ECE695DL: Homework 4

Vaibhav Ramachandran

15 March 2021

hw04_coco_downloader.py

```
# -------------------------------------------------

# ECE 69500 - Deep Learning
# Hw-04 COCO Image Downloader
# Vaibhav Ramachandran

# -------------------------------------------------

# -------------------------------------------------
# library imports

import torchvision
import torch
import torch.utils.data
import glob
import os
import sys
import numpy
import PIL
import argparse
import requests
import logging
import json
import random

from PIL import Image
from pycocotools.coco import COCO
from requests.exceptions import ConnectionError, ReadTimeout,
TooManyRedirects, MissingSchema, InvalidURL
```

```
# end of library imports
# -------------------------------------------------

# -------------------------------------------------
# code to ensure reproducibility

seed = 0
random.seed(seed)
torch.manual_seed(seed)
numpy.random.seed(seed)
torch.backends.cudnn.deterministic=True
torch.backends.cudnn.benchmarks=False
os.environ['PYTHONHASHSEED'] = str(seed)

# end of code to ensure reproducibility
# -------------------------------------------------

# -------------------------------------------------
# parser setup

parser = argparse.ArgumentParser(description='HW04 COCO
Downloader')
parser.add_argument ('--class_list', nargs='*', type=str,
required=True)
parser.add_argument ('--images_per_class', type=int, required=True)
parser.add_argument ('--root_path', type=str, required=True)
parser.add_argument ('--coco_json_path', type=str, required=True)
args, args_other = parser.parse_known_args()

# end of parser setup
# -------------------------------------------------

# -------------------------------------------------
# get image function definition

def get_image(img_url, class_folder, image_name):
"This function downloads and downsamples an image from a given
url and stores it in the given class folder"

try :
img_resp = requests.get(img_url, timeout = 1)
# handle all exceptions
except ConnectionError:
logging.debug(f"Connection Error for url {img_url}")
return 0
except ReadTimeout:
```

```
logging.debug(f"Read Timeout for url {img_url}")
return 0
except TooManyRedirects:
logging.debug(f"Too Many Redirects for url {img_url}")
return 0
except MissingSchema:
logging.debug(f"Missing Schema for url {img_url}")
return 0
except InvalidURL:
logging.debug(f"Invalid url {img_url}")
return 0

img_file_path = os.path.join(class_folder, image_name)

if os.path.isfile(img_file_path):
# checking if image has already been downloaded
return 0

if os.path.exists(img_file_path):
# redundant check to see if file has already been
downloaded
return 0

with open (img_file_path, 'wb') as img_f:
img_f.write(img_resp.content)

# resize image to 64x64
im = Image.open(img_file_path)
if im.mode != "RGB":
im = im.convert(mode = "RGB")
im_resized = im.resize((64,64), Image.BOX)

# overwrite original image with downsampled image
im_resized.save(img_file_path)
return 1

# end of get image function definition
# ------------------------------------------------

# ------------------------------------------------
# main code body

class_list = args.class_list
images_per_class = args.images_per_class
coco_root = args.root_path
coco_json_path = args.coco_json_path
```

```
# setting the folder path for the training/validation images
if not os.path.exists(coco_root):
os.makedirs(coco_root)

coco = COCO(coco_json_path)

for class_name in class_list:
class_folder = os.path.join(coco_root, class_name)
if not os.path.exists(class_folder):
os.mkdir(class_folder)
# specifying the categories of interest
catIds = coco.getCatIds(catNms=class_name)
# get the corresponding image ids and images using loadImgs
imgIds = coco.getImgIds(catIds=catIds)
images = coco.loadImgs(imgIds)

downloaded_images = 0

# Save the images into a local folder
for im in images:
# download and downsample the required number of images
if(downloaded_images < images_per_class):
downloaded_images += get_image(im['coco_url'],
class_folder, im['file_name'])
else:
break

# end of main code body
# -------------------------------------------------
```

hw04_training.py

```
# -------------------------------------------------

# ECE 69500 - Deep Learning
# Hw-04 COCO Model Training Script
# Vaibhav Ramachandran

# -------------------------------------------------

# -------------------------------------------------
# library imports
```

```
import torchvision
import torch
import torch.utils.data
import glob
import sys,os,os.path
import numpy
import PIL
import argparse
import requests
import logging
import json
import random
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt

from torch.utils.data import DataLoader, Dataset
from PIL import Image
from torchvision import transforms
from requests.exceptions import ConnectionError, ReadTimeout,
TooManyRedirects, MissingSchema, InvalidURL

# end of library imports
# -------------------------------------------------

# -------------------------------------------------
# code to ensure reproducibility

seed = 0
random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
numpy.random.seed(seed)
torch.backends.cudnn.deterministic=True
torch.backends.cudnn.benchmarks=False
os.environ['PYTHONHASHSEED'] = str(seed)

def seed_worker(worker_id):
numpy.random.seed(0)

# end of code to ensure reproducibility
# -------------------------------------------------

# -------------------------------------------------
# global definitions
```

```
dtype = torch.float64
device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
batch_size = 10
learning_rate = 1e-3
epochs = 10
image_size = [64,64]

# end of global definitions
# -------------------------------------------------

# -------------------------------------------------
# parser setup

parser = argparse.ArgumentParser(description='HW04 Training')
parser.add_argument ('--root_path', type=str, required=True)
parser.add_argument ('--class_list', nargs='*', type=str,
required=True)
args, args_other = parser.parse_known_args()

# end of parser setup
# -------------------------------------------------

# -------------------------------------------------
# custom dataset class definition

class ImageDataset( Dataset ):
def __init__(self, class_list, coco_root, transform):
self.class_list = class_list
self.coco_root = coco_root
self.transform = transform
self.image_path_list = []
self.image_label_list = []
i = 0
for class_name in self.class_list:
for filename in glob.glob(self.coco_root + '/' +
class_name + '/*.jpg'): # assuming jpg
self.image_path_list.append(filename)
self.image_label_list.append(i) # Ex: label: 0 -
refrigerator, 1 - airplane, 2 - ...
i += 1
self.dataset_size = len(self.image_path_list)

def __len__(self):
return self.dataset_size
```

```
def __getitem__(self, index):
image_pil_obj = Image.open(self.image_path_list[index])
image_data_tensor = self.transform(image_pil_obj)
image_label_tensor =
torch.tensor(self.image_label_list[index])
return image_data_tensor, image_label_tensor

# end of custom dataset class definition
# -------------------------------------------------

# -------------------------------------------------
# custom neural network class definition

class TemplateNet(nn.Module):
def __init__(self, net=1, class_list_size=10,
pool_kernel_size=2, pool_stride=2):
super(TemplateNet, self).__init__()
self.net = net
self.class_list_size = class_list_size
self.pool_kernel_size = pool_kernel_size
self.pool_stride = pool_stride
if(self.net == 3):
self.image_padding = 1
# pool_output_dim here is calculated after the
10x3x64x64 input
# tensor has been passed through the first and second
# convolution layers and twice through the max pooling
layer
# the output of the first convolution layer is a
10x128x64x64
# tensor (because image_padding is 1 now instead of 0.
so
# convolution of the image with a 3x3 filter will
preserve the
# input image dimensions along axes 0 and 1)
# the tensor dimensions after first passing through the

max pool
# layer is 10x128x32x32 (hence the division by
pool_stride in the
# calculation)
# the output of the second convolution layer is a
10x128x30x30
# tensor (hence the second -2 in the calculation)
# the tensor dimensions after passing through the max
pool layer
```

```
# a second and final time is 10x128x15x15 (hence the
division by
# pool_stride again in the calculation)
self.pool_output_dim = ((image_size[0] //
self.pool_stride) - 2) // self.pool_stride
elif(self.net == 2):
self.image_padding = 0
# pool_output_dim here is calculated after the
10x3x64x64 input
# tensor has been passed through the first and second
# convolution layers and twice through the max pooling
layer
# the output of the first convolution layer is a
10x128x62x62
# tensor (hence the -2 in the calculation)
# the tensor dimensions after first passing through
the max pool
# layer is 10x128x31x31 (hence the division by
pool_stride in the
# calculation)
# the output of the second convolution layer is a
10x128x29x29
# tensor (hence the second -2 in the calculation)
# the tensor dimensions after passing through the max
pool layer
# a second and final time is 10x128x14x14 (hence the
division by
# pool_stride again in the calculation)
self.pool_output_dim = (((image_size[0] - 2) //
self.pool_stride) - 2) // self.pool_stride
else:
self.image_padding = 0
# pool_output_dim here is calculated after the
10x3x64x64 input
# tensor has been passed through the first convolution
layer and
# the max pooling layer
# the output of the first convolution layer is a
10x128x62x62
# tensor (hence the -2 in the calculation)
# the final tensor dimensions after passing through
the
max pool
# layer is 10x128x31x31 (hence the division by
pool_stride in the
# calculation)
```

```
self.pool_output_dim = (image_size[0] - 2) //
self.pool_stride
self.conv1 = nn.Conv2d(3, 128, 3,
padding=self.image_padding) ## (A)
self.conv2 = nn.Conv2d(128, 128, 3) ## (B)
self.pool = nn.MaxPool2d(self.pool_kernel_size,
self.pool_stride)
self.fc1 = nn.Linear(128 * self.pool_output_dim *
self.pool_output_dim, 1000) ## (C)
self.fc2 = nn.Linear(1000, self.class_list_size)

def forward(self, x):
x = self.pool(F.relu(self.conv1(x)))
if(self.net != 1):
x = self.pool(F.relu(self.conv2(x))) ## (D)
x = x.view(-1, 128 * self.pool_output_dim *
self.pool_output_dim) ## (E)
x = F.relu(self.fc1(x))
x = self.fc2(x)
return x

# end of custom neural network class definition
# ------------------------------------------------

# ------------------------------------------------
# training function definition

def run_code_for_training(net, learning_rate, epochs,
train_data_loader, net_save_file_path):
loss_running_record = []
net = net.to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate,
momentum=0.9)
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) % 500 == 0:
print("\n[epoch:%d, batch:%5d] loss: %.3f" %
(epoch + 1, i + 1, running_loss / float(500)))
loss_running_record.append((running_loss
float(500)))
running_loss = 0.0
torch.save(net, net_save_file_path)
return loss_running_record

# end of training function definition
# ------------------------------------------------

# ------------------------------------------------
# main function definition

def main():
coco_root = args.root_path
class_list = args.class_list
class_list_size = len(class_list)
loss_running_records = []

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

# load and transform the training images dataset using a
dataloader
train_dataset = ImageDataset(class_list, coco_root, transform)
train_data_loader =
torch.utils.data.DataLoader(dataset=train_dataset,
batch_size=10, shuffle=True, num_workers=2,
worker_init_fn=seed_worker)

for net_number in [1,2,3]:
net_save_file_path = 'net' + str(net_number) + '.pth'
net = TemplateNet(net_number, class_list_size)
loss_running_record = run_code_for_training(net,
learning_rate, epochs,train_data_loader,
net_save_file_path)
loss_running_records.append(loss_running_record)

plt.figure()
for i in range(len(loss_running_records)):
legend_entry = 'Net' + str(i + 1) + ' Training Loss'
plt.plot(loss_running_records[i], label=legend_entry)
plt.xlabel('epochs')
plt.ylabel('loss')
```

```
plt.title('Training Loss')
plt.legend()
plt.grid()
plt.savefig('train_loss.jpg')

# end of main function definition
# ------------------------------------------------

# ------------------------------------------------
# main execution

if __name__=="__main__":
main()

# end of main execution
# ------------------------------------------------
```

hw04_validation.py

```
# ------------------------------------------------

# ECE 69500 - Deep Learning
# Hw-04 COCO Model Validation Script
# Vaibhav Ramachandran

# ------------------------------------------------

# ------------------------------------------------
# library imports

import torchvision
import torch
import torch.utils.data
import glob
import sys,os,os.path
import numpy
import PIL
import argparse
import requests
import logging
import json
import random
```

```
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
import seaborn as sns

from torch.utils.data import DataLoader, Dataset
from PIL import Image
from torchvision import transforms
from requests.exceptions import ConnectionError, ReadTimeout,
TooManyRedirects, MissingSchema, InvalidURL

# end of library imports
# -------------------------------------------------

# -------------------------------------------------
# code to ensure reproducibility

seed = 0
random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
numpy.random.seed(seed)
torch.backends.cudnn.deterministic=True
torch.backends.cudnn.benchmarks=False
os.environ['PYTHONHASHSEED'] = str(seed)

def seed_worker(worker_id):
numpy.random.seed(0)

# end of code to ensure reproducibility
# -------------------------------------------------

# -------------------------------------------------
# global definitions

dtype = torch.float64
device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
batch_size = 10
learning_rate = 1e-3
epochs = 10
image_size = [64,64]

# end of global definitions
# -------------------------------------------------
```

```
# ------------------------------------------------
# parser setup

parser = argparse.ArgumentParser(description='HW04 Validation')
parser.add_argument ('--root_path', type=str, required=True)
parser.add_argument ('--class_list', nargs='*', type=str,
required=True)
args, args_other = parser.parse_known_args()

# end of parser setup
# ------------------------------------------------

# ------------------------------------------------
# custom dataset class definition

class ImageDataset( Dataset ):
def __init__(self, class_list, coco_root, transform):
self.class_list = class_list
self.coco_root = coco_root
self.transform = transform
self.image_path_list = []
self.image_label_list = []
i = 0
for class_name in self.class_list:
for filename in glob.glob(self.coco_root + '/' +
class_name + '/*.jpg'): # assuming jpg
self.image_path_list.append(filename)
self.image_label_list.append(i) # Ex: label: 0 -
refrigerator, 1 - airplane, 2 - ...
i += 1
self.dataset_size = len(self.image_path_list)

def __len__(self):
return self.dataset_size

def __getitem__(self, index):
image_pil_obj = Image.open(self.image_path_list[index])
image_data_tensor = self.transform(image_pil_obj)
image_label_tensor =
torch.tensor(self.image_label_list[index])
return image_data_tensor, image_label_tensor

# end of custom dataset class definition
# ------------------------------------------------

# ------------------------------------------------
```

```
# custom neural network class definition

class TemplateNet(nn.Module):
def __init__(self, net=1, class_list_size=10,
pool_kernel_size=2, pool_stride=2):
super(TemplateNet, self).__init__()
self.net = net
self.class_list_size = class_list_size
self.pool_kernel_size = pool_kernel_size
self.pool_stride = pool_stride
if(self.net == 3):
self.image_padding = 1
# pool_output_dim here is calculated after the
10x3x64x64 input
# tensor has been passed through the first and second
# convolution layers and twice through the max pooling
layer
# the output of the first convolution layer is a
10x128x64x64
# tensor (because image_padding is 1 now instead of 0.
so
# convolution of the image with a 3x3 filter will
preserve the
# input image dimensions along axes 0 and 1)
# the tensor dimensions after first passing through
the max pool
# layer is 10x128x32x32 (hence the division by
pool_stride in the
# calculation)
# the output of the second convolution layer is a
10x128x30x30
# tensor (hence the second -2 in the calculation)
# the tensor dimensions after passing through the max
pool layer
# a second and final time is 10x128x15x15 (hence the
division by
# pool_stride again in the calculation)
self.pool_output_dim = ((image_size[0] //
self.pool_stride) - 2) // self.pool_stride
elif(self.net == 2):
self.image_padding = 0
# pool_output_dim here is calculated after the
10x3x64x64 input
# tensor has been passed through the first and second
# convolution layers and twice through the max pooling
layer
```

```
# the output of the first convolution layer is a
10x128x62x62
# tensor (hence the -2 in the calculation)
# the tensor dimensions after first passing through the
max pool
# layer is 10x128x31x31 (hence the division by
pool_stride in the
# calculation)
# the output of the second convolution layer is a
10x128x29x29
# tensor (hence the second -2 in the calculation)
# the tensor dimensions after passing through the max
pool layer
# a second and final time is 10x128x14x14 (hence the
division by
# pool_stride again in the calculation)
self.pool_output_dim = (((image_size[0] - 2) //
self.pool_stride) - 2) // self.pool_stride
else:
self.image_padding = 0
# pool_output_dim here is calculated after the
10x3x64x64 input
# tensor has been passed through the first convolution
layer and
# the max pooling layer
# the output of the first convolution layer is a
10x128x62x62
# tensor (hence the -2 in the calculation)
# the final tensor dimensions after passing through the
max pool
# layer is 10x128x31x31 (hence the division by
pool_stride in the
# calculation)
self.pool_output_dim = (image_size[0] - 2) //
self.pool_stride
self.conv1 = nn.Conv2d(3, 128, 3) ## (A)
self.conv2 = nn.Conv2d(128, 128, 3) ## (B)
self.pool = nn.MaxPool2d(self.pool_kernel_size,
self.pool_stride)
self.fc1 = nn.Linear(128 * self.pool_output_dim *
self.pool_output_dim, 1000) ## (C)
self.fc2 = nn.Linear(1000, self.class_list_size)

def forward(self, x):
x = self.pool(F.relu(self.conv1(x)))
if(self.net != 1):
```

```
x = self.pool(F.relu(self.conv2(x))) ## (D)
x = x.view(-1, 128 * self.pool_output_dim *
self.pool_output_dim) ## (E)
x = F.relu(self.fc1(x))
x = self.fc2(x)
return x

# end of custom neural network class definition
# -------------------------------------------------

# -------------------------------------------------
# validation function definition

def run_code_for_validation(net, class_list_size, val_data_loader):
with torch.no_grad():
correct = 0
confusion_matrix = numpy.zeros([class_list_size,
class_list_size], dtype=int)
for i, data in enumerate(val_data_loader):
# get the input features and corresponding labels
inputs, labels = data
# Shape of input is (m,a1,a2,a3,...)
# where m is the batch size

inputs = inputs.to(device)
labels = labels.to(device)

# forward pass
outputs = net(inputs)
for ground_truth, prediction in zip(labels,
outputs.max(1)[1]):
confusion_matrix[ground_truth][prediction] += 1

postives = outputs.max(1)[1] == labels
correct += postives.sum().item()

return confusion_matrix

# end of validation function definition
# -------------------------------------------------

# -------------------------------------------------
# function to plot confusion matrix

def plot_confusion_matrix(confusion_matrix, matrix_save_file_path,
categories='auto', cmap='Blues', title='Confusion Matrix'):
```

```
group_counts = ["{0:0.0f}\n".format(value) for value in
confusion_matrix.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
confusion_matrix.flatten()/numpy.sum(confusion_matrix)]

box_labels = [f"{v1}{v2}".strip() for v1, v2 in
zip(group_counts,group_percentages)]
box_labels = numpy.asarray(box_labels).reshape
(confusion_matrix.shape[0],confusion_matrix.shape[1])

#Accuracy is sum of diagonal divided by total observations
accuracy  = numpy.trace(confusion_matrix) /
float(numpy.sum(confusion_matrix))
stats_text = "\n\nAccuracy={:0.3f}".format(accuracy)

plt.figure(figsize=(10,10))
sns.heatmap(confusion_matrix,annot=box_labels,fmt="",
cmap=cmap,cbar=False,xticklabels=categories,
yticklabels=categories)

plt.ylabel('True label')
plt.xlabel('Predicted label' + stats_text)
plt.title(title)
plt.savefig(matrix_save_file_path)

# end of function to plot confusion matrix
# -----------------------------------------------

# -----------------------------------------------
# main function definition

def main():
coco_root = args.root_path
class_list = args.class_list
class_list_size = len(class_list)

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

# load and transform the validation images dataset using a
dataloader
val_dataset = ImageDataset(class_list, coco_root, transform)
val_data_loader =
torch.utils.data.DataLoader(dataset=val_dataset, batch_size=10,
```

```
shuffle=True, num_workers=2, worker_init_fn=seed_worker)

for net_number in [1,2,3]:
net_load_file_path = 'net' + str(net_number) + '.pth'
matrix_save_file_path = 'net' + str(net_number) +
'_confusion_matrix.jpg'
net = torch.load(net_load_file_path)
net.eval()
confusion_matrix = run_code_for_validation(net,
class_list_size, val_data_loader)
plot_confusion_matrix(confusion_matrix,
matrix_save_file_path, class_list)

# end of main function definition
# -------------------------------------------------

# -------------------------------------------------
# main execution

if __name__=="__main__":
main()

# end of main execution
# -------------------------------------------------
```

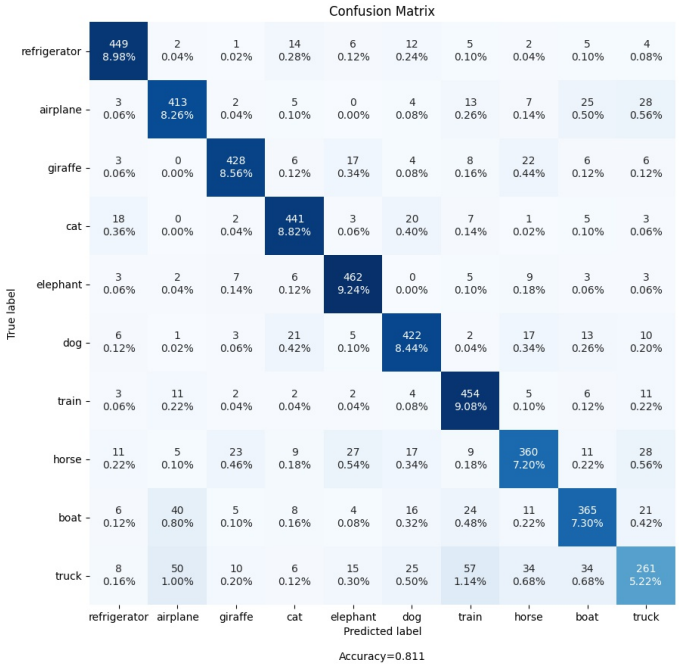

Figure 1: net1 confusion matrix

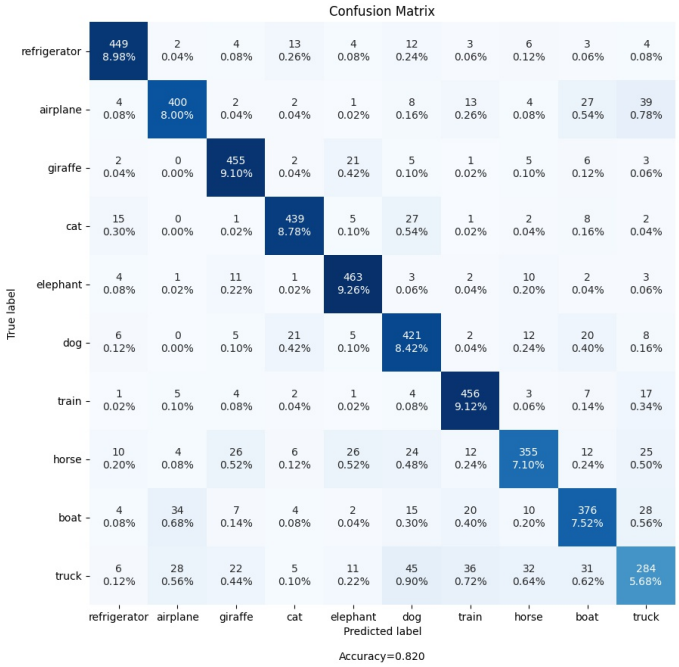

Figure 2: net2 confusion matrix

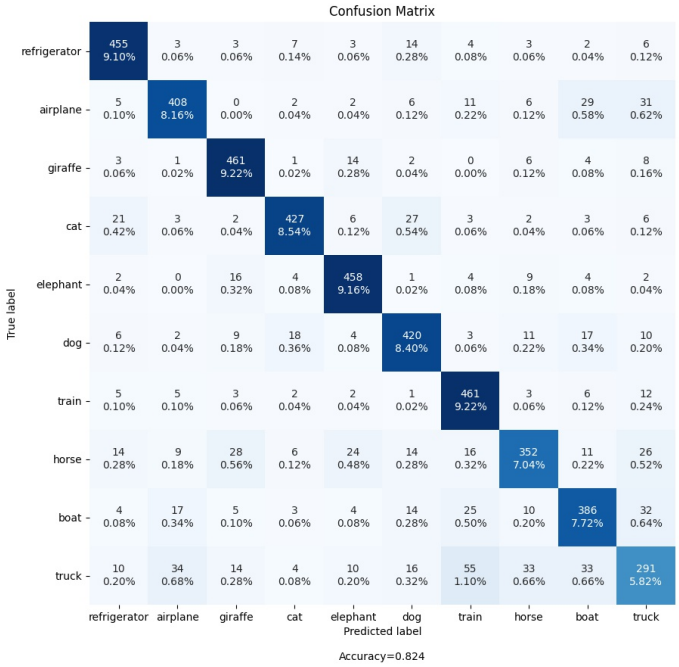

Figure 3: net3 confusion matrix

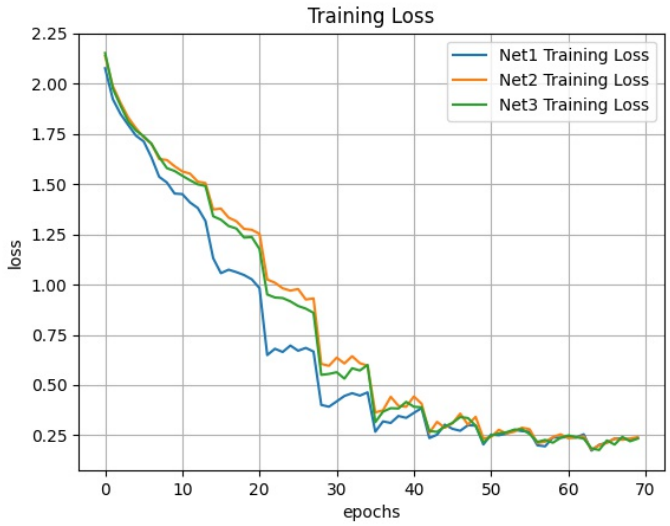

Figure 4: train loss