1.a.i)

Mean of channel R: 124.495

Mean of channel G: 118.847

Mean of channel B: 95.293

Standard Deviation of channel R: 62.754

Standard Deviation of channel G: 59.597

Standard Deviation of channel B: 62.425

1.a.ii)

By extracting the mean and standard deviation from the training set as opposed to other data partitions, we are not taking into account the structure of the data partitions that we use to evaluate our model. The validation and test performance helps us evaluate our model by making sure our model generalizes well to unseen data.

1.b)

Graphical user interface

Description automatically generated

2.a)

Conv1: 5 \* 5 \* 3 \* 16 \* 16 + 16 = 1,216

Pool: 0 + 0 = 0

Conv2: 5 \* 5 \* 16 \* 64 + 64 = 25,664

Pool: 0 + 0 = 0

Conv3: 5 \* 5 \* 64 \* 8 + 8 = 12,808

Fc\_1: 32 \* 2 + 2 = 66

There are 1,216 + 25,664 + 12,808 + 66 = 39,754 learnable float-valued parameters.

2.f.i)

We can observe that the validation loss doesn’t monotonically decrease in the training plot. This is because we randomly shuffle training data at the end of every epoch. Also, the noise in validation loss can be induced from data as it contains noise such as color balance.

2.f.ii)

Training plot generated using patience = 5:

Chart, line chart

Description automatically generated

The model stopped training at epoch = 10

Training plot generated using patience = 10:

Chart, scatter chart

Description automatically generated

The model stopped training at epoch = 15

Patience = 10 works better for this data set as it converges better for all three graphs. The training plot generated using patience = 5 looks like it is still decreasing so increasing the patience ensures that we have a model that has better predictions.

Increased patience might work better if we haven’t detected convergence to a specific value. Then by increasing patience, the objective function will be optimized.

2.f.iii)

New size of the input to the fully connected layer = 256

|  |  |  |  |
| --- | --- | --- | --- |
|  | Epoch | **Training AUROC** | **Validation AUROC** |
| 8 filters | 5 | 0.9793 | 0.93 |
| 64 filters | 2 | 0.9768 | 0.9239 |

Training plot for 8 filters:

**Chart, line chart

Description automatically generated**

Training plot for 64 filters:

Chart, line chart

Description automatically generated

As we increased the number of filters from 8 to 64, there is a larger gap between train and validation performance. The curves are smoother, and it was trained for fewer epochs.

Because the model has more parameters, it is more likely to overfit to the training data which resulted in a larger gap. Also, due to having more parameters, we are converging quicker so the curves appear smoother.

2.g.i)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Training** | **Validation** | **Testing** |
| Accuracy | 0.9067 | 0.8667 | 0.6 |
| AUROC | 0.9793 | 0.93 | 0.6556 |

2.g.ii)

The training and validation performance only differs by .04 which means that there is no evidence of overfitting.

2.g.iii)

The testing performance is much lower than the validation performance. This indicates that our validation data was not representative of the testing data. A possible explanation for such a trend could be validation data containing features that are not included in the test data such as validation data containing all the white-colored dogs while training data doesn’t contain any.

3.b)

A picture containing table

Description automatically generated

3.b)

CNN appears to be using background features and using the features of the dog itself to identify the Collie class.

3.c)

The Grad-Cam visualizations confirm our hypothesis that the training and validation set fails to be representative of the test set. Because our model uses background features to identify the Collie class, we can infer that in the training and validation set, backgrounds are critical and helps our model classify the dogs. Therefore, our test performance was lower as classifying dogs based on backgrounds is not generalizable to other scenarios.

4.1.c)

Chart, scatter chart

Description automatically generated

Epoch = 13 resulted in lowest validation loss of 1.7923

4.1.d)

Graphical user interface, diagram, text

Description automatically generated

The classifier was most accurate for Yorkshire Terrier, Dalmatian, and Samoyed while it was least accurate for Chihuahua. This might be the case as our model classified most images as either a Yorkshire Terrier or a Dalmatian. Dogs of small sizes such as a Miniature Poodle and Chihuahua were mostly classified as a Yorkshire Terrier while dogs of bigger sizes such as Siberian Husky were classified as a Dalmatian. The classifier was least accurate for Chihuahua as it has similar size as a Yorkshire Terrier while not having a distinct color as a Dalmatian.

4.1.f)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **AUROC** | | |
|  | **TRAIN** | **VAL** | **TEST** |
| Freeze all CONV layers | 0.9783 | 0.9145 | 0.8116 |
| Freeze first two CONV layers | 0.9822 | 0.9116 | 0.7964 |
| Freeze first CONV layer | 0.9898 | 0.9259 | 0.7636 |
| Freeze no layer | 0.9789 | 0.9241 | 0.8208 |
| No Pretraining or Transfer Learning (Section 2) | 0.9793 | 0.93 | 0.6556 |

Plot for Freeze no layers (Fine-tune all layers):

Graphical user interface, chart

Description automatically generated

Plot for Freeze first CONV layer (Fine-tune last 2 conv. and fc layers):

Chart

Description automatically generated

Plot for Freeze first two CONV layers (Fine-tune last CONV and FC layers):

Graphical user interface, chart

Description automatically generated

Plot for Freeze all CONV layers (Fine-tune FC layer): Graphical user interface, chart

Description automatically generated

Looking at the test performance, we can see that the test AUROC isn’t significantly smaller than the train and validation AUROC. Therefore, we can infer that the source task was helpful as train and validation data were representative of the test data. The transfer learning resulted in better performance than the Section 2 performance, even with freezing none. Freezing all convolutional layers resulted in a decrease of test AUROC compared to freezing just a subset or freezing none. This is because the model isn’t able to learn the bias.

4.2.iii)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **AUROC** | | |
|  | **TRAIN** | **VAL** | **TEST** |
| Rotation (keep original) | 0.9913 | 0.9323 | 0.6608 |
| Grayscale (keep original) | 0.9761 | 0.9193 | 0.7248 |
| Grayscale (discard original) | 0.8829 | 0.7913 | 0.7752 |
| No augmentation (Section 2 performance) | 0.9793 | 0.93 | 0.6556 |

Training Plot for Rotation (keep original):

**Chart

Description automatically generated**

Training Plot for Grayscale (keep original):

**Chart, line chart

Description automatically generated**

Training Plot for Grayscale (discard original):

**Chart, line chart

Description automatically generated**

4.2.c)

Rotation and Grayscale (keep original) training plots are similar to that of the Section 2. However, Grayscale (discard original) training plots display much worse performance than that of Section 2. This is because the Grayscale (discard original) hinders the model from classifying dogs based off on irrelevant features such as background while Rotation and Grayscale (keep original) doesn’t.

5)

I have decided to keep the model architecture and use transfer learning and data augmentation for this part. Even when I tried to increase the number of convolutional layers, the performance wasn’t any different from what we had implemented using Appendix B and C. Therefore, I assumed that the model architecture was complex enough to classify Collies and Golden Retrievers.

From (4.1.f), I discovered that Transfer learning gave a much better performance result even when no layers were frozen compared to the performance when no transfer learning was conducted. Although the chart from (4.1.f) shows that freezing all three layers yielded highest test performance, I got the same performance result for freezing one, two, and all layers. Therefore, I froze all layers because as we freeze more layers, the model will not be able to learn any biases in the dataset.

For the data augmentation, from (4.2.iii) we learned that if the training and the validation set is not representative of the test data, data augmentation helps our model generalize to the test dataset. Therefore, I used grayscale and discarded the original image for the data augmentation part to ensure that our model generalizes to the test dataset and don’t classify dogs based on irrelevant features such as the background of the image.

Because the spec mentions that we will be evaluated on the AUROC of my classifier’s predictions, I used AUROC to determine which model is best.

The model was trained on MacBook Pro, M1, 2020.

Running train\_challenge.py, the epoch with the lowest validation loss yielded Test AUROC:0.8208. Running test\_cnn\_challenge.py yielded Test AUROC:0.7604.

"""

EECS 445 - Introduction to Machine Learning

Winter 2022 - Project 2

Target CNN

Constructs a pytorch model for a convolutional neural network

Usage: from model.target import target

"""

import torch

import torch.nn as nn

import torch.nn.functional as F

from math import sqrt

from utils import config

class Target(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

## TODO: define each layer

padding\_size = 2

self.conv1 = nn.Conv2d(3, 16, (5,5), stride = (2,2), padding = padding\_size)

self.pool = nn.MaxPool2d((2, 2), stride = (2,2))

self.conv2 = nn.Conv2d(16, 64, (5,5), stride = (2,2), padding = padding\_size)

self.conv3 = nn.Conv2d(64, 8, (5,5), stride = (2,2), padding = padding\_size)

self.fc\_1 = nn.Linear(32, 2)

# 2.f.iii

# self.conv3 = nn.Conv2d(64, 64, (5,5), stride = (2,2), padding = padding\_size)

# self.fc\_1 = nn.Linear(256, 2)

##

self.init\_weights()

def init\_weights(self):

torch.manual\_seed(42)

for conv in [self.conv1, self.conv2, self.conv3]:

C\_in = conv.weight.size(1)

nn.init.normal\_(conv.weight, 0.0, 1 / sqrt(5 \* 5 \* C\_in))

nn.init.constant\_(conv.bias, 0.0)

## TODO: initialize the parameters for [self.fc\_1]

nn.init.normal\_(self.fc\_1.weight, 0.0, 1/32)

nn.init.constant\_(self.fc\_1.bias, 0.0)

##

def forward(self, x):

""" You may optionally use the x.shape variables below to resize/view the size of

the input matrix at different points of the forward pass

"""

N, C, H, W = x.shape

## TODO: forward pass

x = self.pool(F.relu(self.conv1(x)))

x = self.pool(F.relu(self.conv2(x)))

x = F.relu(self.conv3(x))

x = self.fc\_1(torch.flatten(x, 1))

##

return x

"""

EECS 445 - Introduction to Machine Learning

Winter 2022 - Project 2

Source CNN

Constructs a pytorch model for a convolutional neural network

Usage: from model.source import Source

"""

import torch

import torch.nn as nn

import torch.nn.functional as F

from math import sqrt

from utils import config

class Source(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

## TODO: define each layer

padding\_size = 2

self.conv1 = nn.Conv2d(3, 16, (5,5), stride = (2,2), padding = padding\_size)

self.pool = nn.MaxPool2d((2, 2), stride = (2,2))

self.conv2 = nn.Conv2d(16, 64, (5,5), stride = (2,2), padding = padding\_size)

self.conv3 = nn.Conv2d(64, 8, (5,5), stride = (2,2), padding = padding\_size)

self.fc1 = nn.Linear(32, 8)

##

self.init\_weights()

def init\_weights(self):

torch.manual\_seed(42)

for conv in [self.conv1, self.conv2, self.conv3]:

C\_in = conv.weight.size(1)

nn.init.normal\_(conv.weight, 0.0, 1 / sqrt(5 \* 5 \* C\_in))

nn.init.constant\_(conv.bias, 0.0)

## TODO: initialize the parameters for [self.fc1]

nn.init.normal\_(self.fc1.weight, 0.0, 1/32)

nn.init.constant\_(self.fc1.bias, 0.0)

##

def forward(self, x):

""" You may optionally use the x.shape variables below to resize/view the size of

the input matrix at different points of the forward pass

"""

N, C, H, W = x.shape

## TODO: forward pass

x = self.pool(F.relu(self.conv1(x)))

x = self.pool(F.relu(self.conv2(x)))

x = F.relu(self.conv3(x))

x = self.fc1(torch.flatten(x, 1))

##

return x

"""

EECS 445 - Introduction to Machine Learning

Winter 2022 - Project 2

Challenge

Constructs a pytorch model for a convolutional neural network

Usage: from model.challenge import Challenge

"""

import torch

import torch.nn as nn

import torch.nn.functional as F

from math import sqrt

from utils import config

class Challenge(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

## TODO: define each layer of your network

padding\_size = 2

self.conv1 = nn.Conv2d(3, 16, (5,5), stride = (2,2), padding = padding\_size)

self.pool = nn.MaxPool2d((2, 2), stride = (2,2))

self.conv2 = nn.Conv2d(16, 64, (5,5), stride = (2,2), padding = padding\_size)

self.conv3 = nn.Conv2d(64, 8, (5,5), stride = (2,2), padding = padding\_size)

self.fc\_1 = nn.Linear(32, 2)

##

self.init\_weights()

def init\_weights(self):

## TODO: initialize the parameters for your network

torch.manual\_seed(42)

for conv in [self.conv1, self.conv2, self.conv3]:

C\_in = conv.weight.size(1)

nn.init.normal\_(conv.weight, 0.0, 1 / sqrt(5 \* 5 \* C\_in))

nn.init.constant\_(conv.bias, 0.0)

nn.init.normal\_(self.fc\_1.weight, 0.0, 1/32)

nn.init.constant\_(self.fc\_1.bias, 0.0)

##

def forward(self, x):

""" You may optionally use the x.shape variables below to resize/view the size of

the input matrix at different points of the forward pass

"""

N, C, H, W = x.shape

## TODO: forward pass

x = self.pool(F.relu(self.conv1(x)))

x = self.pool(F.relu(self.conv2(x)))

x = F.relu(self.conv3(x))

x = self.fc\_1(torch.flatten(x, 1))

##

return x

"""

EECS 445 - Introduction to Machine Learning

Winter 2022 - Project 2

Source CNN Challenge

Constructs a pytorch model for a convolutional neural network

Usage: from model.source import Source

"""

import torch

import torch.nn as nn

import torch.nn.functional as F

from math import sqrt

from utils import config

class Source(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

## TODO: define each layer

padding\_size = 2

self.conv1 = nn.Conv2d(3, 16, (5,5), stride = (2,2), padding = padding\_size)

self.pool = nn.MaxPool2d((2, 2), stride = (2,2))

self.conv2 = nn.Conv2d(16, 64, (5,5), stride = (2,2), padding = padding\_size)

self.conv3 = nn.Conv2d(64, 8, (5,5), stride = (2,2), padding = padding\_size)

self.fc1 = nn.Linear(32, 8)

##

self.init\_weights()

def init\_weights(self):

torch.manual\_seed(42)

for conv in [self.conv1, self.conv2, self.conv3]:

C\_in = conv.weight.size(1)

nn.init.normal\_(conv.weight, 0.0, 1 / sqrt(5 \* 5 \* C\_in))

nn.init.constant\_(conv.bias, 0.0)

## TODO: initialize the parameters for [self.fc1]

nn.init.normal\_(self.fc1.weight, 0.0, 1/32)

nn.init.constant\_(self.fc1.bias, 0.0)

##

def forward(self, x):

""" You may optionally use the x.shape variables below to resize/view the size of

the input matrix at different points of the forward pass

"""

N, C, H, W = x.shape

## TODO: forward pass

x = self.pool(F.relu(self.conv1(x)))

x = self.pool(F.relu(self.conv2(x)))

x = F.relu(self.conv3(x))

x = self.fc1(torch.flatten(x, 1))

##

return x

"""

EECS 445 - Introduction to Machine Learning

Winter 2022 - Project 2

Script to create an augmented dataset.

"""

import argparse

import csv

import glob

import os

import sys

import numpy as np

from scipy.ndimage import rotate

from imageio import imread, imwrite

def Rotate(deg=20):

"""Return function to rotate image."""

def \_rotate(img):

"""Rotate a random amount in the range (-deg, deg).

Keep the dimensions the same and fill any missing pixels with black.

:img: H x W x C numpy array

:returns: H x W x C numpy array

"""

# TODO

degree = np.random.randint(-deg, deg)

return rotate(img, degree, reshape = False)

return \_rotate

def Grayscale():

"""Return function to grayscale image."""

def \_grayscale(img):

"""Return 3-channel grayscale of image.

Compute grayscale values by taking average across the three channels.

Round to the nearest integer.

:img: H x W x C numpy array

:returns: H x W x C numpy array

"""

# TODO

# gray = np.zeros((img.shape[0], img.shape[1]))

# for i in range(img.shape[0]):

# for j in range(img.shape[1]):

# R = img[i][j][0]

# G = img[i][j][1]

# B = img[i][j][2]

# mean = np.mean(R,G,B)

# gray[i][j] = mean

gray = img.mean(axis = 2)

# G = np.stack((R, G, B), axis = 2).astype(np.uint8)

return np.stack((gray, gray, gray), axis = 2).astype(np.uint8)

return \_grayscale

def augment(filename, transforms, n=1, original=True):

"""Augment image at filename.

:filename: name of image to be augmented

:transforms: List of image transformations

:n: number of augmented images to save

:returns: a list of augmented images, where the first image is the original

"""

print(f"Augmenting {filename}")

img = imread(filename)

res = [img] if original else []

for i in range(n):

new = img

for transform in transforms:

new = transform(new)

res.append(new)

return res

def main(args):

"""Create augmented dataset."""

reader = csv.DictReader(open(args.input, "r"), delimiter=",")

writer = csv.DictWriter(

open(f"{args.datadir}/augmented\_dogs.csv", "w"),

fieldnames=["filename", "semantic\_label", "partition", "numeric\_label", "task"],

)

augment\_partitions = set(args.partitions)

# TODO: change `augmentations` to specify which augmentations to apply

# augmentations = [Grayscale(), Rotate()]

# augmentations = [Rotate()]

augmentations = [Grayscale()]

writer.writeheader()

os.makedirs(f"{args.datadir}/augmented/", exist\_ok=True)

for f in glob.glob(f"{args.datadir}/augmented/\*"):

print(f"Deleting {f}")

os.remove(f)

for row in reader:

if row["partition"] not in augment\_partitions:

imwrite(

f"{args.datadir}/augmented/{row['filename']}",

imread(f"{args.datadir}/images/{row['filename']}"),

)

writer.writerow(row)

continue

imgs = augment(

f"{args.datadir}/images/{row['filename']}",

augmentations,

n=1,

original=False, # TODO: change to False to exclude original image.

)

for i, img in enumerate(imgs):

fname = f"{row['filename'][:-4]}\_aug\_{i}.png"

imwrite(f"{args.datadir}/augmented/{fname}", img)

writer.writerow(

{

"filename": fname,

"semantic\_label": row["semantic\_label"],

"partition": row["partition"],

"numeric\_label": row["numeric\_label"],

"task": row["task"],

}

)

if \_\_name\_\_ == "\_\_main\_\_":

parser = argparse.ArgumentParser()

parser.add\_argument("input", help="Path to input CSV file")

parser.add\_argument("datadir", help="Data directory", default="./data/")

parser.add\_argument(

"-p",

"--partitions",

nargs="+",

help="Partitions (train|val|test|challenge|none)+ to apply augmentations to. Defaults to train",

default=["train"],

)

main(parser.parse\_args(sys.argv[1:]))

"""

EECS 445 - Introduction to Machine Learning

Winter 2022 - Project 2

Test CNN Challenge

Test our trained CNN from train\_cnn.py on the heldout test data.

Load the trained CNN model from a saved checkpoint and evaulates using

accuracy and AUROC metrics.

Usage: python test\_cnn.py

"""

import torch

import numpy as np

import random

from dataset import get\_train\_val\_test\_loaders

from model.challenge import Challenge

from train\_common import \*

from utils import config

import utils

torch.manual\_seed(42)

np.random.seed(42)

random.seed(42)

def main():

"""Print performance metrics for model at specified epoch."""

# Data loaders

tr\_loader, va\_loader, te\_loader, \_ = get\_train\_val\_test\_loaders(

task="target",

batch\_size=config("target.batch\_size"),

)

# Model

model = Challenge()

# define loss function

criterion = torch.nn.CrossEntropyLoss()

# Attempts to restore the latest checkpoint if exists

print("Loading cnn...")

model, start\_epoch, stats = restore\_checkpoint(model, config("target.checkpoint"))

axes = utils.make\_training\_plot()

# Evaluate the model

evaluate\_epoch(

axes,

tr\_loader,

va\_loader,

te\_loader,

model,

criterion,

start\_epoch,

stats,

include\_test=True,

update\_plot=False,

)

if \_\_name\_\_ == "\_\_main\_\_":

main()

"""

EECS 445 - Introduction to Machine Learning

Winter 2022 - Project 2

Test CNN

Test our trained CNN from train\_cnn.py on the heldout test data.

Load the trained CNN model from a saved checkpoint and evaulates using

accuracy and AUROC metrics.

Usage: python test\_cnn.py

"""

import torch

import numpy as np

import random

from dataset import get\_train\_val\_test\_loaders

from model.target import Target

from train\_common import \*

from utils import config

import utils

torch.manual\_seed(42)

np.random.seed(42)

random.seed(42)

def main():

"""Print performance metrics for model at specified epoch."""

# Data loaders

tr\_loader, va\_loader, te\_loader, \_ = get\_train\_val\_test\_loaders(

task="target",

batch\_size=config("target.batch\_size"),

)

# Model

model = Target()

# define loss function

criterion = torch.nn.CrossEntropyLoss()

# Attempts to restore the latest checkpoint if exists

print("Loading cnn...")

model, start\_epoch, stats = restore\_checkpoint(model, config("target.checkpoint"))

axes = utils.make\_training\_plot()

# Evaluate the model

evaluate\_epoch(

axes,

tr\_loader,

va\_loader,

te\_loader,

model,

criterion,

start\_epoch,

stats,

include\_test=True,

update\_plot=False,

)

if \_\_name\_\_ == "\_\_main\_\_":

main()

"""

EECS 445 - Introduction to Machine Learning

Winter 2022 - Project 2

Train Challenge

Train a convolutional neural network to classify the heldout images

Periodically output training information, and saves model checkpoints

Usage: python train\_challenge.py

"""

import torch

import numpy as np

import random

from dataset import get\_train\_val\_test\_loaders

from model.challenge import Challenge

from train\_common import \*

from train\_target import train

from utils import config

import utils

import copy

def freeze\_layers(model, num\_layers=0):

"""Stop tracking gradients on selected layers."""

#TODO: modify model with the given layers frozen

# e.g. if num\_layers=2, freeze CONV1 and CONV2

# Hint: https://pytorch.org/docs/master/notes/autograd.html

layer = num\_layers \* 2

for name, param in model.named\_parameters():

if layer != 0:

param.requires\_grad\_=False

layer -= 1

else:

break

def main():

"""Train transfer learning model and display training plots.

Train four different models with {0, 1, 2, 3} layers frozen.

"""

# data loaders

tr\_loader, va\_loader, te\_loader, \_ = get\_train\_val\_test\_loaders(

task="target",

batch\_size=config("target.batch\_size"),

)

freeze\_none = Challenge()

print("Loading source...")

freeze\_none, \_, \_ = restore\_checkpoint(

freeze\_none, config("source.checkpoint"), force=True, pretrain=True

)

freeze\_three = copy.deepcopy(freeze\_none)

freeze\_layers(freeze\_three, 3)

train(tr\_loader, va\_loader, te\_loader, freeze\_three, "./checkpoints/target3/", 3)

# # Data loaders

# if check\_for\_augmented\_data("./data"):

# tr\_loader, va\_loader, te\_loader, \_ = get\_train\_val\_test\_loaders(

# task="target",

# batch\_size=config("challenge.batch\_size"), augment = True

# )

# else:

# tr\_loader, va\_loader, te\_loader, \_ = get\_train\_val\_test\_loaders(

# task="target",

# batch\_size=config("challenge.batch\_size"),

# )

# # Model

# model = Challenge()

# # TODO: define loss function, and optimizer

# criterion = torch.nn.CrossEntropyLoss()

# optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)

# #

# # Attempts to restore the latest checkpoint if exists

# print("Loading challenge...")

# model, start\_epoch, stats = restore\_checkpoint(model, config("challenge.checkpoint"))

# axes = utils.make\_training\_plot()

# # Transfer learning freeze layer

# freeze\_none = Challenge()

# freeze\_none, \_, \_ = restore\_checkpoint(

# freeze\_none, config("source.checkpoint"), force=True, pretrain=True

# )

# freeze\_one = copy.deepcopy(freeze\_none)

# freeze\_layers(freeze\_one, 1)

# train(tr\_loader, va\_loader, te\_loader, freeze\_none, "./checkpoints/target0/", 0)

# # Evaluate the randomly initialized model

# evaluate\_epoch(

# axes, tr\_loader, va\_loader, te\_loader, model, criterion, start\_epoch, stats

# )

# # initial val loss for early stopping

# global\_min\_loss = stats[0][1]

# #TODO: define patience for early stopping

# patience = 10

# curr\_count\_to\_patience = 0

# #

# # Loop over the entire dataset multiple times

# epoch = start\_epoch

# while curr\_count\_to\_patience < patience:

# # Train model

# train\_epoch(tr\_loader, model, criterion, optimizer)

# # Evaluate model

# evaluate\_epoch(

# axes, tr\_loader, va\_loader, te\_loader, model, criterion, epoch + 1, stats

# )

# # Save model parameters

# save\_checkpoint(model, epoch + 1, config("challenge.checkpoint"), stats)

# # Updates early stopping parameters

# curr\_count\_to\_patience, global\_min\_loss = early\_stopping(

# stats, curr\_count\_to\_patience, global\_min\_loss

# )

# #

# epoch += 1

# print("Finished Training")

# # Save figure and keep plot open

# utils.save\_challenge\_training\_plot()

# utils.hold\_training\_plot()

if \_\_name\_\_ == "\_\_main\_\_":

main()

"""

EECS 445 - Introduction to Machine Learning

Winter 2022 - Project 2

Train CNN

Train a convolutional neural network to classify images

Periodically output training information, and saves model checkpoints

Usage: python train\_cnn.py

"""

import torch

import numpy as np

import random

from dataset import get\_train\_val\_test\_loaders

from model.target import Target

from train\_common import \*

from utils import config

import utils

torch.manual\_seed(42)

np.random.seed(42)

random.seed(42)

def main():

"""Train CNN and show training plots."""

# Data loaders

if check\_for\_augmented\_data("./data"):

tr\_loader, va\_loader, te\_loader, \_ = get\_train\_val\_test\_loaders(

task="target", batch\_size=config("target.batch\_size"), augment=True

)

else:

tr\_loader, va\_loader, te\_loader, \_ = get\_train\_val\_test\_loaders(

task="target",

batch\_size=config("target.batch\_size"),

)

# Model

model = Target()

# TODO: define loss function, and optimizer

criterion = torch.nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)

#

print("Number of float-valued parameters:", count\_parameters(model))

# Attempts to restore the latest checkpoint if exists

print("Loading cnn...")

model, start\_epoch, stats = restore\_checkpoint(model, config("target.checkpoint"))

axes = utils.make\_training\_plot()

# Evaluate the randomly initialized model

evaluate\_epoch(

axes, tr\_loader, va\_loader, te\_loader, model, criterion, start\_epoch, stats

)

# initial val loss for early stopping

global\_min\_loss = stats[0][1]

# TODO: define patience for early stopping

patience = 5

curr\_count\_to\_patience = 0

#

# Loop over the entire dataset multiple times

epoch = start\_epoch

while curr\_count\_to\_patience < patience:

# Train model

train\_epoch(tr\_loader, model, criterion, optimizer)

# Evaluate model

evaluate\_epoch(

axes, tr\_loader, va\_loader, te\_loader, model, criterion, epoch + 1, stats

)

# Save model parameters

save\_checkpoint(model, epoch + 1, config("target.checkpoint"), stats)

# update early stopping parameters

curr\_count\_to\_patience, global\_min\_loss = early\_stopping(

stats, curr\_count\_to\_patience, global\_min\_loss

)

epoch += 1

print("Finished Training")

# Save figure and keep plot open

utils.save\_cnn\_training\_plot()

utils.hold\_training\_plot()

if \_\_name\_\_ == "\_\_main\_\_":

main()

"""

EECS 445 - Introduction to Machine Learning

Winter 2022 - Project 2

Helper file for common training functions.

"""

from utils import config

import numpy as np

import itertools

import os

import torch

from torch.nn.functional import softmax

from sklearn import metrics

import utils

def count\_parameters(model):

"""Count number of learnable parameters."""

return sum(p.numel() for p in model.parameters() if p.requires\_grad)

def save\_checkpoint(model, epoch, checkpoint\_dir, stats):

"""Save a checkpoint file to `checkpoint\_dir`."""

state = {

"epoch": epoch,

"state\_dict": model.state\_dict(),

"stats": stats,

}

filename = os.path.join(checkpoint\_dir, "epoch={}.checkpoint.pth.tar".format(epoch))

torch.save(state, filename)

def check\_for\_augmented\_data(data\_dir):

"""Ask to use augmented data if `augmented\_dogs.csv` exists in the data directory."""

if "augmented\_dogs.csv" in os.listdir(data\_dir):

print("Augmented data found, would you like to use it? y/n")

print(">> ", end="")

rep = str(input())

return rep == "y"

return False

def restore\_checkpoint(model, checkpoint\_dir, cuda=False, force=False, pretrain=False):

"""Restore model from checkpoint if it exists.

Returns the model and the current epoch.

"""

try:

cp\_files = [

file\_

for file\_ in os.listdir(checkpoint\_dir)

if file\_.startswith("epoch=") and file\_.endswith(".checkpoint.pth.tar")

]

except FileNotFoundError:

cp\_files = None

os.makedirs(checkpoint\_dir)

if not cp\_files:

print("No saved model parameters found")

if force:

raise Exception("Checkpoint not found")

else:

return model, 0, []

# Find latest epoch

for i in itertools.count(1):

if "epoch={}.checkpoint.pth.tar".format(i) in cp\_files:

epoch = i

else:

break

if not force:

print(

"Which epoch to load from? Choose in range [0, {}].".format(epoch),

"Enter 0 to train from scratch.",

)

print(">> ", end="")

inp\_epoch = int(input())

if inp\_epoch not in range(epoch + 1):

raise Exception("Invalid epoch number")

if inp\_epoch == 0:

print("Checkpoint not loaded")

clear\_checkpoint(checkpoint\_dir)

return model, 0, []

else:

print("Which epoch to load from? Choose in range [1, {}].".format(epoch))

inp\_epoch = int(input())

if inp\_epoch not in range(1, epoch + 1):

raise Exception("Invalid epoch number")

filename = os.path.join(

checkpoint\_dir, "epoch={}.checkpoint.pth.tar".format(inp\_epoch)

)

print("Loading from checkpoint {}?".format(filename))

if cuda:

checkpoint = torch.load(filename)

else:

# Load GPU model on CPU

checkpoint = torch.load(filename, map\_location=lambda storage, loc: storage)

try:

start\_epoch = checkpoint["epoch"]

stats = checkpoint["stats"]

if pretrain:

model.load\_state\_dict(checkpoint["state\_dict"], strict=False)

else:

model.load\_state\_dict(checkpoint["state\_dict"])

print(

"=> Successfully restored checkpoint (trained for {} epochs)".format(

checkpoint["epoch"]

)

)

except:

print("=> Checkpoint not successfully restored")

raise

return model, inp\_epoch, stats

def clear\_checkpoint(checkpoint\_dir):

"""Remove checkpoints in `checkpoint\_dir`."""

filelist = [f for f in os.listdir(checkpoint\_dir) if f.endswith(".pth.tar")]

for f in filelist:

os.remove(os.path.join(checkpoint\_dir, f))

print("Checkpoint successfully removed")

def early\_stopping(stats, curr\_count\_to\_patience, global\_min\_loss):

"""Calculate new patience and validation loss.

Increment curr\_count\_to\_patience by one if new loss is not less than global\_min\_loss

Otherwise, update global\_min\_loss with the current val loss

Returns: new values of curr\_count\_to\_patience and global\_min\_loss

"""

# TODO implement early stopping

#

if stats[-1][1] >= global\_min\_loss:

curr\_count\_to\_patience += 1

else:

global\_min\_loss = stats[-1][1]

curr\_count\_to\_patience = 0

return curr\_count\_to\_patience, global\_min\_loss

def evaluate\_epoch(

axes,

tr\_loader,

val\_loader,

te\_loader,

model,

criterion,

epoch,

stats,

include\_test=False,

update\_plot=True,

multiclass=False,

):

"""Evaluate the `model` on the train and validation set."""

def \_get\_metrics(loader):

y\_true, y\_pred, y\_score = [], [], []

correct, total = 0, 0

running\_loss = []

for X, y in loader:

with torch.no\_grad():

output = model(X)

predicted = predictions(output.data)

y\_true.append(y)

y\_pred.append(predicted)

if not multiclass:

y\_score.append(softmax(output.data, dim=1)[:, 1])

else:

y\_score.append(softmax(output.data, dim=1))

total += y.size(0)

correct += (predicted == y).sum().item()

running\_loss.append(criterion(output, y).item())

y\_true = torch.cat(y\_true)

y\_pred = torch.cat(y\_pred)

y\_score = torch.cat(y\_score)

loss = np.mean(running\_loss)

acc = correct / total

if not multiclass:

auroc = metrics.roc\_auc\_score(y\_true, y\_score)

else:

auroc = metrics.roc\_auc\_score(y\_true, y\_score, multi\_class="ovo")

return acc, loss, auroc

train\_acc, train\_loss, train\_auc = \_get\_metrics(tr\_loader)

val\_acc, val\_loss, val\_auc = \_get\_metrics(val\_loader)

stats\_at\_epoch = [

val\_acc,

val\_loss,

val\_auc,

train\_acc,

train\_loss,

train\_auc,

]

if include\_test:

stats\_at\_epoch += list(\_get\_metrics(te\_loader))

stats.append(stats\_at\_epoch)

utils.log\_training(epoch, stats)

if update\_plot:

utils.update\_training\_plot(axes, epoch, stats)

def train\_epoch(data\_loader, model, criterion, optimizer):

"""Train the `model` for one epoch of data from `data\_loader`.

Use `optimizer` to optimize the specified `criterion`

"""

for i, (X, y) in enumerate(data\_loader):

# TODO implement training steps

predictions = model(X)

loss = criterion(predictions, y)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

def predictions(logits):

"""Determine predicted class index given logits.

Returns:

the predicted class output as a PyTorch Tensor

"""

# TODO implement predictions

max = np.argmax(np.array(logits), axis = 1)

max = torch.tensor(max)

return max

"""

EECS 445 - Introduction to Machine Learning

Winter 2022 - Project 2

Train Source CNN Challenge

Train a convolutional neural network to classify images.

Periodically output training information, and saves model checkpoints

Usage: python3 train\_source.py

"""

import torch

import numpy as np

import random

from dataset import get\_train\_val\_test\_loaders

from model.source import Source

from train\_common import \*

from utils import config

import utils

torch.manual\_seed(42)

np.random.seed(42)

random.seed(42)

def main():

"""Train source model on multiclass data."""

# Data loaders

tr\_loader, va\_loader, te\_loader, \_ = get\_train\_val\_test\_loaders(

task="source",

batch\_size=config("source.batch\_size"),

)

# Model

model = Source()

# TODO: define loss function, and optimizer

criterion = torch.nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(), lr = 0.001, weight\_decay = 0.01)

#

print("Number of float-valued parameters:", count\_parameters(model))

# Attempts to restore the latest checkpoint if exists

print("Loading source...")

model, start\_epoch, stats = restore\_checkpoint(model, config("source.checkpoint"))

axes = utils.make\_training\_plot("Source Training")

# Evaluate the randomly initialized model

evaluate\_epoch(

axes,

tr\_loader,

va\_loader,

te\_loader,

model,

criterion,

start\_epoch,

stats,

multiclass=True,

)

# initial val loss for early stopping

global\_min\_loss = stats[0][1]

# TODO: patience for early stopping

patience = 10

curr\_count\_to\_patience = 0

#

# Loop over the entire dataset multiple times

epoch = start\_epoch

while curr\_count\_to\_patience < patience:

# Train model

train\_epoch(tr\_loader, model, criterion, optimizer)

# Evaluate model

evaluate\_epoch(

axes,

tr\_loader,

va\_loader,

te\_loader,

model,

criterion,

epoch + 1,

stats,

multiclass=True,

)

# Save model parameters

save\_checkpoint(model, epoch + 1, config("source.checkpoint"), stats)

curr\_count\_to\_patience, global\_min\_loss = early\_stopping(

stats, curr\_count\_to\_patience, global\_min\_loss

)

epoch += 1

# Save figure and keep plot open

print("Finished Training")

utils.save\_source\_training\_plot()

utils.hold\_training\_plot()

if \_\_name\_\_ == "\_\_main\_\_":

main()

"""

EECS 445 - Introduction to Machine Learning

Winter 2022 - Project 2

Train Source CNN

Train a convolutional neural network to classify images.

Periodically output training information, and saves model checkpoints

Usage: python3 train\_source.py

"""

import torch

import numpy as np

import random

from dataset import get\_train\_val\_test\_loaders

from model.source import Source

from train\_common import \*

from utils import config

import utils

torch.manual\_seed(42)

np.random.seed(42)

random.seed(42)

def main():

"""Train source model on multiclass data."""

# Data loaders

tr\_loader, va\_loader, te\_loader, \_ = get\_train\_val\_test\_loaders(

task="source",

batch\_size=config("source.batch\_size"),

)

# Model

model = Source()

# TODO: define loss function, and optimizer

criterion = torch.nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(), lr = 0.001, weight\_decay = 0.01)

#

print("Number of float-valued parameters:", count\_parameters(model))

# Attempts to restore the latest checkpoint if exists

print("Loading source...")

model, start\_epoch, stats = restore\_checkpoint(model, config("source.checkpoint"))

axes = utils.make\_training\_plot("Source Training")

# Evaluate the randomly initialized model

evaluate\_epoch(

axes,

tr\_loader,

va\_loader,

te\_loader,

model,

criterion,

start\_epoch,

stats,

multiclass=True,

)

# initial val loss for early stopping

global\_min\_loss = stats[0][1]

# TODO: patience for early stopping

patience = 10

curr\_count\_to\_patience = 0

#

# Loop over the entire dataset multiple times

epoch = start\_epoch

while curr\_count\_to\_patience < patience:

# Train model

train\_epoch(tr\_loader, model, criterion, optimizer)

# Evaluate model

evaluate\_epoch(

axes,

tr\_loader,

va\_loader,

te\_loader,

model,

criterion,

epoch + 1,

stats,

multiclass=True,

)

# Save model parameters

save\_checkpoint(model, epoch + 1, config("source.checkpoint"), stats)

curr\_count\_to\_patience, global\_min\_loss = early\_stopping(

stats, curr\_count\_to\_patience, global\_min\_loss

)

epoch += 1

# Save figure and keep plot open

print("Finished Training")

utils.save\_source\_training\_plot()

utils.hold\_training\_plot()

if \_\_name\_\_ == "\_\_main\_\_":

main()

"""

EECS 445 - Introduction to Machine Learning

Winter 2022 - Project 2

Train Target

Train a convolutional neural network to classify images.

Periodically output training information, and saves model checkpoints

Usage: python train\_target.py

"""

from gc import freeze

import torch

import numpy as np

import random

from dataset import get\_train\_val\_test\_loaders

from model.target import Target

from train\_common import \*

from utils import config

import utils

import copy

torch.manual\_seed(42)

np.random.seed(42)

random.seed(42)

def freeze\_layers(model, num\_layers=0):

"""Stop tracking gradients on selected layers."""

#TODO: modify model with the given layers frozen

# e.g. if num\_layers=2, freeze CONV1 and CONV2

# Hint: https://pytorch.org/docs/master/notes/autograd.html

layer = num\_layers \* 2

for name, param in model.named\_parameters():

if layer != 0:

param.requires\_grad\_=False

layer -= 1

else:

break

def train(tr\_loader, va\_loader, te\_loader, model, model\_name, num\_layers=0):

"""Train transfer learning model."""

#TODO: define loss function, and optimizer

criterion = torch.nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)

#

print("Loading target model with", num\_layers, "layers frozen")

model, start\_epoch, stats = restore\_checkpoint(model, model\_name)

axes = utils.make\_training\_plot("Target Training")

evaluate\_epoch(

axes,

tr\_loader,

va\_loader,

te\_loader,

model,

criterion,

start\_epoch,

stats,

include\_test=True,

)

# initial val loss for early stopping

global\_min\_loss = stats[0][1]

#TODO: patience for early stopping

patience = 5

curr\_count\_to\_patience = 0

#

# Loop over the entire dataset multiple times

epoch = start\_epoch

while curr\_count\_to\_patience < patience:

# Train model

train\_epoch(tr\_loader, model, criterion, optimizer)

# Evaluate model

evaluate\_epoch(

axes,

tr\_loader,

va\_loader,

te\_loader,

model,

criterion,

epoch + 1,

stats,

include\_test=True,

)

# Save model parameters

save\_checkpoint(model, epoch + 1, model\_name, stats)

curr\_count\_to\_patience, global\_min\_loss = early\_stopping(

stats, curr\_count\_to\_patience, global\_min\_loss

)

epoch += 1

print("Finished Training")

# Keep plot open

utils.save\_tl\_training\_plot(num\_layers)

utils.hold\_training\_plot()

def main():

"""Train transfer learning model and display training plots.

Train four different models with {0, 1, 2, 3} layers frozen.

"""

# data loaders

tr\_loader, va\_loader, te\_loader, \_ = get\_train\_val\_test\_loaders(

task="target",

batch\_size=config("target.batch\_size"),

)

freeze\_none = Target()

print("Loading source...")

freeze\_none, \_, \_ = restore\_checkpoint(

freeze\_none, config("source.checkpoint"), force=True, pretrain=True

)

freeze\_one = copy.deepcopy(freeze\_none)

freeze\_two = copy.deepcopy(freeze\_none)

freeze\_three = copy.deepcopy(freeze\_none)

freeze\_layers(freeze\_one, 1)

freeze\_layers(freeze\_two, 2)

freeze\_layers(freeze\_three, 3)

print("zero")

train(tr\_loader, va\_loader, te\_loader, freeze\_none, "./checkpoints/target0/", 0)

print("one")

train(tr\_loader, va\_loader, te\_loader, freeze\_one, "./checkpoints/target1/", 1)

print("two")

train(tr\_loader, va\_loader, te\_loader, freeze\_two, "./checkpoints/target2/", 2)

print("three")

train(tr\_loader, va\_loader, te\_loader, freeze\_three, "./checkpoints/target3/", 3)

if \_\_name\_\_ == "\_\_main\_\_":

main()