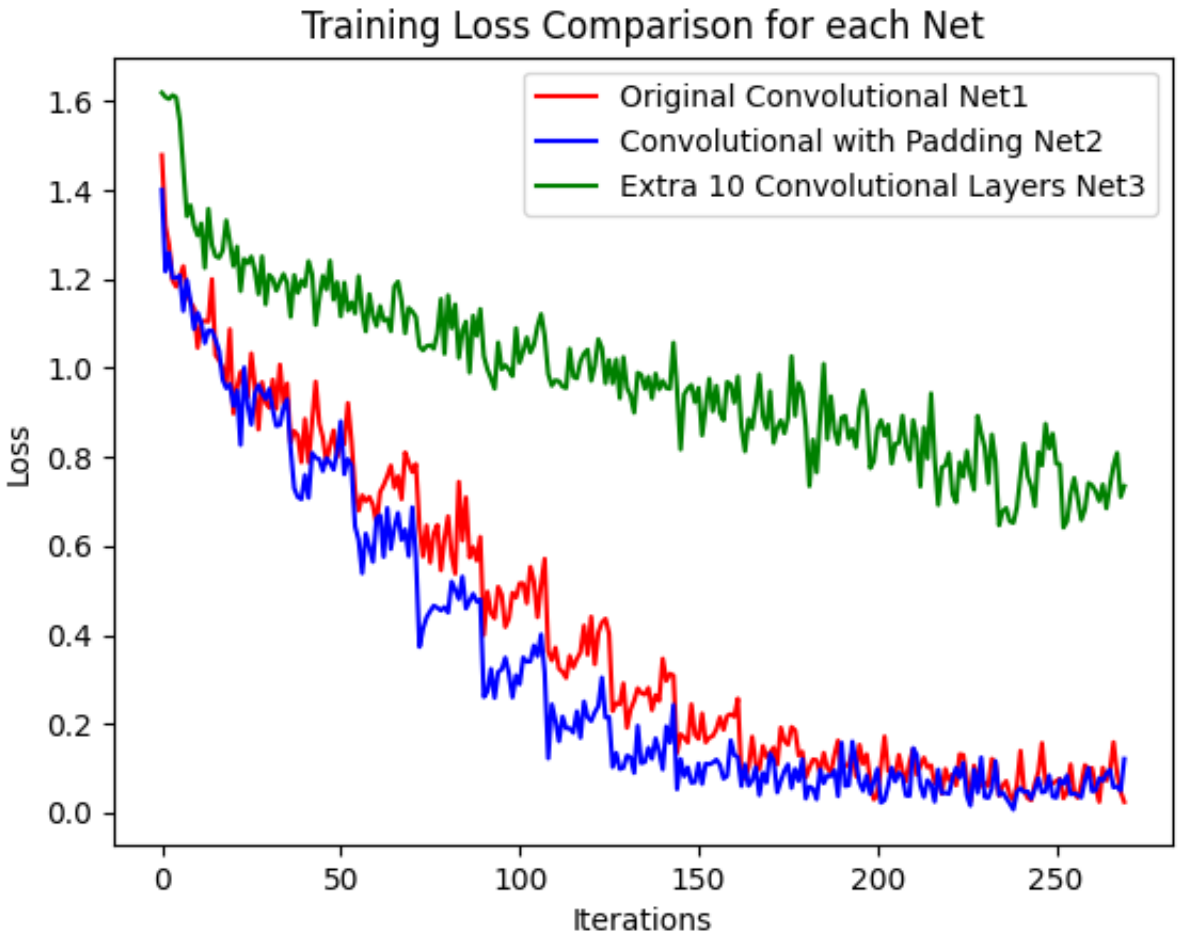

Figure 9. Training Loss Comparison.

As can be seen on Figure 9 above. After training these networks, we can see that both Network # 1 and Network # 2 have similar performance with Network # 2 performing slightly better loss wise.

However, Network # 3 does not perform as well as the other two networks. It does not seem to be learning as much as the ther networks as the loss does not decrease to the same levels of the previous netwokrs over the same iterations.

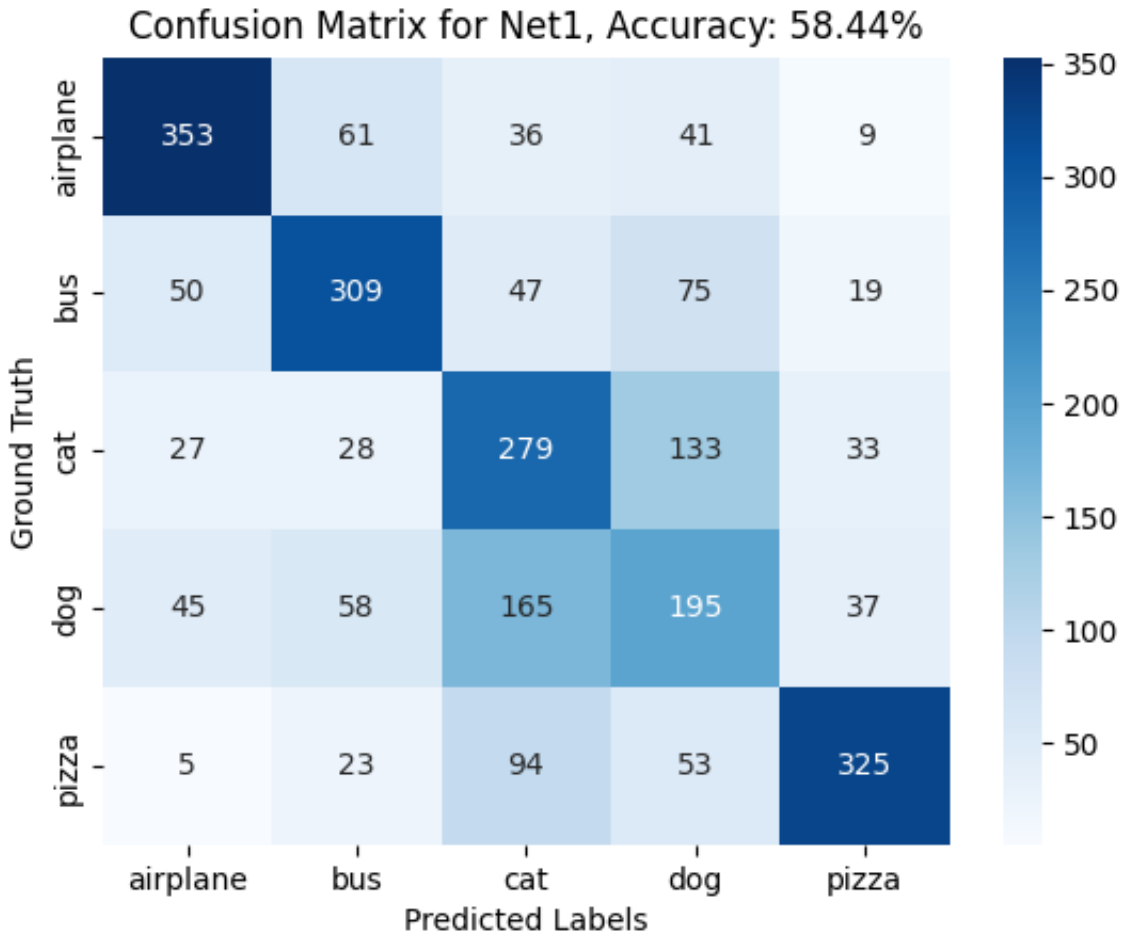

Figure 10. Confusion Matrix for Network 1

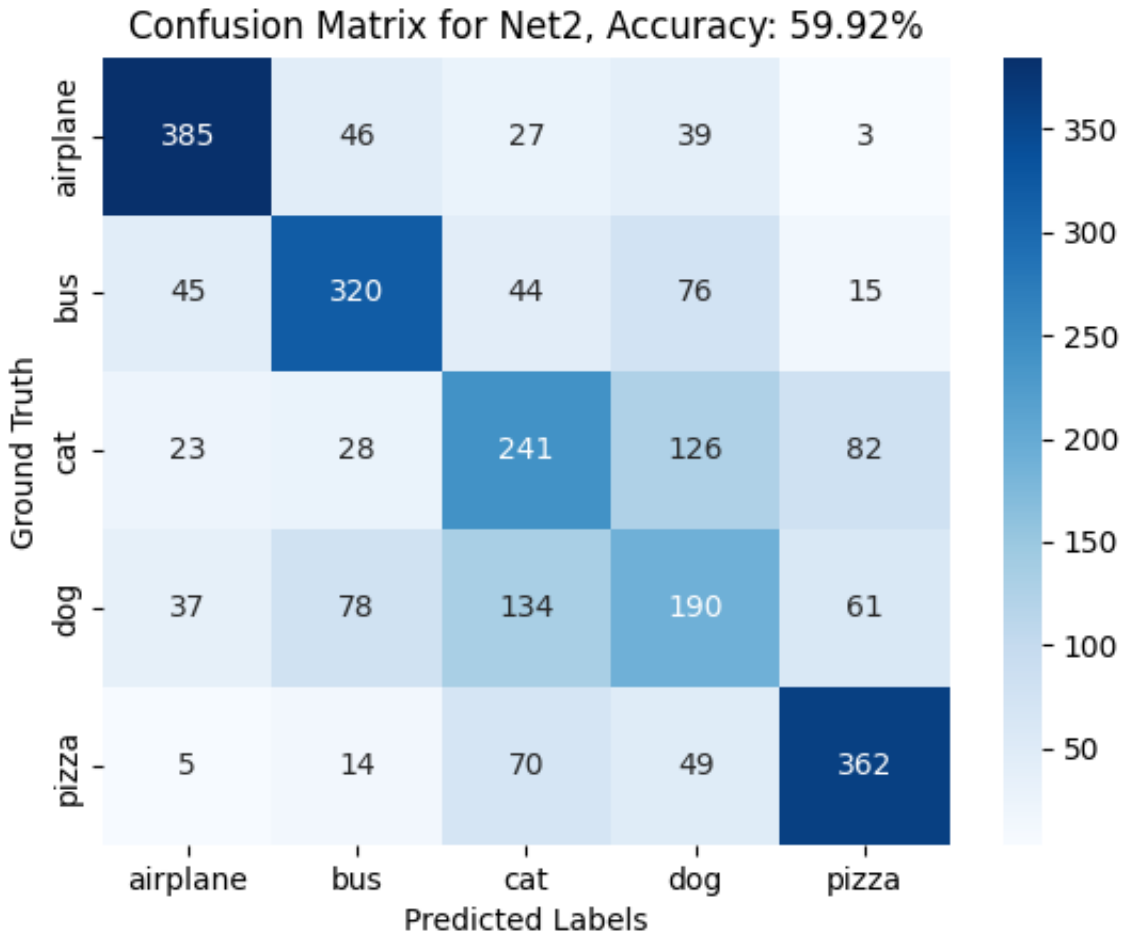

Figure 11. Confusion Matrix for Network 2

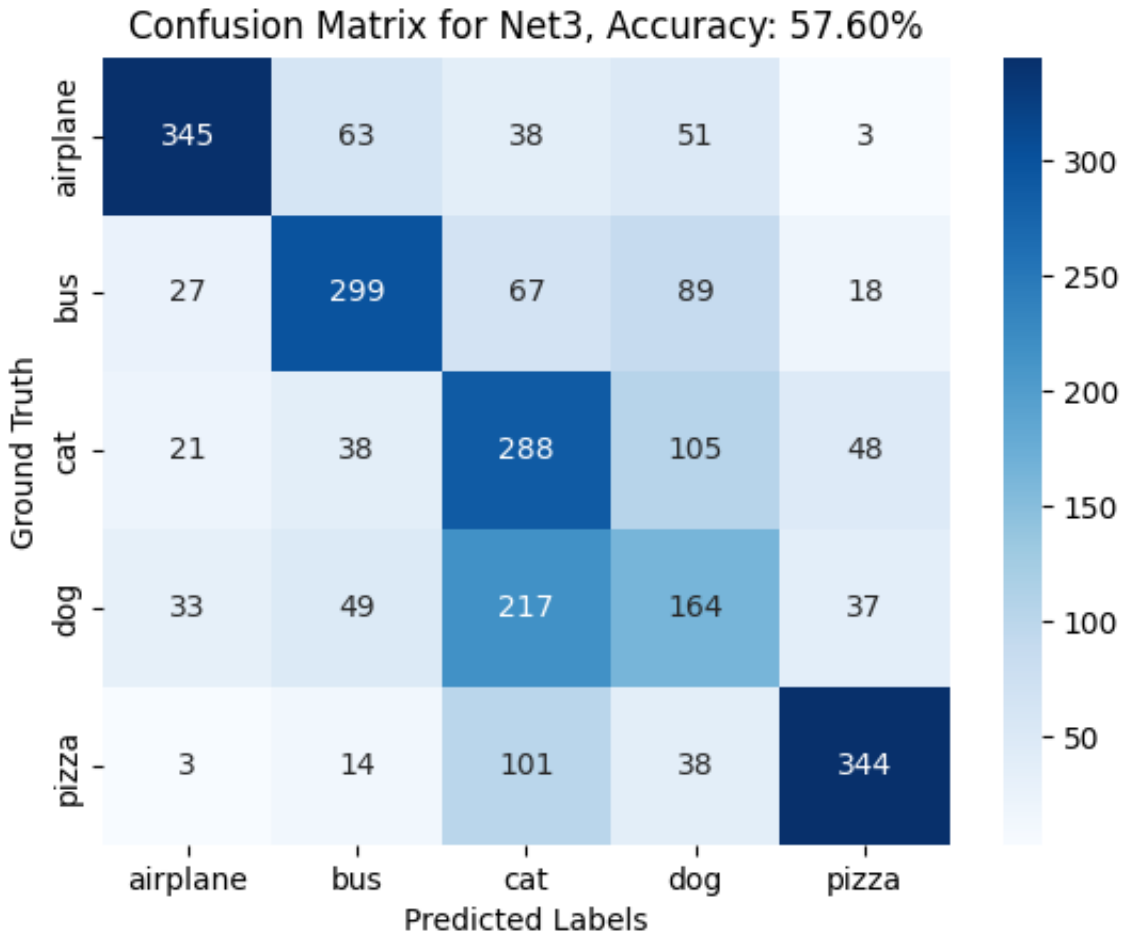

Figure 12. Confusion Matrix for Network 3

As can be seen from figures 10 - 12. The accuracy results on the networks are as follows:

- Net # 1: 58.44%
- Net # 2: 59.92%
- Net # 3: 57.60%

These results fall in line with the respective training losses, noting that Network # 3 had the highest loss and lowest testing accuracy.

With the training losses and confusion matrices readily available, the following questions can be answered:

1. Does adding padding to the convolutional layers make a difference in classification performance? **Based on the confusion matrix results, it seems to 'boost' the classification performance. By including the padding, we are keeping the image dimensions the same as they move through the convolutional network. This seems to have a positive effect in training and classification as can be seen from the training loss and confusion matrices for both Network # 1 and Network # 2.**
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, it is evident from both the training loss and confusion matrix for this network that passing through so many layers create a negative effect in the training and inference as can be seen on the plots above. The model does not seem to 'learn' as well for Net # 3 when compared to Net # 2 and Net # 1.**
3. Compare the classification results by all three networks, which CNN do you think is the best performer? **When comparing classification results, Network # 2 seems to perform the best. One thing of note, however, is that it still struggles with the animal categories, presumably because they have similar features.**
4. By observing your confusion matrices, which class or classes do you think are more difficult to correctly differentiate and why? **Definitely the animals as they score very low in the confusion matrices across the networks. Even though cats and dogs are clearly different (at least to humans), their disntiguishing features 'vanish' as these images pass through the network. Because of this, the network learns these features, but is unable to accuratley pinpoint which class it belongs to. Also, compounded by the fact that COCO images are difficult as some of the animals shown are sometimes a small portion of the image and in some cases both animals can be in the same image.**
5. What is one thing that you propose to make the classification performance better? **I think something that improve classification performance is to use more images for training. By having a bigger training pool, more features can be extracted from each class and the network will have more varied features to learn.**

4. Lessons Learned

The assignment introduced the MS-COCO dataset and the many classes that it included. Through this assignment I was able to take a raw dataset and adapt it to my own needs for training. I was also able to learn about how CNNs work and that they may not be the end-all solution for our problems.

5. Suggested Enhancements

The assignment provided a great introduction to CNNs and curating one's own dataset. The only enhancement I would suggest is including some sort of expected performance range to try and gauge whether our networks are performing as expected.

6. Code

In [ ]:
```
# Mount Google Drive

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

```
Mounted at /content/drive
```

In [ ]:
```
# Import necessary libraries for the assignment
import os
import torch
import torchvision.transforms as tvt
from PIL import Image
import numpy as np
import glob
import torch
import torch.nn as nn
import torch.nn.functional as F
import seaborn as sns
import matplotlib.pyplot as plt
import random
import time
from pycocotools.coco import COCO

# Import from self-developed files
from DataCreation import *
from CustomCOCODataset import *
from Networks import *
from ModelTrainingTesting import *
```

```
from Plotting import *



class CuratedCOCO(torch.utils.data.Dataset):

    def __init__(self, root_path, desired_categories):
        self.root_path = root_path
        self.desired_categories = desired_categories
        self.images_and_labels = []

        for categories in self.desired_categories:
            img_file_path = os.path.join(root_path, categories)
            label = self.desired_categories.index(categories) # Index
of Categories, need to create a dictionary
            for path in os.listdir(img_file_path):
                info_to_append = [os.path.join(img_file_path,path), la
bel]
                self.images_and_labels.append(info_to_append)


        self.transforms = tvt.Compose([tvt.ToTensor(),
                                       tvt.Normalize((0.5, 0.5, 0.5),
                                                     (0.5, 0.5, 0.5))]
)

    def __len__(self):
        self.dataset_length = len(self.images_and_labels)
        return self.dataset_length

    def __getitem__(self, index):
        file_name = self.images_and_labels[index][0]
        label = self.images_and_labels[index][1]
        img = Image.open(file_name)
        img = self.transforms(img)

        # Need to return file_name
        return img, label

def directory_creation(desired_cats, parent_dir_root_train, parent_dir
_root_testing):
    for categories in desired_cats:
        path_training = parent_dir_root_train + "/" + categories
        path_testing = parent_dir_root_testing + "/" + categories
        if not os.path.exists(path_training):
            print("Making Training Directory for: {}".format(categorie
s))
            os.makedirs(path_training)
            print("Finished Making Training Directory for: {}".format(
categories))
```

```
        if not os.path.exists(path_testing):
            print("Making Testing Directory for: {}".format(categories
))
            os.makedirs(path_testing)
            print("Finished Making Testing Directory for: {}".format(c
ategories))


def file_count(desired_cats, parent_dir_root_train, parent_dir_root_te
sting):
    count = 0
    for category in desired_cats:
        for path in os.listdir(parent_dir_root_train + "/" + category)
:
            if os.path.isfile(os.path.join(parent_dir_root_train + "/"
+ category, path)):
                count += 1
        for path2 in os.listdir(parent_dir_root_testing + "/" + catego
ry):
            if os.path.isfile(os.path.join(parent_dir_root_testing + "
/" + category, path2)):
                count += 1

    return count


def data_creation(count, coco, desired_cats, training_size, testing_si
ze, new_size, root_images,
                  parent_dir_root_train, parent_dir_root_testing):

    if count == 10000:
        return

    # Master List
    full_img_list_with_bw = coco.getImgIds()
    full_img_list = []
    for image_id in full_img_list_with_bw:
        potential_img = coco.loadImgs(int(image_id))[0]
        img_to_open = Image.open(os.path.join(root_images, potential_i
mg['file_name']))
        img_class = img_to_open.mode
        if img_class == 'RGB':
            full_img_list.append(image_id)

    for category in desired_cats:
        print("Populating Training and Testing folders with {} images"
.format(category))
        cat_id = coco.getCatIds(catNms=[category])
        cat_img_list = coco.getImgIds(catIds=[cat_id[0]]) # Get images
of a specific category
```

```
        final_cat_img_list = [id for id in cat_img_list if id in full_
img_list] # Only get images that have not been picked
        random_training_images = list(np.random.choice(final_cat_img_l
ist, training_size, replace=False)) # Pick random training
        new_cat_list_no_repeat = [id for id in final_cat_img_list if i
d not in random_training_images] # Create new list with pickable testi
ng images
        random_testing_images = list(np.random.choice(new_cat_list_no_
repeat, testing_size, replace=False)) # Pick random testing
        full_img_list = [id for id in full_img_list if id not in final
_cat_img_list] # Remove picked training and testing images
        for img_number in random_training_images:
            im = coco.loadImgs(int(img_number))[0]
            orig_img_train = Image.open(os.path.join(root_images, im['
file_name']))
            resized_img_train = orig_img_train.resize(new_size)
            final_img = resized_img_train.save(os.path.join(parent_dir
_root_train + "/" + category, im['file_name']))
        for img_number in random_testing_images:
            im = coco.loadImgs(int(img_number))[0]
            orig_img_test = Image.open(os.path.join(root_images, im['f
ile_name']))
            resized_img_test = orig_img_test.resize(new_size)
            # Add if image exists condition here
            final_img = resized_img_test.save(os.path.join(parent_dir_
root_testing + "/" + category, im['file_name']))

    print("Finished Populating Dataset")


def remove_files(desired_cats, parent_dir_root_train, parent_dir_root_
testing):
    for categories in desired_cats:
        path_training = parent_dir_root_train + "/" + categories
        path_testing = parent_dir_root_testing + "/" + categories
        if not os.path.exists(path_training):
            os.makedirs(path_training)
        if not os.path.exists(path_testing):
            os.makedirs(path_testing)

    for categories in desired_cats:
        path_training = parent_dir_root_train + "/" + categories
        path_testing = parent_dir_root_testing + "/" + categories

        imgs_files_training = glob.glob(os.path.join(path_training, '*
.jpg'))
        imgs_files_testing = glob.glob(os.path.join(path_testing, '*.j
pg'))

        for file_path in imgs_files_training:
```

```
            try:
                os.remove(file_path)
            except OSError as e:
                print("Error: %s : %s" % (file_path, e.strerror))

        for file_path in imgs_files_testing:
            try:
                os.remove(file_path)
            except OSError as e:
                print("Error: %s : %s" % (file_path, e.strerror))

        try:
            os.rmdir(path_training)
        except OSError as e:
            print("Error: %s: %s" % (path_training, e.strerror))

        try:
            os.rmdir(path_testing)
        except OSError as e:
            print("Error: %s: %s" % (path_training, e.strerror))


class HW4Net(nn.Module):
    def __init__(self, net):
        super(HW4Net, self).__init__()
        self.net = net
        if self.net == 'Net1':
            self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, ker
nel_size=3)
            self.pool = nn.MaxPool2d(kernel_size=2,stride=2)
            self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, ke
rnel_size=3)
            # INPUT FEATURES CALCULATED BY 'FLATTENING THE OUTPUT OF C
ONV2 AND POOL
            # FINAL OUTPUT AFTER THAT LAYER HAS SHAPE BATCHSIZE X 32 X
16 X 16 (CONV2D FORMULA)
            # THEREFORE NUMBER OF INPUT PARAMETERS ARE 32*16*16
            self.fc1 = nn.Linear(in_features=32 * 14 * 14, out_feature
s=64)
            # OUTPUT FEATURES OF LAST LAYER SHOULD BE NUMBER OF DESIRE
D CATEGORIES TO DISCRIMINATE FROM
            # IN THIS CASE, IT IS 5 CLASSES
            self.fc2 = nn.Linear(in_features=64, out_features=5)
        elif self.net == 'Net2':
            self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, ker
nel_size=3, padding=1)
            self.pool = nn.MaxPool2d(kernel_size=2,stride=2)
            self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, ke
rnel_size=3, padding=1)
            # INPUT FEATURES CALCULATED BY 'FLATTENING THE OUTPUT OF C
```

```
ONV2 AND POOL
            # FINAL OUTPUT AFTER THAT LAYER HAS SHAPE BATCHSIZE X 32 X
16 X 16 (CONV2D FORMULA)
            # THEREFORE NUMBER OF INPUT PARAMETERS ARE 32*16*16
            self.fc1 = nn.Linear(in_features=32 * 16 * 16, out_feature
s=64)
            # OUTPUT FEATURES OF LAST LAYER SHOULD BE NUMBER OF DESIRE
D CATEGORIES TO DISCRIMINATE FROM
            # IN THIS CASE, IT IS 5 CLASSES
            self.fc2 = nn.Linear(in_features=64, out_features=5)
        elif self.net == 'Net3':
            self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, ker
nel_size=3, padding=1)
            self.pool = nn.MaxPool2d(kernel_size=2,stride=2)
            self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, ke
rnel_size=3, padding=1)
            self.conv_layers = nn.ModuleList()
            for _ in range(10):
                self.conv_layers.append(nn.Conv2d(in_channels=32,out_c
hannels=32,kernel_size=3,padding=1))
            # INPUT FEATURES CALCULATED BY 'FLATTENING THE OUTPUT OF 1
0 LAYERS OF CONVOLUTION
            # FINAL OUTPUT AFTER THAT LAYER HAS SHAPE BATCHSIZE X 32 X
16 X 16 (CONV2D FORMULA)
            # THEREFORE NUMBER OF INPUT PARAMETERS ARE 32*16*16
            self.fc1 = nn.Linear(in_features=32 * 16 * 16, out_feature
s=64)
            # OUTPUT FEATURES OF LAST LAYER SHOULD BE NUMBER OF DESIRE
D CATEGORIES TO DISCRIMINATE FROM
            # IN THIS CASE, IT IS 5 CLASSES
            self.fc2 = nn.Linear(in_features=64, out_features=5)

    def forward(self, x):
        if self.net == 'Net1':
            # x originally has size batchsize x 3 x 64 x 64
            x = self.pool(F.relu(self.conv1(x))) # x now has size batc
hsize x 16 x 31 x 31
            x = self.pool(F.relu(self.conv2(x))) # x now has size batc
hsize x 32 x 14 x 14
            x = x.view(x.shape[0], -1)
            x = F.relu(self.fc1(x))
            x = self.fc2(x)
            return x
        elif self.net == 'Net2':
            # x originally has size batchsize x 3 x 64 x 64
            x = self.pool(F.relu(self.conv1(x))) # x now has size batc
hsize x 16 x 32 x 32
            x = self.pool(F.relu(self.conv2(x))) # x now has size batc
hsize x 32 x 16 x 16
            x = x.view(x.shape[0], -1)
```

```
            x = F.relu(self.fc1(x))
            x = self.fc2(x)
            return x
        elif self.net == 'Net3':
            # x originally has size batchsize x 3 x 64 x 64
            x = self.pool(F.relu(self.conv1(x))) # x now has size batc
hsize x 16 x 32 x 32 - Original Convolution # 1
            x = self.pool(F.relu(self.conv2(x))) # x now has size batc
hsize x 32 x 16 x 16 - Original Convolution # 2
            for m in self.conv_layers:
                x = F.relu(m(x))
            x = x.view(x.shape[0], -1)
            x = F.relu(self.fc1(x))
            x = self.fc2(x)
            return x

def model_training(net, epochs, train_data_loader, device, save_path):
    training_loss = []
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(net.parameters(), lr=1e-3, betas=(0.9
, 0.99))
    print("Begin Training...\n")
    net = net.to(device)
    net.train()
    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.item()
            if (i + 1) % 100 == 0:
                print("[epoch: %d, batch: %5d] loss: %3f" % (epoch + 1
, i + 1, running_loss / 100))
                training_loss.append(running_loss / 100)
                running_loss = 0.0

    print("Finished Training!\n")
    torch.save(net.state_dict(), save_path)

    return training_loss

def model_testing(net, test_data_loader, batch_size, device, desired_c
ats, save_path):
```

```
    # Part of the code Borrowed from Dr. Kak's DL Studio Module
    print("Begin Evaluation...\n")
    net.load_state_dict(torch.load(save_path))
    net.eval()
    correct, total = 0, 0
    confusion_matrix = torch.zeros(5, 5)
    class_correct = [0] * 5
    class_total = [0] * 5
    with torch.no_grad():
        for i, data in enumerate(test_data_loader):
            inputs, labels = data
            inputs = inputs.to(device)
            labels = labels.to(device)
            output = net(inputs)
            _, predicted = torch.max(output.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            comp = predicted == labels
            for label, prediction in zip(labels, predicted):
                confusion_matrix[label][prediction] += 1

            for j in range(batch_size):
                label = labels[j]
                class_correct[label] += comp[j].item()
                class_total[label] += 1

    for j in range(5):
        print('Prediction accuracy for %5s : %2d %%' % (desired_cats[j
], 100 * class_correct[j] / class_total[j]))
    print("Finished Evaluation!\n")
    print('Accuracy of the network on 2500 test images: {}%'.format(10
0 * float(correct / total)))

    return confusion_matrix, float(correct / total)


def cf_matrix_plot(cf_matrix, title, desired_cats):

    sns.heatmap(cf_matrix, annot=True, cmap='Blues', fmt='g', xticklab
els=desired_cats, yticklabels=desired_cats)
    plt.xlabel('Predicted Labels')
    plt.ylabel('Ground Truth')
    plt.title(title)
    plt.show()

def dataset_plot(classes, num_imgs, train_dataset, idx_list, desired_c
ats, title):

    img_net_mean = [0.5, 0.5, 0.5]
    img_net_std = [0.5, 0.5, 0.5]
```

```
    fig, axes = plt.subplots(nrows=classes, ncols=num_imgs)
    for i in range(classes):
        for j in range(num_imgs):
            img, _ = train_dataset[idx_list[i * num_imgs + j]]
            for l in range(3):
                img[l] = (img[l] * img_net_std[l]) + img_net_mean[l]
            img = img.numpy().transpose(1, 2, 0)
            if i == 0:
                axes[i, j].set_title('Image # {}'.format(j + 1))
            if j == 0:
                axes[i, j].set_ylabel(desired_cats[i])
            axes[i, j].imshow(img)
            axes[i, j].set_yticks([0], labels=[])
            axes[i, j].set_xticks([0], labels=[])
            axes[i, j].tick_params(axis=u'both', which=u'both', length
=0)

    fig.suptitle(title)
    plt.show()


def train_loss_plot(train_loss, title):

    plt.plot(train_loss[0], 'r', label='Original Convolutional Net1')
    plt.plot(train_loss[1], 'b', label='Convolutional with Padding Net
2')
    plt.plot(train_loss[2], 'g', label='Extra 10 Convolutional Layers
Net3')
    plt.title(title)
    plt.legend(loc="upper right")
    plt.xlabel("Iterations")
    plt.ylabel("Loss")
    plt.show()


# Set seeds
seed = 123
random.seed(seed)
np.random.seed(seed)

# See if GPU is available
if torch.backends.mps.is_available() == True:
    device = torch.device("mps")
else:
    device = torch.device("cpu")

# Print out current device (either GPU or CPU)
print("Currently using: {}".format(device))
```

```
training_size = 1500 # Amount of training images (per category)
testing_size = 500 # Amount of testing images (per category)
new_size = (64, 64) # New Image ssize

# List of Directories
root_images = "/Users/castil12/Documents/ECE 60146/Homework # 4/train2
014"
root_annotations = "/Users/castil12/Documents/ECE 60146/Homework # 4/a
nnotations/instances_train2014.json"
parent_dir_root_train = "/Users/castil12/Documents/ECE 60146/Homework
# 4/CustomCOCODataset/Training"
parent_dir_root_testing = "/Users/castil12/Documents/ECE 60146/Homewor
k # 4/CustomCOCODataset/Testing"
directory_name_training = "CustomCOCOTraining"
directory_name_testing = "CustomCOCOTesting"
save_path = "/Users/castil12/Documents/ECE 60146/Homework # 4/"

# Need 1500 Training Images & 500 Testing Images of Each Category
# Desired Categories of Images
desired_cats = ['airplane', 'bus', 'cat', 'dog', 'pizza']

start_time = time.time()

# Create directory which will host each image
directory_creation(desired_cats, parent_dir_root_train, parent_dir_roo
t_testing)

# Count if training and testing directories have been populated
count = file_count(desired_cats, parent_dir_root_train, parent_dir_roo
t_testing)
print("File count for Training and Testing: {}".format(count))

# Instantiate COCO
coco = COCO(root_annotations)

# Populate folders with training and testing images
data_creation(count, coco, desired_cats, training_size, testing_size,
new_size, root_images,
              parent_dir_root_train, parent_dir_root_testing)


# Create training and testing datasets
train_dataset = CuratedCOCO(parent_dir_root_train, desired_cats)
test_dataset = CuratedCOCO(parent_dir_root_testing, desired_cats)

# Need to plot 3 images of 5 classes
idx_list = []
j = 0

for cat in range(len(desired_cats)):
```

```
    for idx in range(len(train_dataset)):
        _, label = train_dataset[idx]
        if label == cat:
            idx_list.append(idx)
            j += 1
        if j == 3:
            j = 0
            break

# Plot 3 images of each class from Curated Dataset
dataset_plot(classes=5, num_imgs=3, train_dataset=train_dataset, idx_l
ist=idx_list,
             desired_cats=desired_cats, title='Selection of Images fro
m Curated COCO')

# Create train and test dataloaders
train_data_loader = torch.utils.data.DataLoader(train_dataset, batch_s
ize = 4, shuffle=True)
test_data_loader = torch.utils.data.DataLoader(test_dataset, batch_siz
e = 4, shuffle=False)

# Create Network # 1 and generate confusion matrix plot
net1 = HW4Net("Net1")
epochs_net1 = 15
batch_size_net1 = 4
training_loss_net1 = model_training(net1, epochs_net1, train_data_load
er, device, os.path.join(save_path, 'net1.pth'))
confusion_matrix_net1, testing_acc_net1 = model_testing(net1, test_dat
a_loader, batch_size_net1, device, desired_cats,
                                    os.path.join(save_path, 'net1.pth'))
cf_matrix_plot(confusion_matrix_net1, 'Confusion Matrix for Net1, Accu
racy: {}'.format(testing_acc_net1), desired_cats)

# Create Network # 2 and generate confusion matrix plot
net2 = HW4Net("Net2")
epochs_net2 = 15
batch_size_net2 = 4
training_loss_net2 = model_training(net2, epochs_net2, train_data_load
er, device, os.path.join(save_path, 'net2.pth'))
confusion_matrix_net2, testing_acc_net2 = model_testing(net2, test_dat
a_loader, batch_size_net2, device, desired_cats,
                                    os.path.join(save_path, 'net2.pth'))
cf_matrix_plot(confusion_matrix_net2, 'Confusion Matrix for Net2, Accu
racy: {}'.format(testing_acc_net2), desired_cats)

# Create Network # 3 and generate confusion matrix plot
net3 = HW4Net("Net3")
epochs_net3 = 15
batch_size_net3 = 4
training_loss_net3 = model_training(net3, epochs_net3, train_data_load
```

```
er, device, os.path.join(save_path, 'net3.pth'))
confusion_matrix_net3, testing_acc_net3 = model_testing(net3, test_dat
a_loader, batch_size_net3, device, desired_cats,
                                    os.path.join(save_path, 'net3.pth'))
cf_matrix_plot(confusion_matrix_net3, 'Confusion Matrix for Net3, Accu
racy: {}'.format(testing_acc_net3), desired_cats)

# Append losses to plot
train_loss = []
train_loss.append(training_loss_net1)
train_loss.append(training_loss_net2)
train_loss.append(training_loss_net3)
train_loss_plot(train_loss, 'Training Loss Comparison for each Net')


# Remove images and directories after training
#remove_files(desired_cats, parent_dir_root_train, parent_dir_root_tes
ting)
end_time = time.time()

print("Finished executing program. Elapsed time: {}".format(end_time -
start_time))
```