Report

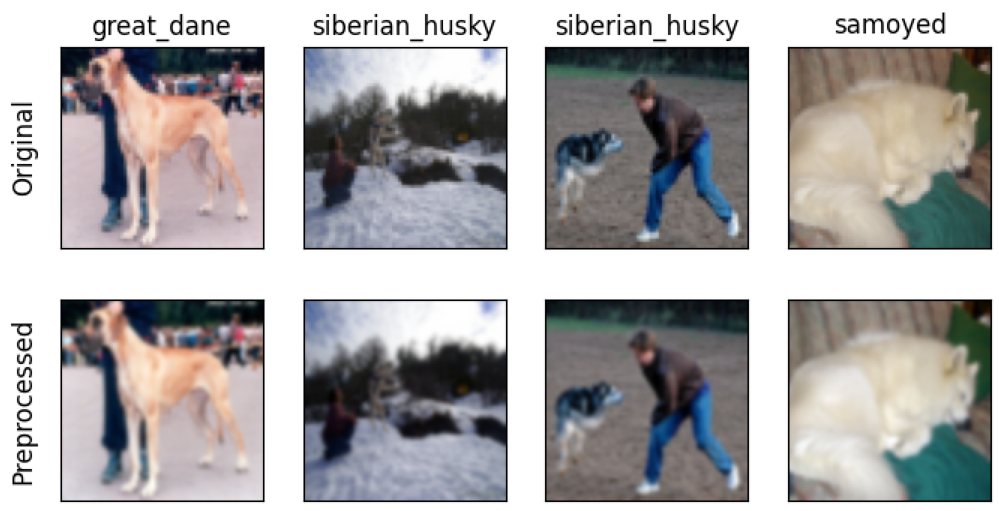
1 Data Preprocessing (See Appendix A for code)

(a)

i).
Mean: [124.495 118.847 95.293]
Std: [62.754 59.597 62.425]

ii).
Because we are training a model that is representative of the training dataset. This means that the model should be fit based on properties of the training data. We are simply applying the model to validation and testing data; so if we use their properties, it may cause overfitting. It is also te case that normalization puts everything on the same scale and this simplifies the learning process.

(b)



2 Convolutional Neural Network (See Appendix B)

(a)

For CNN, the number of learnable parameter (weights) can be calculated by (num_filters *filter_size* filter_size *num_channels) + num_filters*
*For fully connected layers, the number of weights is (input_size*output_size) + num_biases

So for layer 1, there are 16*553 + 16 = 1216 learnable parameters.*
*For layer 2, there are 645*516 + 64 = 25664 learnable parameters.
For layer 3, there are 8*5564 + 6 = 12808 learnable parameters.*
*For the fully connected layer, there are 32*2 + 2 = 66 learnable parameters.
In total, there are 39754 parameters.

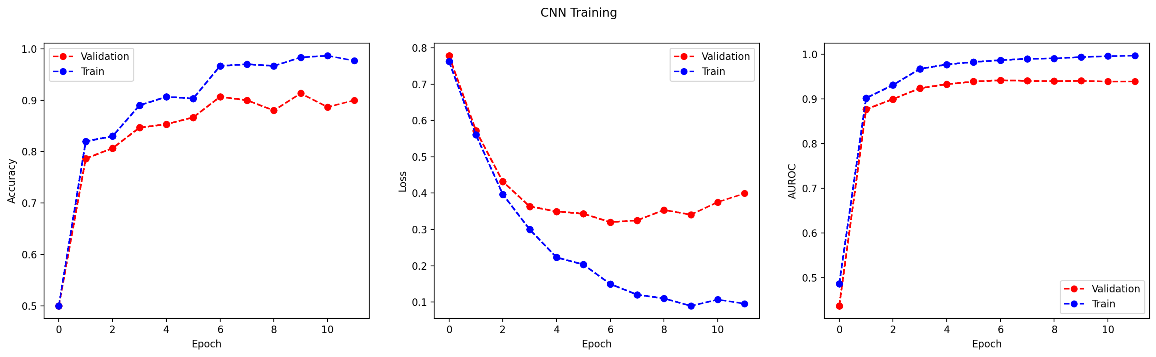
(f)

i).
One reason is that our the model is overfitting the training data so it won't perform well on the validation dataset. Another possible reason is that the the training, validation, and testing dataset aren't distributed similarly. So what the model learns from the training data can't be used to predict the validation data well. It is also possible that the amount of data we have isn't sufficient and this will lead to a model with poor performance.

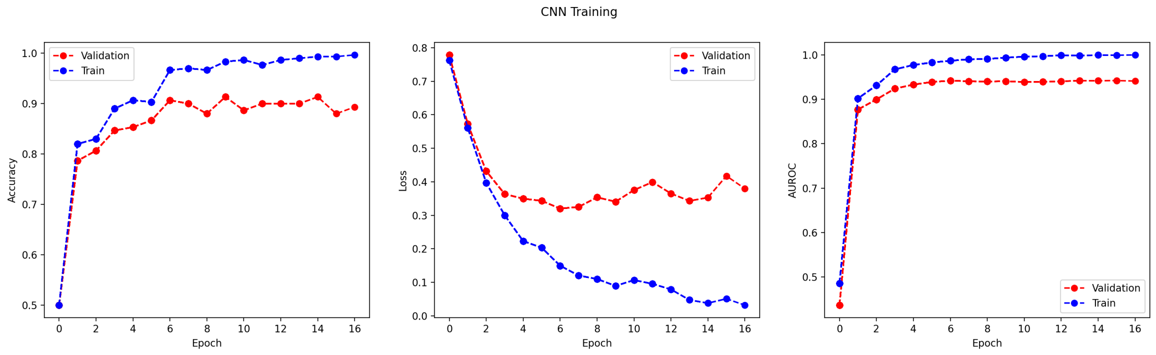
ii).
The model stopped training at epoch 11. With a patience of 10, the model would have to wait until epoch 16 to stop. Based on the training graphs, patience = 5 is a better choice because the graphs are all generally identical. However, patience = 5 takes fewer epoch to reach the performance. A

higher patience might be better for training data that has a local minimum; if the patience is high, we won't stop at the local minimum and could potentially get a better model.

When patience = 5



When patience = 10



iii).
The new size is 64x2x2 = 256.

	Epoch	Training AUROC	Validation AUROC
8 filters	6	0.9867	0.9419
64 filters	2	0.9777	0.923

The performance actually decreases as we increase the number of filters. This could be because that the architecture we set up makes the model overfits; we are having more neurons then we need to correctly classify an image. It could also be the case that the added inputs introduce unnecessary noise to the fully connected layer, which can distrub the overall accuracy of the model.

(g)

i).

	Training	Validation	Testing
Accuracy	0.9667	0.9067	0.61
AUROC	0.9867	0.9419	0.6548

ii).
Yes. We can see that the training accuracy is far better than the validation performance and it's similar for the AUROC score. So the model could potentially be overfitting.

iii).
We see that there the testing perforamnce is much wrose than the validation performance in both the accuracy and AUROC score. This could be because we are not splitting our data properly and the validation data has a lot more label belonging to golden retriever while the testing data has a lot more labels of Collies. Therefore, the model will perform poorly on the testing data.

3 Visualizing what the CNN has learned

(a)

$$\alpha_1 = \frac{1}{16}\sum_i\sum_j \frac{\partial dy}{\partial A'_{ij}} = 1+1+2+1+1+2+1+1-1+1-2-2 = \frac{3}{16}$$

$$\alpha_2 = \frac{1}{16}\sum_i\sum_j \frac{\partial dy}{\partial A'_{ij}} = 1+1+1+1+2+2+2+2+2+2+1-1-1-1 = \frac{7}{16}$$

$$L' = ReLU(\sum_k \alpha'_k A^{(k)}) = ReLU(\frac{3}{16}\begin{bmatrix}1 & 1 & 2 & 1\\ 1 & 2 & 1 & 0\\ 0 & 1 & 0 & -1\\ 0 & 1 & -2 & -2\end{bmatrix} + \frac{7}{16}\begin{bmatrix}1 & 1 & 1 & 1\\ 2 & 2 & 2 & 2\\ 2 & 2 & 1 & 0\\ -1 & -1 & -1 & 0\end{bmatrix}) = \frac{1}{16}\begin{bmatrix}10 & 10 & 13 & 10\\ 17 & 20 & 17 & 14\\ 14 & 17 & 7 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}$$

(b)

The CNN appears to be using green pixels to identify between Golden Retrievers and Collies.

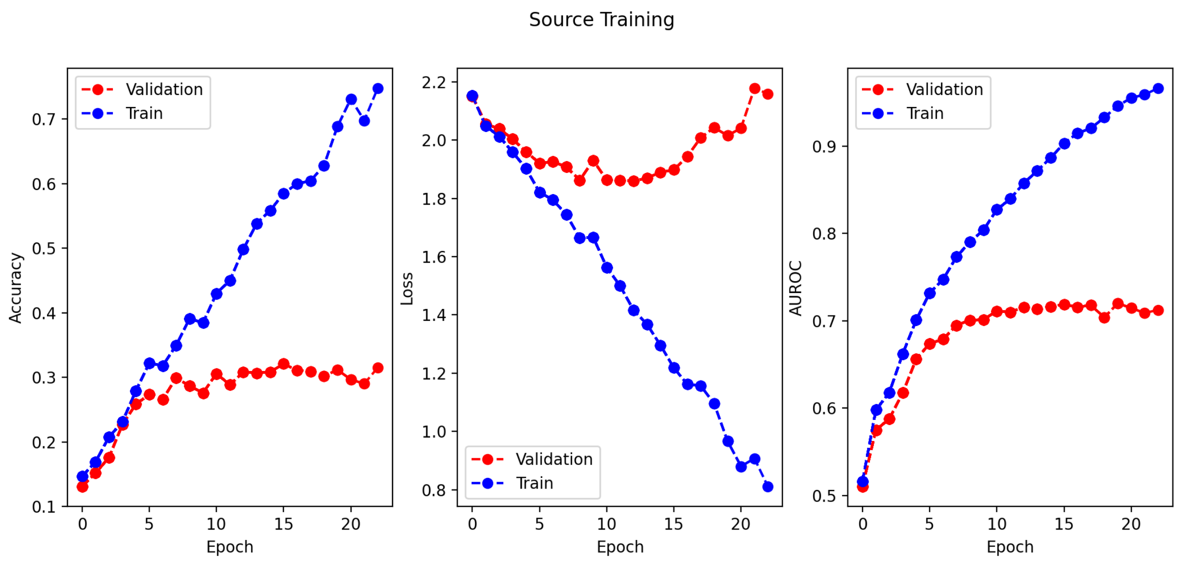
(C)

Yes, it confirms the hypothesis. It seems like the model relies heavily on green pixels to identify between Golden Retrievers and Collies. However, we as humans know that this is not the distinctive feature between the two types of dogs. This shows that the model has a strong bias toward green color, and it further implies that the green is over-represented in the training data and our model is biased and cannot perform as well on the testing data.

4 Transfer Learning & Data Augmentation

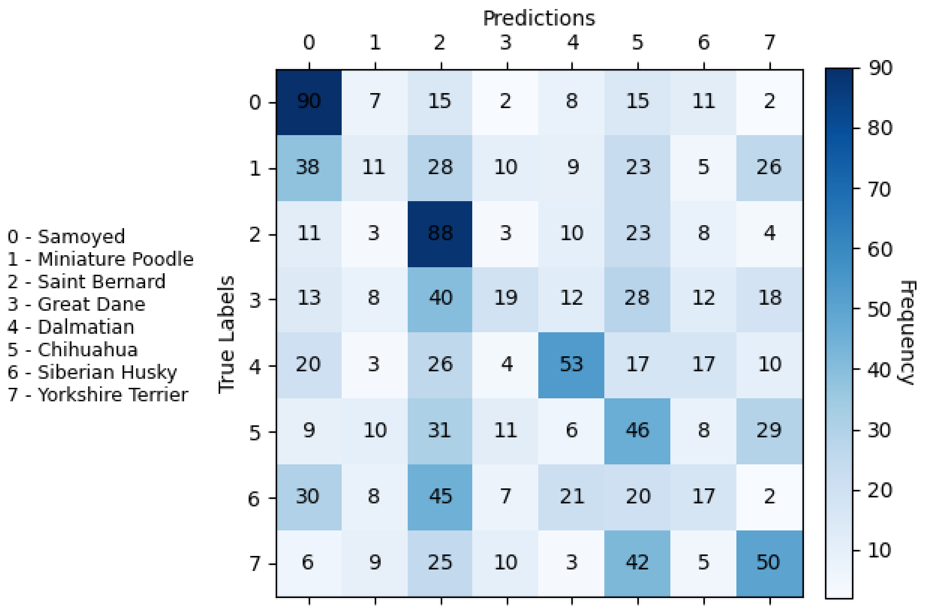
4.1 Transfer Learning (See Appendix C)

(c)



Epoch 10 has the lowest validation loss with a value of 1.8664.

(d)



	0	1	2	3	4	5	6	7
0	90	7	15	2	8	15	11	2
1	38	11	28	10	9	23	5	26
2	11	3	88	3	10	23	8	4
3	13	8	40	19	12	28	12	18
4	20	3	26	4	53	17	17	10
5	9	10	31	11	6	46	8	29
6	30	8	45	7	21	20	17	2
7	6	9	25	10	3	42	5	50

The classifier is the most accurate when predicting for Samoyed and it is the least accurate when predicting for Syberian Husky. This might be because we the training data we have is mislabeled and most Syberian Husky use Samoyed's label. This means that the classifier will treat most Syberian Husky as Samoyed and classify them correctly. Since Samoyed and Siberian Husky have very similar features, this strengthens the model's connection with Samoyed. On the other hand, the classifier doesn't have enough information to classify Syberian Husky because the train dataset doesn't contain many samples of it. This will result in very low accuracy for Syberian Husky.

(f)

	AUROC		
	TRAIN	VAL	TEST
Freeze all CONV layers (Fine_tune FC layer)	0.8822	0.8832	0.826
Freeze first two CONV layers (Fine-tune last CONV and FC layers)	0.9865	0.9102	0.7932
Freeze first CONV layer (Fine-tune last 2 conv, and fc layer)	0.9904	0.922	0.788
Freeze no layers (Fine-tune all layers)	0.9905	0.9262	0.7576
No pretraining or Transfer Learning (Section 2 performance)	0.9867	0.9419	0.6548

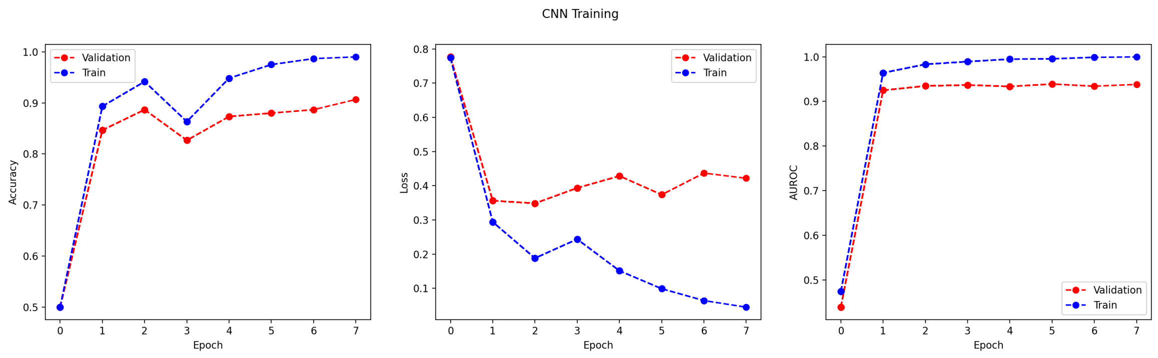
Transfer learning helps significantly and the source task we used is very helpful because we witness a huge imporvement in testing performance. Freezing all layers results in a lot more epoches than when we freeze a subset of the layers. This is because without convolutional layers, we are not filtering our data. So our fully connected layer will receive many many more inputs. This means that it will take a lot longer to train the classifier and many epoches are taken before we reach a good performance. Nevertheless, freezing all convolutional layer results in a much better performance; this is because we are preserving all learnable features in our dataset. There will always be information loss during the filtering stage. The negative consequence of having no convolutional layer is high computation cost.

4.2 Data Augmentation (See Appendix D and E)

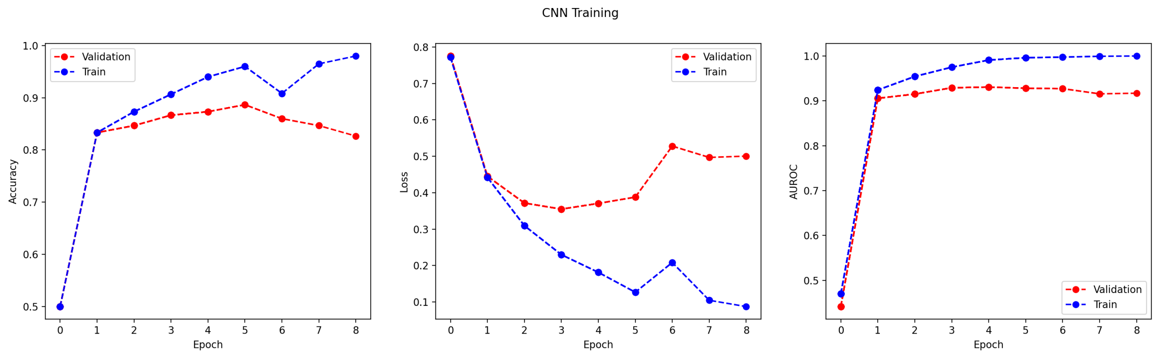
(b)

	AUROC		
	TRAIN	VAL	TEST
Rotation (Keep original)	0.9812	0.934	0.6584
Grayscale (keep original)	0.9915	0.9275	0.7404
Grayscale (discard original)	0.9672	0.8025	0.7776
No augmentation (section 2 performance)	0.9867	0.9419	0.6548

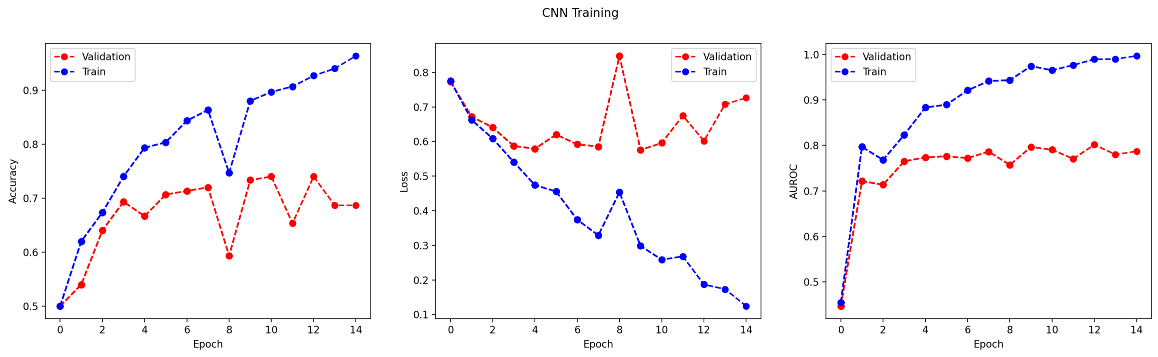
Training plot of Rotation (Keep original)



Training plot of Grayscale (keep original)



Training plot of Grayscale (discard original)



(c)

When we apply rotation, the performance is roughly the same compared to when there is no augmentation. This might be because the validation and testing dataset doesn't contain many rotated objects. It's also possible that the rotation results in loss of information and the model is underfitted. One other possibility is that the augmentated samples aren't sufficient for the model to learn about potential rotated objects.
When we apply grayscale, either keeping or discarding the original images, we witness a great increase in testing performance. This could be due to the fact that grayscale essentially reduces the noise and thus prevents overfitting. However, one exception is that when we discard the original images, the model does worse on the validation data. This may be due to loss of information from the original dataset and insufficient training samples. However, the testing performance remains better that when we don't do any augmentation because of noise reduction.

Challenge (See appendix F)

Regularization: I decided to apply Ridge Regularization by setting the weight decay to 0.01. I also added a dropout feature with a probability of 0.5. Both of these features help reduce overfitting. I tried using l1 regularization, but it doesn't seem to help with the performance.

Feature selection: I didn't implement feature selection because I lack understanding of the dataset. However, if we know the dataset enough, for example, the dataset is ranked from the least blurry to the most blurry images, we can use only the top 50% of the dataset because they are better training data.

Model architecture: Initially, I tried using a VGG16 model. I realized that the dataset we are given (64x64) isn't really compatible with VGG16 (requires 224x224). For this reason, I cannot implement 5 convolutonal layers. Eventually, I decided to use a modify the archietecture the appendix provides, with three convolutional layer. The first one has 32 filters, the second one has 64 filter, and the third one has 32 filters. There are two max pooling layers between these three convolutional layers. After that, I set up one fully connected layers that takes in 128 inputs.

Hyperparameters: I decide to use a patience of 10 because our learning rate is reletively high. A high learning rate means that it is likely that our loss oscillates between the left and right side of the minimum. Therefore, a higher patience allows extra time for the model to converge to a potential global minimum.

Transfer learning: I decide to use transfer learning to boost the performance of my model. As we can see from previous parts, using transfer learning could help improve our model's accuracy when we also freeze some layers of the CNN. I modified the source model so that it is compatible with the challenge model in terms of the architecture. After some experimentation, I chose to freeze all convolutional layers except for the fully connected layer because it gives the best performance. This might be because we are perserving more features than when we applied filters on images, which means that our model can learn more about the dataset.

Data augmentation: I decide not to apply any data augmentation because it results in really bad performance compared to when there is no augmentation. I think this might be because the mixture of using transfer learning and data agumentation results in the model being overfit, but there could be other reasons behind this. We can potentially investigate the how these two methods affect each other in the future.

Model evaluation: I decide to stick with validation loss as the evaluation metric because using any training metric will likely result in overfitting. Using validation accuracy or AUROC might be good. However, we don't have enough insight of the dataset, and if it is heavily imbalanced, using neither of these metrics gives an accurate description of the performance. So validation loss remains the best metric.

I disabled CUDA acceleration.

Appendix A

In [ ]:
```
"""
EECS 445 - Introduction to Machine Learning
Winter 2022 - Project 2
Dogs Dataset
    Class wrapper for interfacing with the dataset of dog images
    Usage: python dataset.py
"""

import os
import random
import numpy as np
import pandas as pd
import torch
from matplotlib import pyplot as plt
from imageio import imread
from PIL import Image
from torch.utils.data import Dataset, DataLoader
from utils import config


def get_train_val_test_loaders(task, batch_size, **kwargs):
    """Return DataLoaders for train, val and test splits.

    Any keyword arguments are forwarded to the DogsDataset constructor.
    """
    tr, va, te, _ = get_train_val_test_datasets(task, **kwargs)

    tr_loader = DataLoader(tr, batch_size=batch_size, shuffle=True)
    va_loader = DataLoader(va, batch_size=batch_size, shuffle=False)
    te_loader = DataLoader(te, batch_size=batch_size, shuffle=False)

    return tr_loader, va_loader, te_loader, tr.get_semantic_label


def get_challenge(task, batch_size, **kwargs):
    """Return DataLoader for challenge dataset.

    Any keyword arguments are forwarded to the DogsDataset constructor.
    """
    tr = DogsDataset("train", task, **kwargs)
    ch = DogsDataset("challenge", task, **kwargs)

    standardizer = ImageStandardizer()
    standardizer.fit(tr.X)
    tr.X = standardizer.transform(tr.X)
```

```python
    ch.X = standardizer.transform(ch.X)

    tr.X = tr.X.transpose(0, 3, 1, 2)
    ch.X = ch.X.transpose(0, 3, 1, 2)

    ch_loader = DataLoader(ch, batch_size=batch_size, shuffle=False)
    return ch_loader, tr.get_semantic_label


def get_train_val_test_datasets(task="default", **kwargs):
    """Return DogsDatasets and image standardizer.

    Image standardizer should be fit to train data and applied to all splits.
    """
    tr = DogsDataset("train", task, **kwargs)
    va = DogsDataset("val", task, **kwargs)
    te = DogsDataset("test", task, **kwargs)

    # Resize
    # We don't resize images, but you may want to experiment with resizing
    # images to be smaller for the challenge portion. How might this affect
    # your training?
    # tr.X = resize(tr.X)
    # va.X = resize(va.X)
    # te.X = resize(te.X)

    # Standardize
    standardizer = ImageStandardizer()
    standardizer.fit(tr.X)
    tr.X = standardizer.transform(tr.X)
    va.X = standardizer.transform(va.X)
    te.X = standardizer.transform(te.X)

    # Transpose the dimensions from (N,H,W,C) to (N,C,H,W)
    tr.X = tr.X.transpose(0, 3, 1, 2)
    va.X = va.X.transpose(0, 3, 1, 2)
    te.X = te.X.transpose(0, 3, 1, 2)

    return tr, va, te, standardizer


def resize(X):
    """Resize the data partition X to the size specified in the config file.

    Use bicubic interpolation for resizing.

    Returns:
        the resized images as a numpy array.
    """
    image_dim = config("image_dim")
    image_size = (image_dim, image_dim)
    resized = []
    for i in range(X.shape[0]):
        xi = Image.fromarray(X[i]).resize(image_size, resample=2)
        resized.append(xi)
    resized = [np.asarray(im) for im in resized]

    return resized


class ImageStandardizer(object):
    """Standardize a batch of images to mean 0 and variance 1.

    The standardization should be applied separately to each channel.
    The mean and standard deviation parameters are computed in `fit(X)` and
    applied using `transform(X)`.

    X has shape (N, image_height, image_width, color_channel)
    """

    def __init__(self):
        """Initialize mean and standard deviations to None."""
        super().__init__()
        self.image_mean = None
        self.image_std = None

    def fit(self, X):
        """Calculate per-channel mean and standard deviation from dataset X."""
        # TODO: Complete this function
        print(X.shape)
        self.image_mean = np.mean(X, axis=(0,1,2))
        self.image_std = np.std(X, axis=(0,1,2))
    def transform(self, X):
        """Return standardized dataset given dataset X."""
        # TODO: Complete this function
        X_transformed = (X - self.image_mean) / self.image_std
        return X_transformed


class DogsDataset(Dataset):
    """Dataset class for dog images."""

    def __init__(self, partition, task="target", augment=False):
        """Read in the necessary data from disk.

        For parts 2, 3 and data augmentation, `task` should be "target".
        For source task of part 4, `task` should be "source".

        For data augmentation, `augment` should be True.
        """
        super().__init__()

        if partition not in ["train", "val", "test", "challenge"]:
            raise ValueError("Partition {} does not exist".format(partition))

        np.random.seed(42)
        torch.manual_seed(42)
        random.seed(42)
        self.partition = partition
        self.task = task
        self.augment = augment
        # Load in all the data we need from disk
        if task == "target" or task == "source":
            self.metadata = pd.read_csv(config("csv_file"))
        if self.augment:
            print("Augmented")
            self.metadata = pd.read_csv(config("augmented_csv_file"))
        self.X, self.y = self._load_data()

        self.semantic_labels = dict(
            zip(
                self.metadata[self.metadata.task == self.task]["numeric_label"],
                self.metadata[self.metadata.task == self.task]["semantic_label"],
            )
        )

    def __len__(self):
        """Return size of dataset."""
        return len(self.X)

    def __getitem__(self, idx):
        """Return (image, label) pair at index `idx` of dataset."""
        return torch.from_numpy(self.X[idx]).float(), torch.tensor(self.y[idx]).long()

    def _load_data(self):
        """Load a single data partition from file."""
        print("loading %s..." % self.partition)

        df = self.metadata[
            (self.metadata.task == self.task)
            & (self.metadata.partition == self.partition)
        ]

        if self.augment:
            path = config("augmented_image_path")
        else:
            path = config("image_path")

        X, y = [], []
        for i, row in df.iterrows():
            label = row["numeric_label"]
            image = imread(os.path.join(path, row["filename"]))
            X.append(image)
            y.append(row["numeric_label"])
        return np.array(X), np.array(y)

    def get_semantic_label(self, numeric_label):
        """Return the string representation of the numeric class label.

        (e.g., the numberic label 1 maps to the semantic label 'miniature_poodle').
        """
        return self.semantic_labels[numeric_label]


if __name__ == "__main__":
    np.set_printoptions(precision=3)
    tr, va, te, standardizer = get_train_val_test_datasets(task="target", augment=False)
    print("Train:\t", len(tr.X))
    print("Val:\t", len(va.X))
    print("Test:\t", len(te.X))
    print("Mean:", standardizer.image_mean)
    print("Std: ", standardizer.image_std)
```

Appendix B

In [ ]:

```
"""
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

from math import floor


class Target(nn.Module):
    def __init__(self):
        super().__init__()

        # TODO: define each layer
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=(
            5, 5), stride=(2, 2), padding=2)
        self.pool = nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=64, kernel_size=(
            5, 5), stride=(2, 2), padding=2)
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=8, kernel_size=(
            5, 5), stride=(2, 2), padding=2)
        self.fc_1 = nn.Linear(in_features=32, out_features=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 / sqrt(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 = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = x.view(-1, 32)
        x = self.fc_1(x)

        ##

        return x
```

In [ ]:

```
"""
EECS 445 - Introduction to Machine Learning
Winter 2022 - Project 2
```

```python
Train Target
    Train a convolutional neural network to classify images.
    Periodically output training information, and saves model checkpoints
    Usage: python train_target.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
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

    track = num_layers * 2

    for name, param in model.named_parameters():
        if track == 0:
            break

        param.requires_grad = False
        track -= 1

    for name, param in model.named_parameters():
        print(name, param.requires_grad)


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(params=model.parameters(), lr=1e-3)
    #

    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)

    train(tr_loader, va_loader, te_loader, freeze_none, "./checkpoints/target0/", 0)
    train(tr_loader, va_loader, te_loader, freeze_one, "./checkpoints/target1/", 1)
    train(tr_loader, va_loader, te_loader, freeze_two, "./checkpoints/target2/", 2)
    train(tr_loader, va_loader, te_loader, freeze_three, "./checkpoints/target3/", 3)


if __name__ == "__main__":
    main()
```

In [ ]:

```
"""
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 = {
```

```python
        "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:
        curr_count_to_patience = 0
        global_min_loss = stats[-1][1]

    #
    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
        y_pred = model(X)

        optimizer.zero_grad()

        loss = criterion(y_pred, y)
        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
    pred = np.argmax(logits, axis=1)
    return pred
```

Appendix C

In [ ]:

```
"""
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
        # self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=(
        #     5, 5), stride=(2, 2), padding=2)
        # self.pool = nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0)
        # self.conv2 = nn.Conv2d(in_channels=16, out_channels=64, kernel_size=(
        #     5, 5), stride=(2, 2), padding=2)
        # self.conv3 = nn.Conv2d(in_channels=64, out_channels=8, kernel_size=(
        #     5, 5), stride=(2, 2), padding=2)
        # self.fc1 = nn.Linear(in_features=32, out_features=8)

        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=(
            5, 5), stride=(2, 2), padding=2)
        self.pool = nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=(
            5, 5), stride=(2, 2), padding=2)
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=32, kernel_size=(
            5, 5), stride=(2, 2), padding=2)
        self.fc1 = nn.Linear(in_features=128, out_features=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 / sqrt(128))
        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 = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = x.view(-1, 128)
        x = self.fc1(x)

        ##

        return x
```

In [ ]:

```
"""
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(), weight_decay=0.01, lr=1e-3)
    #
    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()
```

Appendix D

In [ ]:

```
"""
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
```

```python
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(params=model.parameters(), lr=1e-3)
    #

    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()
```

Appendix E

In [ ]:
```python
"""
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
        return rotate(
            input=img,
            angle=np.random.randint(-deg, deg),
            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
        avg = np.round(img.mean(axis=-1), 0).astype(np.uint8)
        avg = np.stack([avg] * 3, axis=-1)
        return avg

    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()]

    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:]))
```

Appendix F

In [ ]:

```
"""
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
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=(
            5, 5), stride=(2, 2), padding=2)
        self.pool = nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=(
            5, 5), stride=(2, 2), padding=2)
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=32, kernel_size=(
            5, 5), stride=(2, 2), padding=2)
        self.fc_1 = nn.Linear(in_features=128, out_features=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)

        # TODO: initialize the parameters for [self.fc1]
        nn.init.normal_(self.fc_1.weight, 0.0, 1 / sqrt(128))
        nn.init.constant_(self.fc_1.bias, 0.0)

        # nn.init.normal_(self.fc_2.weight, 0.0, 1 / sqrt(32))
        # nn.init.constant_(self.fc_2.bias, 0.0)

        # nn.init.normal_(self.fc_3.weight, 0.0, 1 / sqrt(32))
        # nn.init.constant_(self.fc_3.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 = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = x.view(-1, 128)
        x = self.fc_1(x)

        ##

        return x
```

In [ ]:

```
import torch
import numpy as np
import random
from dataset import get_train_val_test_loaders
from model.target import Target
from model.challenge import Challenge
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

    track = num_layers * 2

    for name, param in model.named_parameters():
        if track == 0:
            break

        param.requires_grad = False
        track -= 1

    for name, param in model.named_parameters():
        print(name, param.requires_grad)
```

```python
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(params=model.parameters(), weight_decay=0.01, lr=1e-3)
    #

    print("Loading target model with", num_layers, "layers frozen")
    model, start_epoch, stats = restore_checkpoint(model, model_name)

    axes = utils.make_training_plot("Challenge 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 = 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,
            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 4 layers frozen.
    """
    # 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 = Challenge()

    torch.nn.Dropout(p=0.5)

    freeze_three = copy.deepcopy(model)

    freeze_layers(freeze_three, 3)
    print("Loading source...")
    freeze_three, _, _ = restore_checkpoint(
        freeze_three, config("source.checkpoint"), force=True, pretrain=True
    )

    train(tr_loader, va_loader, te_loader,
          freeze_three, "./checkpoints/challenge3/", 3)


if __name__ == "__main__":
    main()
```