HOMEWORK 5

Alexandre Olive Pellicer

Building the model

I have created the model as an extension of the BMEnet model, the model that is already implemented in DLStudio and used for training the CIFAR_10 dataset. Since the number of classes in the CIFAR_10 dataset is 10, the output of the last linear layer is 10. In our case, since we are working with a dataset which contains 5 classes, we fixed the output of the last linear layer to 5. Furthermore, the size of the images from CIFAR_10 dataset is 32x32 while the size of the images from our dataset is 64x64. For that reason,

The CIFAR_10 dataset contains images of size 32x32 for 10 different classes. Our dataset has images of size 64x64. For that reason, in our dataset we downsampled the images one more time and we also decided to increase the number of channels to 256 following the pattern of BMEnet.

When referring to the BMEnet, I am referring to the initial implementation given when the instructions where submitted. After trying different architectures, I saw it was the one it was giving me better results.

Find at the end of the report the class HW5Net which corresponds to the model used in this homework.

Total number of learnable layers in your network

We print the number of learnable layers that our network has using:

```
num_layers = len( list ( net3.parameters () ) )
print("number of learnable layers: ", num_layers)
```

The obtained result is 188

Collect training loss for at least two different learning rates and include the plots in your report.

The loss curve for lr=1e-4 and lr=5e-4 when training it for 15 epochs is the following:

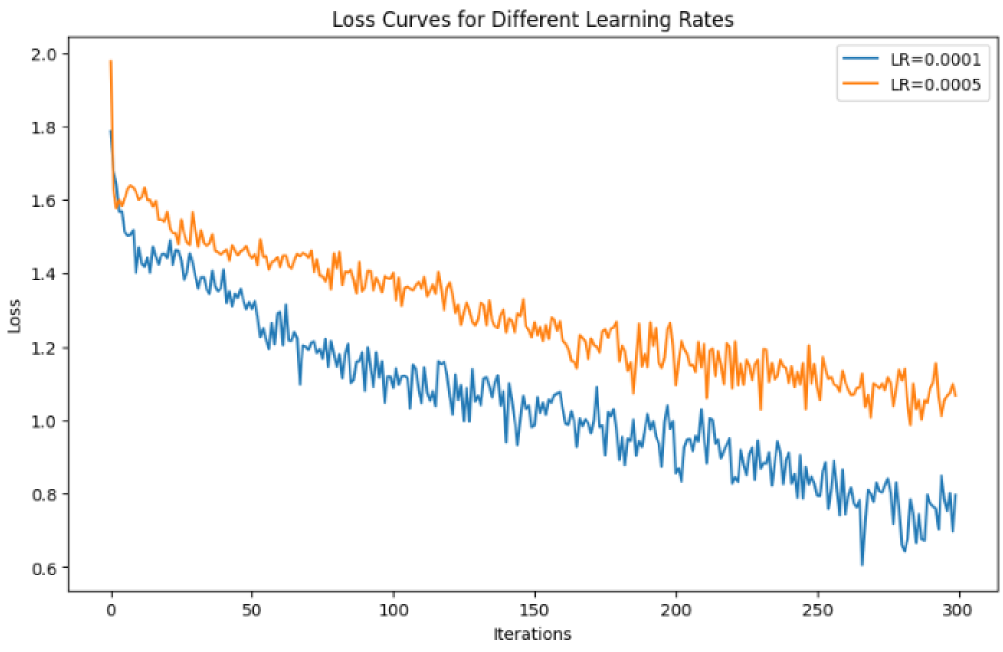


Fig 1: Loss curves for lr = 1e-4 and lr = 5e-4 after 15 epochs

I also saw that by increasing the number of epochs by 60, the training loss could be reduced to values closer to 0. These are the plots obtained by training for 60 epochs using again lr=1e-4 and lr=5e-4:

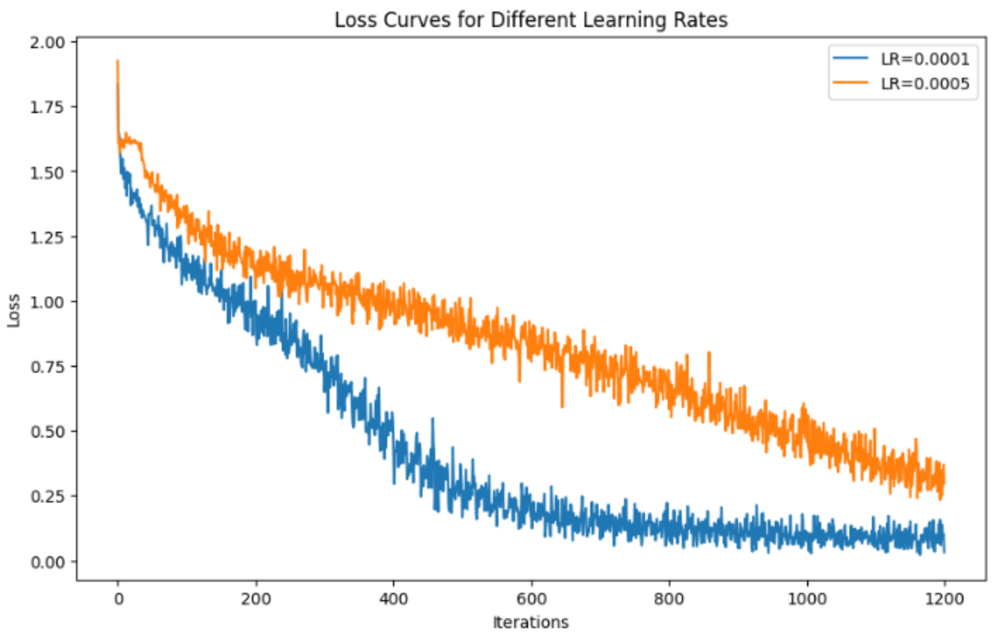


Fig 2: Loss curves for lr = 1e-4 and lr = 5e-4 after 60 epochs

Report the accuracy numbers and the confusion matrix for different learning rates on the test dataset.

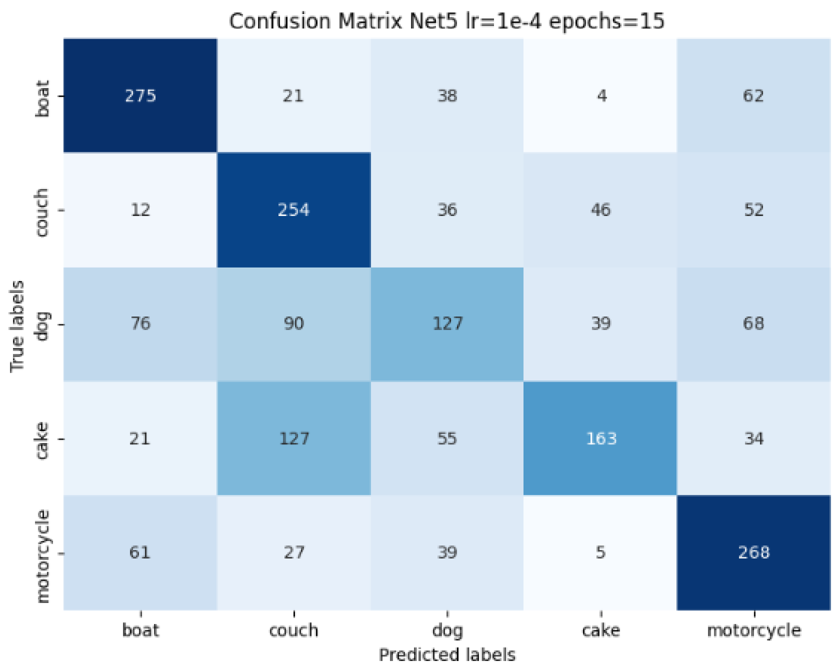


Fig 3: Confusion Matrix of Net5 using a lr=1e-4 and 15 epochs

15 epochs		60 epochs	
Validation accuracy	Training accuracy	Validation accuracy	Training accuracy
0.54	0.66	0.52	0.96

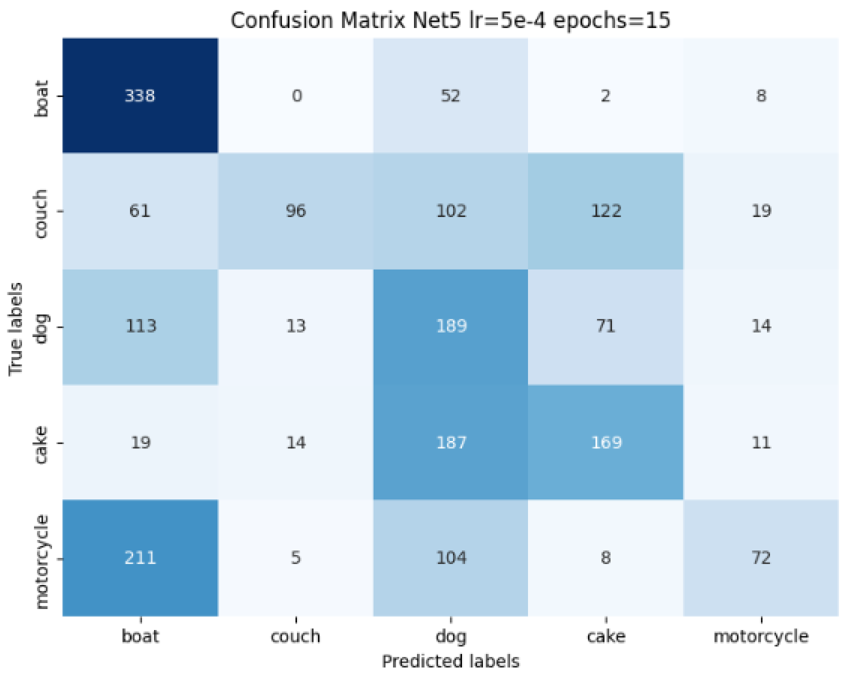


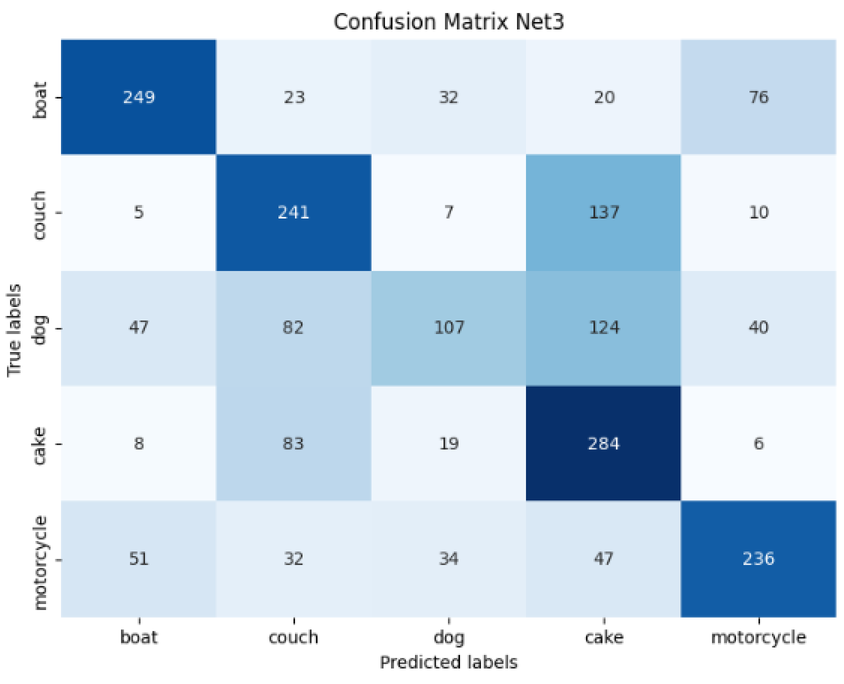
Fig 4: Confusion Matrix of Net5 using a lr=5e-4 and 15 epochs

15 epochs		60 epochs	
Validation accuracy	Training accuracy	Validation accuracy	Training accuracy
0.43	0.44	0.43	0.59

From the obtained results we can say that the best performance is achieved when using lr = 1e-4 and training for 15 epochs since we achieve an accuracy of 54%. From the loss curves we could expect that the models trained after 60 epochs should perform better since a lower value of loss was reached. Nevertheless, looking at the Training accuracy for lr = 1e-4 and 60 epochs we see that we have 96%, a very high value which shows us that we have a case of overfitting and that the model is not able to generalize. Thus, the performance over the validation dataset is not that good.

State your observations regarding the classification performance of HW5Net in comparison with what you achieved previously with Net3 in HW4. Also, attach your confusion matrix of Net3 from HW4.

The confusion matrix of Net3 from HW4 is:



True \ Predicted	boat	couch	dog	cake	motorcycle
boat	249	23	32	20	76
couch	5	241	7	137	10
dog	47	82	107	124	40
cake	8	83	19	284	6
motorcycle	51	32	34	47	236

Fig 5: Confusion matrix of Net3 from HW4

I will consider Net5 the model we have trained with lr=1e-4 for 15 epochs which is the one that gave us a better accuracy: 54%

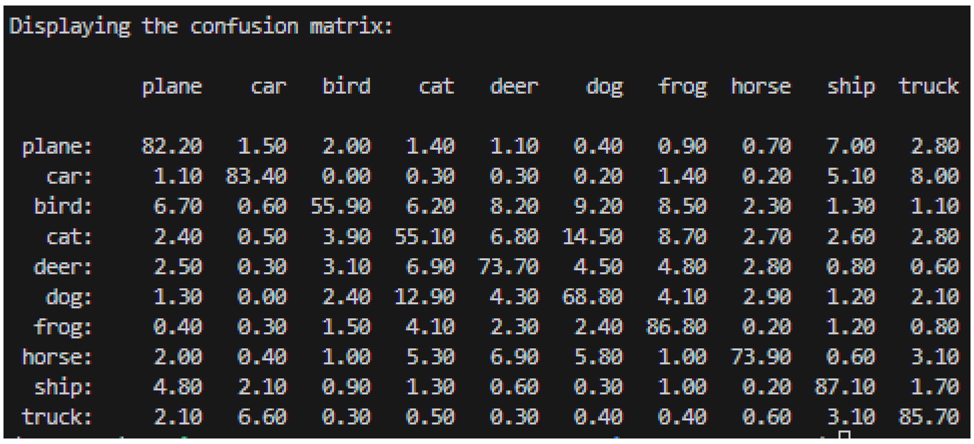
Looking at the loss curve obtained with Net5, we can see that the loss value converges towards a low value. This didn't happen with Net3 where we were in front of a case of a vanishing gradient. In Net5 we have prevented a vanishing gradient case by using skip connections. This allows us to train deeper neural networks. In our case, Net5 has 188 learnable layers, a much bigger number compared to the number of learnable layers used in Net3 which was lower than 30.

In terms of accuracy, we see that the performance hasn't improved that much. Both in Net3 and Net5 the validation accuracy obtained is inside the range 50%-60%. About the capacity of labeling correctly a specific class, we see that in both Net3 and Net5 the class which is more times misclassified is the dog. As we mentioned in hw4, dog is the class that is more difficult to correctly label because probably it is more prompt to appear to images where there is also a "boat" or a "couch" or a "cake" or a "motorcycle" and we are considering as a bad classification although it is correctly done. It is also curious to see that Net5 classifies a considerable number of cakes as couches while in Net3 is in the other way, it classifies a considerable number of couches as cakes. It is difficult to find an explainable reason for these behaviors although we may think that they share some features. For example, both cakes and couches are trend to be in indoors rooms.

As mentioned in HW4, our training a testing dataset have limitations of reduced number of samples and images with more than one class. This makes it more difficult to obtain the desired results. When targeting improving the performance of a deep learning model, the data has a lot to do.

Optional: You may also attach the confusion matrix generated for CIFAR dataset. State your observations in comparison to COCO dataset performance.

The confusion matrix obtained when using the CIFAR dataset is the following:



```
Displaying the confusion matrix:

          plane    car   bird    cat   deer    dog   frog  horse   ship  truck

 plane:   82.20   1.50   2.00   1.40   1.10   0.40   0.90   0.70   7.00   2.80
   car:    1.10  83.40   0.00   0.30   0.30   0.20   1.40   0.20   5.10   8.00
  bird:    6.70   0.60  55.90   6.20   8.20   9.20   8.50   2.30   1.30   1.10
   cat:    2.40   0.50   3.90  55.10   6.80  14.50   8.70   2.70   2.60   2.80
  deer:    2.50   0.30   3.10   6.90  73.70   4.50   4.80   2.80   0.80   0.60
   dog:    1.30   0.00   2.40  12.90   4.30  68.80   4.10   2.90   1.20   2.10
  frog:    0.40   0.30   1.50   4.10   2.30   2.40  86.80   0.20   1.20   0.80
 horse:    2.00   0.40   1.00   5.30   6.90   5.80   1.00  73.90   0.60   3.10
  ship:    4.80   2.10   0.90   1.30   0.60   0.30   1.00   0.20  87.10   1.70
 truck:    2.10   6.60   0.30   0.50   0.30   0.40   0.40   0.60   3.10  85.70
```

Fig 6: Confusion matrix for the CIFAR_10 dataset

The obtained accuracy in this case is 75%.

We can see that when using the CIFAR dataset the obtained accuracy is better than the results obtained with the COCO dataset that we have created by ourselves. In this case we obtain better accuracy than when using the COCO dataset. The difference in accuracy is significant which makes us think that one of the main problems for not improving our training accuracy is the dataset with which we are working (i.e. our dataset has small training and testing datasets, and it contains images where more than a class can be present)

A 75% of training accuracy can be considered a good result. From the confusion matrix we can also see that the model makes mistakes when classifying cats and dogs. This is because they are very similar, and they share features. It also happens, although with less frequency, with the horse. Cats, dogs, and horses are animals with 4 legs, for this reason it is understandable the mistakes when classifying them. Something similar happens with planes and birds. Both have 2 wings and are found probably in the sky. For this reason, there are also mistakes when classifying them.

CODE:

```
import sys,os,os.path
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
import os
from torch.utils.data import DataLoader
import torch
import torch.nn as nn
from tqdm import tqdm
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import torchvision.transforms as tvt
from PIL import Image
import torch.nn.functional as F
import copy

from DLStudio import *

os.environ['CUDA_VISIBLE_DEVICES']='5'

## USED THE MYDATASET CLASS USED IN PREVIOUS HOMEWORKS ------------------
--------------------------------
class MyDataset ( torch.utils.data.Dataset ):

    def __init__ ( self , root ):
        super().__init__()
        # Obtain meta information (e.g. list of file names )
        self.root = root
        self.filenames = os.listdir(self.root)

        # Initialize data transforms , etc.
        self.transform = tvt.Compose([
            tvt.ToTensor(),
            tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
        ])

    def __len__ ( self ):
        # Return the total number of images
        # the number is a place holder only
        return len(self.filenames)

    def __getitem__ ( self , index ):
        # Read an image at index and perform processing
        path = os.path.join(self.root, self.filenames[index])
        cls = self.filenames[index].split('_')[0]
        img = Image.open(path)

        # Get the normalized tensor
        transformed_img = self.transform(img)

        if cls == "boat":
            label = torch.tensor([1.0, 0.0, 0.0, 0.0, 0.0])
        elif cls == "couch":
            label = torch.tensor([0.0, 1.0, 0.0, 0.0, 0.0])
        elif cls == "dog":
            label = torch.tensor([0.0, 0.0, 1.0, 0.0, 0.0])
        elif cls == "cake":
            label = torch.tensor([0.0, 0.0, 0.0, 1.0, 0.0])
        elif cls == "motorcycle":
            label = torch.tensor([0.0, 0.0, 0.0, 0.0, 1.0])

        # Return the tuple : ( normalized tensor, label )
        return transformed_img, label

## COPPIED THE LAST SKIPBLOCK CLASS IMPLEMENTATION PROVIDED BY PROFESSOR
KAK------------------------------------------------------
class SkipBlock(nn.Module):
    """
    Class Path:   DLStudio  ->  SkipConnections  ->  SkipBlock
    """
    def __init__(self, in_ch, out_ch, downsample=False,
skip_connections=True):
        super(SkipBlock, self).__init__()
        self.downsample = downsample
        self.skip_connections = skip_connections
        self.in_ch = in_ch
        self.out_ch = out_ch
        self.convo1 = nn.Conv2d(in_ch, in_ch, 3, stride=1, padding=1)
        self.convo2 = nn.Conv2d(in_ch, out_ch, 3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(in_ch)
        self.bn2 = nn.BatchNorm2d(out_ch)

        self.in2out  =  nn.Conv2d(in_ch, out_ch, 1)

        if downsample:
            self.downsampler1 = nn.Conv2d(in_ch, in_ch, 1, stride=2)
            self.downsampler2 = nn.Conv2d(out_ch, out_ch, 1, stride=2)

    def forward(self, x):
        identity = x

        out = self.convo1(x)
        out = self.bn1(out)
        out = nn.functional.relu(out)

        out = self.convo2(out)
        out = self.bn2(out)
        out = nn.functional.relu(out)

        if self.downsample:
            identity = self.downsampler1(identity)
            out = self.downsampler2(out)

        if self.skip_connections:
            if (self.in_ch == self.out_ch) and (self.downsample is
False):
                out = out + identity
            elif (self.in_ch != self.out_ch) and (self.downsample is
False):
                identity = self.in2out( identity
)                             ###  <<<<  from  Cheng-Hao Chen
                out = out + identity
            elif (self.in_ch != self.out_ch) and (self.downsample is
True):
                out = out + torch.cat((identity, identity), dim=1)

        return out

## COPPIED THE FIRST BMENET CLASS IMPLEMENTATION AVAILABLE IN DLSTUDIO
AND EXTEND IT TO CREATE HW5NET ------------------------------------------
---------
class HW5Net(nn.Module):
    """
    Class Path:   DLStudio  ->  SkipConnections  ->  HW5Net
    """
    def __init__(self, skip_connections=True, depth=32):
        super(HW5Net, self).__init__()
        if depth not in [8, 16, 32, 64]:
            sys.exit("HW5Net has been tested for depth for only 8, 16,
32, and 64")
        self.depth = depth // 8
        self.conv = nn.Conv2d(3, 64, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)

        self.skip64_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip64_arr.append(SkipBlock(64,
64,skip_connections=skip_connections))
        self.skip64ds = SkipBlock(64, 64,downsample=True,
skip_connections=skip_connections)
        self.skip64to128 = SkipBlock(64,
128,skip_connections=skip_connections )

        self.skip128_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip128_arr.append(SkipBlock(128,
128,skip_connections=skip_connections))
        self.skip128ds = SkipBlock(128, 128,downsample=True,
skip_connections=skip_connections)
        self.skip128to256 = SkipBlock(128,
256,skip_connections=skip_connections )

        self.skip256_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip256_arr.append(SkipBlock(256, 256,
skip_connections=skip_connections))
        self.skip256ds = SkipBlock(256,256,downsample=True,
skip_connections=skip_connections)

        self.fc1 =  nn.Linear(1024, 500)
        self.fc2 =  nn.Linear(500, 5)

    # I implement the forward method as an extension of the given BMEnet
given. I downsample the input images one more time and I also increase
the number of channels one more time.
    # Finally I adjust the arguments of the linear layers
    def forward(self, x):
        x = self.pool(nn.functional.relu(self.conv(x)))

        for i,skip64 in enumerate(self.skip64_arr[:self.depth//4]):
            x = skip64(x)
        x = self.skip64ds(x)
        for i,skip64 in enumerate(self.skip64_arr[self.depth//4:]):
            x = skip64(x)
        x = self.skip64ds(x)
        x = self.skip64to128(x)
        for i,skip128 in enumerate(self.skip128_arr[:self.depth//4]):
            x = skip128(x)
        x = self.skip128ds(x)
        for i,skip128 in enumerate(self.skip128_arr[self.depth//4:]):
            x = skip128(x)
        x = self.skip128ds(x)
        x = self.skip128to256(x)

        for i,skip256 in enumerate(self.skip256_arr[:self.depth//4]):
            x = skip256(x)
        for i,skip256 in enumerate(self.skip256_arr[self.depth//4:]):
            x = skip256(x)

        x  =  x.view( x.shape[0], - 1 )
        x = nn.functional.relu(self.fc1(x))
        x = self.fc2(x)
        return x

## SCRIPT TO RUN THE TRAINING -------------------------------------------
--------
device = "cuda:0"

# Dataloader
my_dataset = MyDataset ("../../../COCOTraining")
batch_size = 4
train_data_loader = DataLoader(my_dataset, batch_size = batch_size,
shuffle = True )


# List where the loss values will be stored
loss_net3 = []

net3 = HW5Net()
# Training routine provided by the assignment
net3 = net3.to(device)

# Print number of learnable layers
num_layers = len( list ( net3.parameters () ) )
print("number of learnable layers: ", num_layers)

criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net3.parameters(), lr=5e-4, betas=(0.9,
0.99))
epochs = 60
for epoch in tqdm(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 = net3(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))
            loss_net3.append(running_loss/100)
            running_loss = 0.0

# Save the learned parameters of the model to do inference afterwards
torch.save(net3.state_dict(), './net5_replicate_60epochs.pth')

## PLOT TRAINING LOSS----------------------------------------------------
------------------------------
plt.plot(loss_net3)

plt.legend(["loss_net3"])

# Adding labels and title
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Loss comparisson for the 3 networks')

# Display the plot
plt.show()

## TESTING AND CONFUSSION MATRIX (We run this code for each network)-----
----------------------------------------------
device = "cuda:0"

# Dataloader loading the Validation dataset
my_dataset = MyDataset("../COCOValidation")
batch_size = 4
train_data_loader = DataLoader(my_dataset, batch_size = batch_size,
shuffle = True )

# Lists where the labels will be stored for each of the images from the
Validation dataset
predictions = []
real_labels = []

# Load the trained weights
net3 = HW5Net()
net3.load_state_dict(torch.load('net5_replicate_60epochs.pth'))
```

```
net3 = net3.to(device)

# Set the model to evaluation mode
net3.eval()

# Get the predicted label and the real label from each image and store
them to the lists mentioned before
for i, data in enumerate(train_data_loader):
    inputs, labels = data
    inputs = inputs.to(device)
    labels = labels.to(device)
    with torch.no_grad():
        outputs = net3(inputs)
    outputs = outputs.split(1)
    labels = labels.split(1)
    for lbl in labels:
        real_labels.append(torch.argmax(lbl.squeeze()).item())
    for pred in outputs:
        predictions.append(torch.argmax(pred.squeeze()).item())

# Compute the confusion matrix
cm = confusion_matrix(real_labels, predictions)

# Create a heatmap for visualization
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', cbar=False,
xticklabels=['boat','couch','dog', 'cake', 'motorcycle'],
yticklabels=['boat','couch','dog', 'cake', 'motorcycle'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix Net3')
plt.show()

# Print classification report for additional metrics like accuracy
print("Classification Report:\n", classification_report(real_labels,
predictions))
```