Object Classification Description and Details

1. Introduction:

In this assignment we were supposed to work with the MS COCO(Microsoft Common Objects in Context) dataset to create a subset of the dataset's training data, and used that to train 3 convolutional neural networks for image classification. It is one of the most widely used benchmarks for computer vision, especially object detection, segmentation, and captioning tasks.

Our task for this assignment was to create a subset from the COCO training data of 2000 images and run a CNNs similar to the one defined in experiment_with_CIFAR() in DLStudio.

2. Implementation:

For the implementation the below mentioned steps(they are elaborated on later in the document) were followed:

1. **Environment Creation**: Create a conda environment for working, the required python version and all the required libraries(pytorch, numpy, matplotlib, seaborn etc.) are installed in the new created environment.
2. **Data Generation**: The first actual step, was to download the COCO data set training data for the year 2017 and the corresponding annotations and then create a function to parse through these files to generate our training and testing data.
3. **Dataset and DataLoader:** Once the train and test data is generated, we create the pytorch dataset and dataloader so that the generated data can be fed into the neural networks that we create.

4. **Create Neural Network:** Post that, we design the neural networks as mentioned in the document, the first one is a very simple CNN, the second one is the same CNN but with padding=1 added and the final one is the same CNN with the 10 more convolutional layers chained before the first fully connected layer.
5. **Training, Testing and Visualization:** Once this is done, I wrote the code for model training, testing and plotting the confusion matrix and training loss.

2.1 Environment Creation

For setting up the environment and package management, I am using Conda. To create the environment I used the basic conda commands in the terminal:

```
conda create --name HomeWork4Env python=3.10
conda activate HomeWork4Env
conda install pytorch==2.1.1 torchvision==0.16.1 torchaudio==
2.1.1 -c pytorch
conda install numpy scipy pandas scikit-learn matplotlib seab
orn
conda install -c conda-forge pycocotools
```

Pycocotools is a package that is wrapped around the COCO API, that allows access to the COCO dataset and a number of data access options, the most important of which for us was accessing the labels of corresponding images.

And as I went ahead if need for some other package arose, or one was missing, I would just call the below code:

```
conda install <Package Name>
or
conda install -c conda-forge <Package Name>
```

2.2 Data Generation:

For this step, I used a function that would parse through the downloaded MSCOCO dataset, create the training and testing directories if they do not exists, then using the Pycocotools library, we access the labels of the corresponding file names of the images, and on the basis of the label, we allocate the images to a folder with the label name, in both, training and testing directories. We need to create a data set of 2000 images, so for the same, we have 1600 training images and 400 testing/validation images.

```
dataDir='/Users/avnishkanungo/Desktop/coco-dataset/train2017/
train2017'
dataType='train2017'
annFile='/Users/avnishkanungo/Desktop/coco-dataset/train2017/
train2017/annotations/instances_train2017.json'.format(dataDi
r,dataType)

coco_cnn = COCO(annFile) #Load annotations in the memory and
index them
```

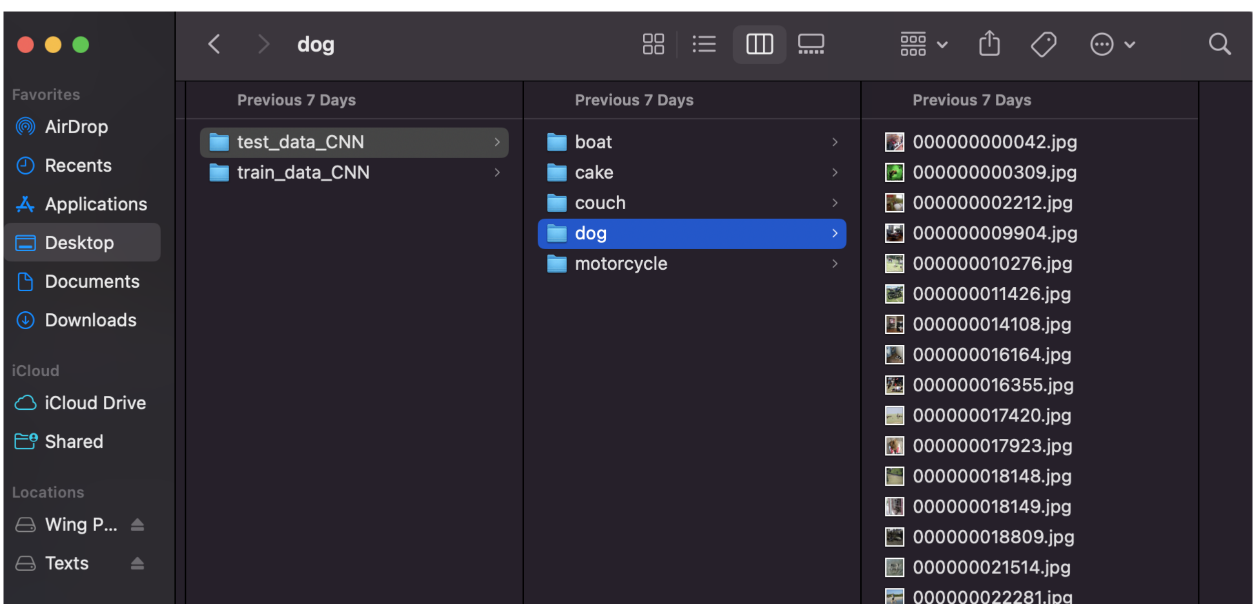
```


## This code has been created by taking reference of last yea
r's homework: https://engineering.purdue.edu/DeepLearn/2_best
_solutions/2023/Homeworks/HW4/2BestSolutions/1.pdf
def create_train_test_data(classes, coco):
    # create folders
    data_dir="/Users/avnishkanungo/Desktop/coco-dataset/train
2017/train2017" #define the directory where the complete COCO
2017 traing images are

    # define location of the directories where the training a
nd test data that we are going to us is to be saved, structur
ed as per their lable
    for c in classes:
        training_path = "/Users/avnishkanungo/Desktop/coco-da
taset/train2017/hw4_dataset/train_data_CNN" + "/" + c
        testing_path = "/Users/avnishkanungo/Desktop/coco-dat
```

```
aset/train2017/hw4_dataset/test_data_CNN" + "/" + c

        # creating directories if they do not exist
        if not os.path.exists(training_path):
            os.makedirs(training_path)
        if not os.path.exists(testing_path):
            os.makedirs(testing_path)

        catIds = coco.getCatIds(catNms=c) # get category ids,
i.e. the IDs to reference the class of images we want
        training_data = dict(zip(classes, [[] for i in range
(len(classes))]))
        testing_data = dict(zip(classes, [[] for i in range(l
en(classes))]))
        print(f"dataset generation has started for {c} ")
        for i, idx in enumerate(catIds):
            imgIds = coco.getImgIds(catIds=idx) # get the ima
ge Ids for the category id in each iteration
            imgIds = np.random.choice(imgIds, size=2000, repl
ace=False) #randomly select 2000 the image Ids for the catego
ry id in each iteration
            for j, jdx in enumerate(imgIds):
                image_file_name = coco.loadImgs(int(jdx))[0]
['file_name']
                img_file_path = os.path.join(data_dir, f"trai
n2017/{image_file_name}")
                img = Image.open(img_file_path).convert("RG
B") #Iterate over and load the random 2000 images, ensure all
images are RGB as the neural net to be used is configured for
3 channel input
                img = img.resize((64, 64)) # resize to 64x64
                save_name = image_file_name
                if j < 1599: #Iteration to save the first 160
0 images to Training directory
                    save_dir = "/Users/avnishkanungo/Desktop/
coco-dataset/train2017/hw4_dataset/train_data_CNN" + "/" + c
```

```
                    training_data[classes[i]].append(image_file_name)
                else: #Iteration to save the remainign 400 images to Testing directory
                    save_dir = "/Users/avnishkanungo/Desktop/coco-dataset/train2017/hw4_dataset/test_data_CNN" + "/" + c
                    testing_data[classes[i]].append(image_file_name)
                img.save(os.path.join(save_dir, save_name))
    print("train and validation datasets are ready!")
```

```
classes = ['boat', 'couch', 'dog', 'cake', 'motorcycle']
create_train_test_data(classes, coco_cnn)
```

```
dataset generation has started for boat
dataset generation has started for couch
dataset generation has started for dog
dataset generation has started for cake
dataset generation has started for motorcycle
train and validation datasets are ready!
```



2.3 Datset and Dataloader:

For this step we use the Pytorch functionality from torch.utils.data, called dataset and dataloader, which allows creation of data to be directly ingested into a neural network. The dataset class will be created to access the data and the dataloader call will be used to load this data into the neural network in a parallel manner. We will be creating separate instances of dataset and dataloader for the training and testing data, but the dataloader class where the data will be accessed and transformed into tensors will be common. The code for the same can be found below:

```
## This code has been created by taking reference of last yea
r's homework: https://engineering.purdue.edu/DeepLearn/2_best
_solutions/2023/Homeworks/HW4/2BestSolutions/1.pdf
####CODE To SETUP TORCH DATASET##############################
##########
class myCOCODataSet(Dataset):
    def __init__(self, file_path, classes, transform=None):
        super (). __init__ ()
        self.file_path = file_path
        self.classes = classes
        self.image_path = []
        self.class_label = []
        for c in classes:
            image_file_path = os.path.join(file_path, c) #cre
ating path with label name
            image_label = self.classes.index(c) #file name sa
ved as lable
            for l in os.listdir(image_file_path):
                self.image_path.append(os.path.join(image_fil
e_path,l)) #configuring the path for each image in the class
                self.class_label.append(image_label) #directo
ry name saved as label

        self.transform = transform
```

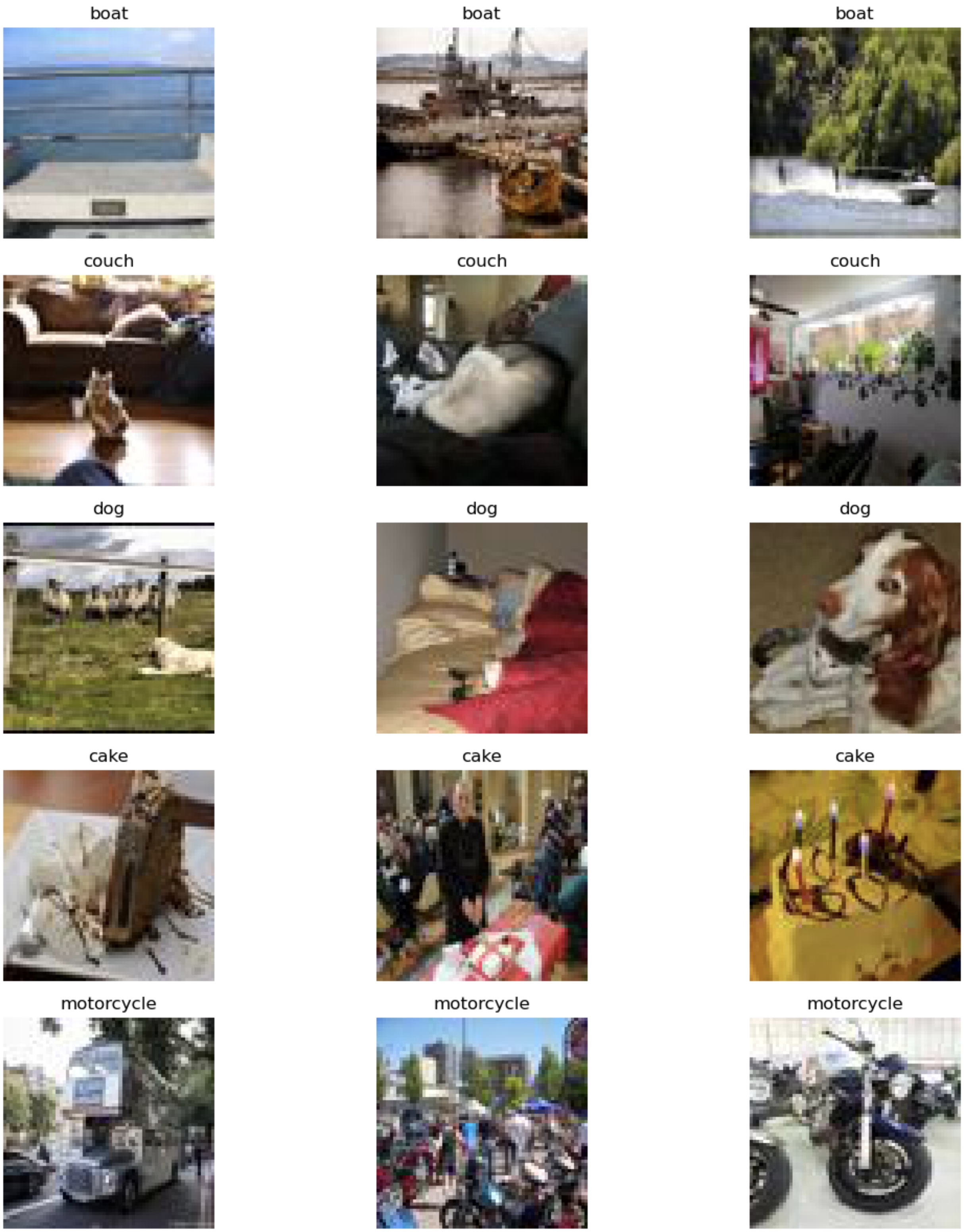
```
    def __len__(self):
        return len(self.class_label)

    def __getitem__(self, idx):
        image_name_path = self.image_path[idx]
        actual_image_label = self.class_label[idx]
        actual_image = Image.open(image_name_path)
        if self.transform:
           actual_image = self.transform(actual_image)

        return actual_image, actual_image_label
```

```
file_path_train = '/Users/avnishkanungo/Desktop/coco-dataset/
train2017/hw4_dataset/train_data_CNN'
file_path_test = '/Users/avnishkanungo/Desktop/coco-dataset/t
rain2017/hw4_dataset/test_data_CNN'

xform = tvt.Compose([
    tvt.ToTensor()])               #Configuring the transform

my_dataset_test = myCOCODataSet(file_path = file_path_test, c
lasses = classes ,transform = xform )
my_dataset_train = myCOCODataSet(file_path = file_path_train,
classes = classes ,transform = xform )

train_dataloader = DataLoader(my_dataset_train, batch_size =
4, shuffle=True)
test_dataloader = DataLoader(my_dataset_test,batch_size = 4,
shuffle=True)
```

2.3.3 Plotting the images to check the data:

Images are a little blurred, as we had to resize them to 64×64, some of the images might be black and white, but have been converted to RGB in dataset generation to make sure that there are 3 channels for the input Neural Net Layer.

```
def plot_train_test_image_sample(dataset,classes_plot):
    # Create a dictionary to store indices of images for each
class
    class_indices = {class_name: [] for class_name in classes
_plot}

    # Iterate through the dataset to collect indices for each
class
    for idx, (image, label) in enumerate(dataset):
        class_indices[classes[label]].append(idx)

    # Display 5 random images from each class
    num_samples_per_class = 3
    fig, axes = plt.subplots(len(classes), num_samples_per_cl
ass, figsize=(12, 12))

    for i, class_name in enumerate(classes):
        indices = class_indices[class_name]
        random_indices = np.random.choice(indices, num_sample
s_per_class, replace=False)

        for j, idx in enumerate(random_indices):
            image, label = dataset[idx]
            image = np.transpose(image.numpy(), (1, 2, 0))
            axes[i, j].imshow(image)
            axes[i, j].set_title(class_name)
            axes[i, j].axis('off')

    plt.tight_layout()
    plt.show()

plot_train_test_image_sample(my_dataset_train, classes)
```


boat
boat
boat
couch
couch
couch
dog
dog
dog
cake
cake
cake
motorcycle
motorcycle
motorcycle

2.4 Create, Train and Test Neural Network:

Directions to test with 3 different types of neural networks has been given in the documentation on the basis of which we have created the below:

```
## This code has been taken from Professor Kak' DL Studio Mod
el(https://engineering.purdue.edu/kak/distDLS/) and
## and in part also configured by referenceing previous yea
r's homework.
class HW4Net(nn.Module):
    def __init__(self, net):
        super(HW4Net, self).__init__()
        self.net = net
        #net.apply(weights_init)
        if self.net == 'Net1':
            self.conv1 = nn.Conv2d(3, 16, 3) #62 X 62 X 16
            self.pool = nn.MaxPool2d(2,2) ## 31 X 31 X 16
            self.conv2 = nn.Conv2d(16, 32, 3) ## 28 X 28 X 32
            self.fc1 = nn.Linear( 14 * 14 * 32, 64) ## 14 X 1
4 X 32 post second maxpool
            self.fc2 = nn.Linear(64, 5) #as there are 5 class
es
        elif self.net == 'Net2':
            self.conv1 = nn.Conv2d(3, 16, 3, 1) #64 X 64 X 16
            self.pool = nn.MaxPool2d(2,2) # 32 X 32 X 16
            self.conv2 = nn.Conv2d(16, 32, 3, 1) # 32 X 32 X
32
            self.fc1 = nn.Linear(16 * 16 * 32, 64) ## 16 X 16
X 32 post second maxpool
            self.fc2 = nn.Linear(64, 5)
        elif self.net == 'Net3':
            self.conv1 = nn.Conv2d(3, 16, 3, 1)
            self.pool = nn.MaxPool2d(2, 2)
            self.conv2 = nn.Conv2d(16, 32, 3, 1)
            self.conv_layers = nn.ModuleList()
            self.conv_layers = nn.ModuleList([nn.Conv2d(32, 3
```

```
2, 3, 1) for _ in range(10)])
            self.fc1 = nn.Linear(16 * 16 * 32, 64) ## 6 X 16
X 32 post second maxpool
            self.fc2 = nn.Linear(64, 5)

    def forward(self, x):
        if self.net == 'Net1':
            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
        elif self.net == 'Net2':
            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
        elif self.net == 'Net3':
            x = self.pool(F.leaky_relu(self.conv1(x))) #Leaky
relu used here in effeor to minimze the vanishing gradient pr
oblem.
            x = self.pool(F.leaky_relu(self.conv2(x))) #Leaky
relu used here in effeor to minimze the vanishing gradient pr
oblem.
            for conv_layer in self.conv_layers:
                x = F.leaky_relu(conv_layer(x)) #Leaky relu u
sed here in effeor to minimze the vanishing gradient problem.
            x = x.view(x.shape[0], -1)
            x = F.leaky_relu(self.fc1(x))
            x = self.fc2(x)
            return x
```

The above code just defines the structure of the neural network, to train and evaluate the our data using the same we will need to write the logic for training and testing/evaluation of the data that has been ingested into the neural network.

For calculating the output dimensions from the convolutional layer for each of the cases I used the below formula, form the pytorch official documentation():

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

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

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

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html#torch.nn.Conv2d

So for each of the networks:

1. Net 1: Input(64 X 64 X 3) → Conv2d(62 X 62 X 16) → MaxPool(31 X 31 X 16) → Conv2d(28 X 28 X 32) → MaxPool(14 X 14 X 32)

   XXXX = 6272, XX=5

2. Net 2: Input(64 X 64 X 3) → Conv2d(64 X 64 X 16) → MaxPool(32 X 32 X 16) → Conv2d(32 X 32 X 32) → MaxPool(16 X 16 X 32)

   XXXX = 8192, XX=5

3. Net 3: Input(64 X 64 X 3) → Conv2d(64 X 64 X 16) → MaxPool(32 X 32 X 16) → Conv2d(32 X 32 X 32) → MaxPool(16 X 16 X 32)

   XXXX = 8192, XX=5

2.4.1 Model Training

The training logic here(which has been picked from the DL studio Module) defines the cost function, the optimizer, the device to run the model and number of epochs for the model to run, using these values, it runs forward passes and backwards passes in batches for the data and repeats the process for the number of epochs defined and calculates the traing loss based on the cost function and optimizer.

The cost function in use here is the Cross Entropy Loss Function which is used primarily for classification tasks and the optimizer in use is the ADAM optimizer with learning rate 1e-3 and beta1 and beta 2 as 0.9 and 0.99. It takes as input the model type(Net1, Net2, Net3) and the data is ingested using the data loadetr.

Once the training is complete we the model using the torch.save function in the .pth format.

```
## This code has been taken from Professor Kak' DL Studio M
odel(https://engineering.purdue.edu/kak/distDLS/) and
## and in part also configured by referenceing previous yea
r's homework.

def model_training(net, epochs, train_data_loader, device,
save_path):
    training_loss = []
    criterion = torch.nn.CrossEntropyLoss() #Loss Function
    optimizer = torch.optim.Adam(net.parameters(), lr=1e-3,
betas=(0.9, 0.99)) #Optimizer
    print("Begin Training...\n")
    net = net.to(device) #Configure 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() #initializer gradient val
ues to zero
            outputs = net(inputs) #forward pass
            loss = criterion(outputs, labels) # loss calcul
ated as per the cost function
```

```
            loss.backward() #backpropogation of the loss
            optimizer.step() #step optimization as per the
optimizer configured
            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
```

2.4.2 Model Testing

The model testing/validation will be applied to the saved model that we have saved in the .pth, pytorch format. Once we load up the model, we initialize an array that will define the confusion matrix for evaluation of the classification. We then iterate over the images that will be loaded using the data loader, and input them into the trained model and evaluate the output by comparing them with the actual label, by mapping the testing output with the actual label for that image, and also returns the accuracy of the model.

We also have a function for creating the confusion matrix defined, which takes the confusion matrix defined in the model testing function as an input.

```
def plot_confusion_martix(cf_matrix, title, desired_cats):
    sns.heatmap(cf_matrix, annot=True, cmap='Greens', fmt
='g', xticklabels=desired_cats, yticklabels=desired_cats)
        plt.title(title)
    plt.xlabel('Predicted Labels')
    plt.ylabel('Actual Labels')
    plt.show()
```

```
## This code has been taken from Professor Kak' DL Studio M
odel(https://engineering.purdue.edu/kak/distDLS/) and
## and in part also configured by referenceing previous yea
r's homework.

def model_testing(net, test_data_loader, batch_size, devic
e, desired_cats, save_path):
    print("Begin Evaluation...\n")
    net.load_state_dict(torch.load(save_path)) #load the sa
ved model from the .pth file
    net.eval()
    correct, total = 0, 0
    confusion_matrix = torch.zeros(5, 5) #initialize the ma
trix for confusion matrix creation
    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) #extra
cting index of the max value, as that is basically the labe
l we are looking for
                        for label, prediction in zip(label
s, predicted):
                confusion_matrix[label][prediction] += 1
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            comp = predicted == labels
            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 %%' % (des
ired_cats[j], 100 * class_correct[j] / class_total[j]))
    print("Finished Evaluation!\n")
    print('Accuracy of the network on 2000 test images:
{}%'.format(100 * float(correct / total)))
    plot_confusion_martix(confusion_matrix, 'Confusion Matr
ix', desired_cats)
    return confusion_matrix, float(correct / total)
```

2.4.3 Calling the Model:

The below code defines how the model is called.

```
net1 = HW4Net("Net1")
epochs_net1 = 15
batch_size_net1 = 4
net1_training_loss = model_training(net1, epochs_net1, trai
n_dataloader, torch.device("mps"), os.path.join('/Users/avn
ishkanungo/Desktop/coco-dataset/train2017', 'net1.pth'))
confusion_matrix_net1, net1_testing_acc = model_testing(net
1, test_dataloader, batch_size_net1, torch.device("mps"), c
lasses,os.path.join('/Users/avnishkanungo/Desktop/coco-data
set/train2017', 'net1.pth'))


net2 = HW4Net("Net2")
epochs_net2 = 15
batch_size_net2 = 4
net2_training_loss = model_training(net2, epochs_net2, trai
n_dataloader, torch.device("mps"), os.path.join('/Users/avn
ishkanungo/Desktop/coco-dataset/train2017', 'net2.pth'))
confusion_matrix_net2, net2_testing_acc = model_testing(net
2, test_dataloader, batch_size_net2, torch.device("mps"), c
lasses,os.path.join('/Users/avnishkanungo/Desktop/coco-data
```

```
set/train2017', 'net2.pth'))


def weights_init(m): # to prevent vanishing gradient proble
m in Net3
    if isinstance(m, nn.Conv2d):
        torch.nn.init.xavier_uniform_(m.weight)

net3 = HW4Net("Net3")
net3.apply(weights_init)
epochs_net3 = 15
batch_size_net3 = 4
net3_training_loss = model_training(net3, epochs_net3, trai
n_dataloader, torch.device("mps"), os.path.join('/Users/avn
ishkanungo/Desktop/coco-dataset/train2017', 'net3.pth'))
confusion_matrix_net3, net3_testing_acc = model_testing(net
3, test_dataloader, batch_size_net3, torch.device("mps"), c
lasses,os.path.join('/Users/avnishkanungo/Desktop/coco-data
set/train2017', 'net3.pth'))
```

3. Results

Parameter Calculation[1]:

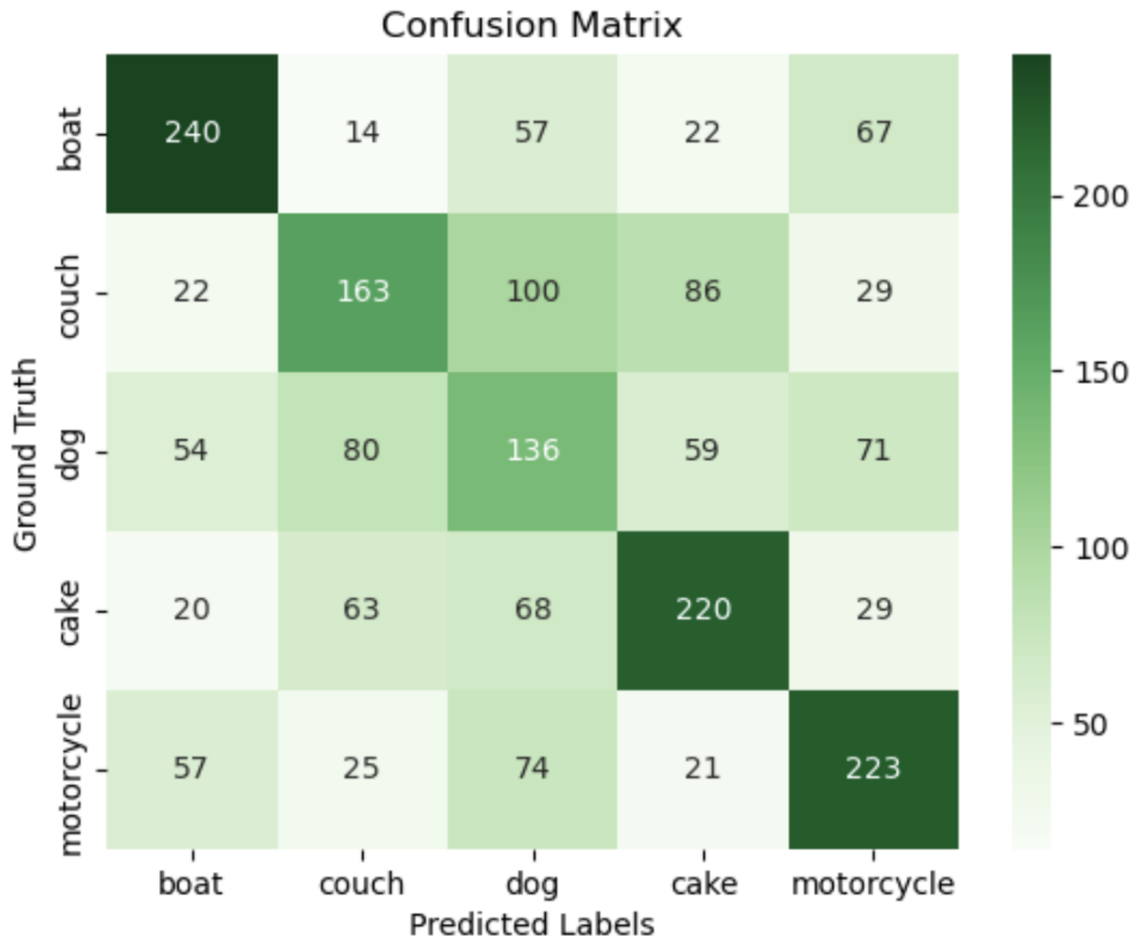
```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.r
equires_grad)

count_parameters(net1)
count_parameters(net2)
count_parameters(net3)
```

Network	No. of Parameters	Classification Accuracy
Net1	`406885`	`49.1%`
Net2	`529765`	`51.7%`
Net3	`622245`	`51.4%`

3.1 Confusion Martix

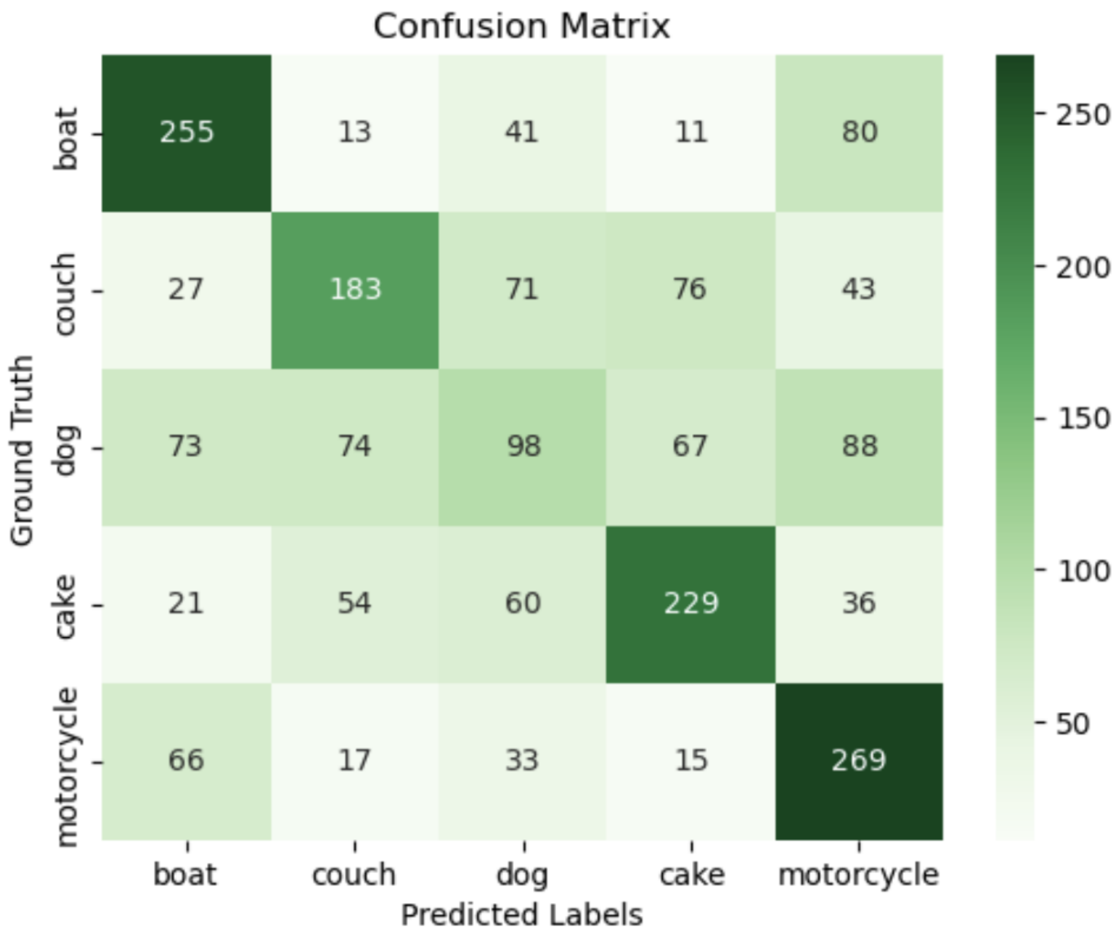
3.1.1 Net 1:

```
Begin Evaluation...

Prediction accuracy for  boat : 60 %
Prediction accuracy for couch : 40 %
Prediction accuracy for   dog : 34 %
Prediction accuracy for  cake : 55 %
Prediction accuracy for motorcycle : 55 %
Finished Evaluation!

Accuracy of the network on 2000 test images: 49.1%
```



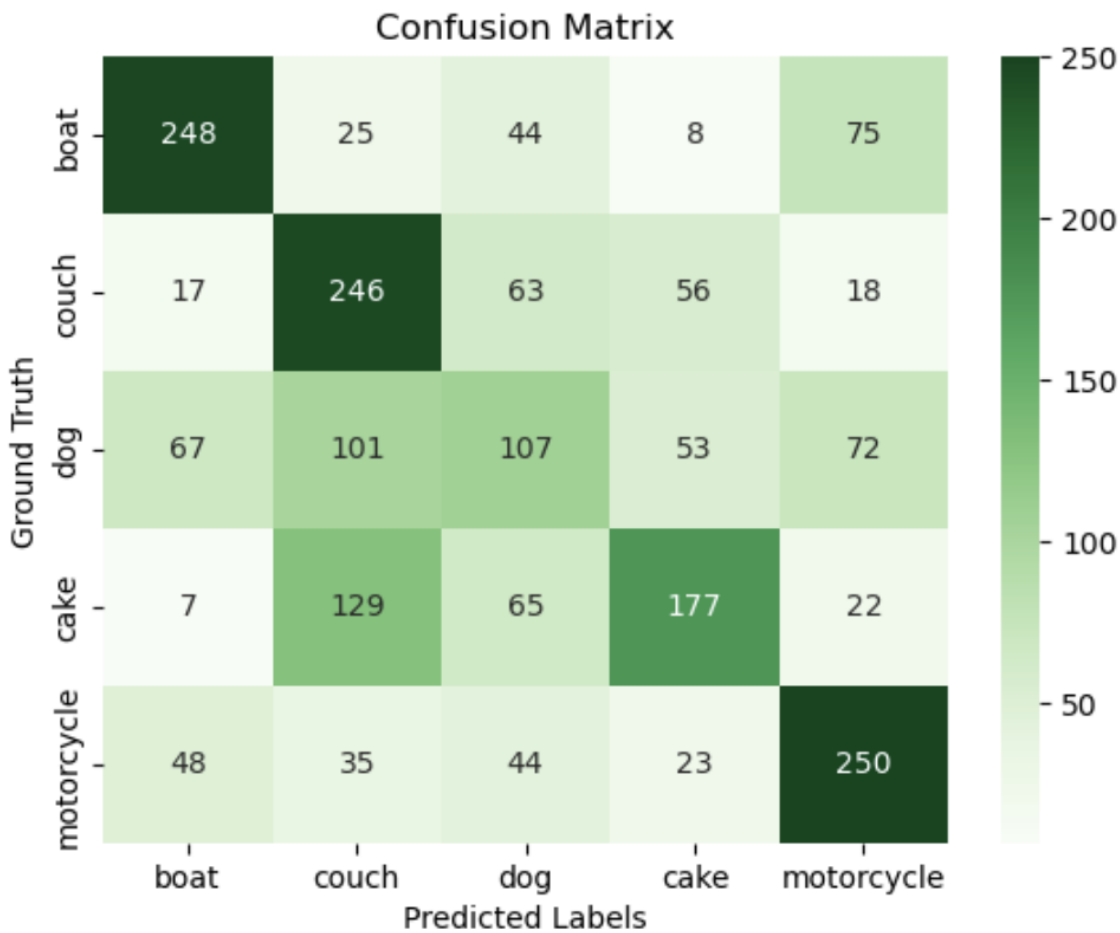
3.1.2 Net2：

```
Prediction accuracy for  boat : 63 %
Prediction accuracy for couch : 45 %
Prediction accuracy for   dog : 24 %
Prediction accuracy for  cake : 57 %
Prediction accuracy for motorcycle : 67 %
Finished Evaluation!

Accuracy of the network on 2000 test images: 51.7%
```

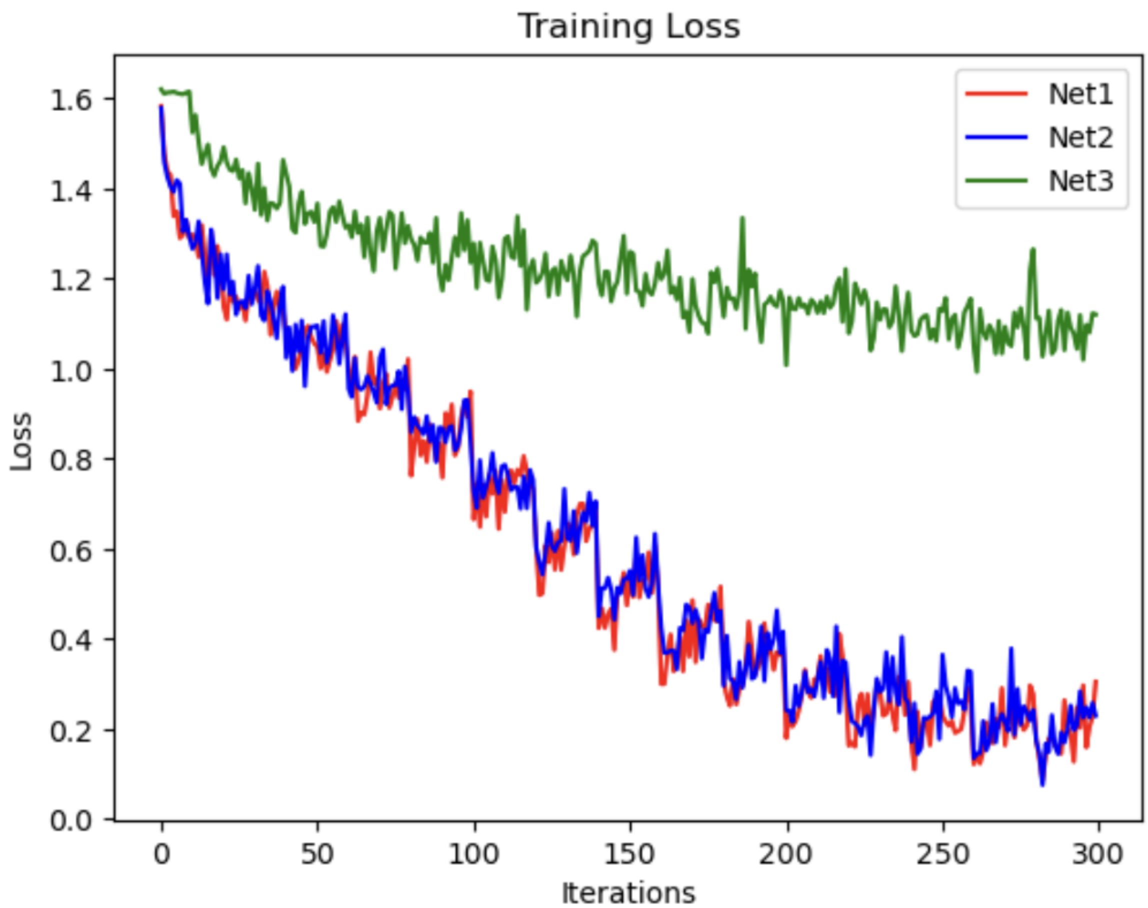


3.1.3 Net3:

```
Begin Evaluation...

Prediction accuracy for  boat : 62 %
Prediction accuracy for couch : 61 %
Prediction accuracy for   dog : 26 %
Prediction accuracy for  cake : 44 %
Prediction accuracy for motorcycle : 62 %
Finished Evaluation!

Accuracy of the network on 2000 test images: 51.4%
```



3.2 Training Loss Plot:



Training Loss
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Loss
0
50
100
150
200
250
300
Iterations
Net1
Net2
Net3