Project 2

● Graded

Student

Cooper Kennelly

Total Points

87 / 94 pts

Question 1

Q1 Data Preprocessing **10** / 10 pts

1.1 **Q1 a.i** **4** / 4 pts

✔ **+ 4 pts** Correct

+ 2 pts Correct mean

+ 2 pts Correct standard deviation

+ 0 pts Incorrect / Incomplete Response

1.2 **Q1 a.ii** **4** / 4 pts

✔ **+ 4 pts** Correct explanation

+ 2 pts Attempts explanation but doesn't identify maintaining validity of heldout data.

+ 0 pts Incorrect / Incomplete Response

1.3 **Q1 b** **2** / 2 pts

✔ **+ 2 pts** Fully Correct

+ 0 pts Incorrect / Incomplete Response

Question 2

Q2 Convolutional Neural Networks — **27** / 30 pts

2.1 Q2 a — **3** / 3 pts

- ✔ **+ 3 pts** Fully Correct
- **+ 1 pt** All convolutional layers correct
- **+ 0.5 pts** At least one convolutional layers correct
- **+ 1 pt** Fully-connected layer correct
- **+ 1 pt** Max Pool layers have no learnable float-valued parameters
- **+ 0 pts** Incorrect / Incomplete Response

2.2 Q2 f.i — **4** / 4 pts

- ✔ **+ 4 pts** Describes 2 or more valid sources
- **+ 2 pts** Describes one valid source
- **+ 0 pts** Incorrect / Incomplete Response

2.3 Q2 f.ii — **4** / 6 pts

- **+ 6 pts** Fully Correct
- ✔ **+ 1 pt** Stopping epoch for patience of 5 plot has lowest validation loss 5 epochs before end
- ✔ **+ 1 pt** Stopping epoch for patience of 10 plot has lowest validation loss 10 epochs before end
- **+ 2 pts** Correctly discusses how patience 5 or 10 is better
- ✔ **+ 2 pts** Correctly discusses scenario
- **+ 0 pts** Incorrect / Incomplete Response

2.4 Q2 f.iii — **9** / 9 pts

- ✔ **+ 9 pts** Fully Correct
- **+ 1 pt** Correct new FC layer size
- **+ 2 pts** Correct observation
- **+ 2 pts** Correct observation explanation
- **+ 2 pts** Correct AUROC
- **+ 2 pts** Training AUROC is greater than corresponding validation AUROC
- **+ 0 pts** Incorrect / Incomplete Response

2.5 **Q2 g.i** **3** / 3 pts

- ✔ **+ 3 pts** Fully Correct
- **+ 1 pt** Correct training and validation AUROC
- **+ 1 pt** Correct testing AUROC
- **+ 1 pt** Correct testing and validation accuracy
- **+ 0 pts** Incorrect / Incomplete Response

2.6 **Q2 g.ii** **2** / 2 pts

- ✔ **+ 2 pts** Fully Correct
- **+ 1 pt** There is little/no overfitting
- **+ 1 pt** Explanation using training/validation performance
- **+ 0 pts** Incorrect / Incomplete Response

2.7 **Q2 g.iii** **2** / 3 pts

- **+ 3 pts** Fully Correct
- ✔ **+ 1 pt** Correct discussion between validation and testing performance
- ✔ **+ 1 pt** States that validation and test data differ in some way
- **+ 1 pt** Correct explanation
- **+ 0 pts** Incorrect / Incomplete Response

Question 3

Q3 Visualizing what the CNN has learned **10** / 10 pts

3.1 **Q3 a** **5** / 5 pts

- ✔ **+ 5 pts** Fully Correct
- **+ 2 pts** Correctly calculates α
- **+ 1 pt** Correctly calculates $\alpha * A + \alpha * A$
- **+ 2 pts** Correctly calculates L
- **+ 0 pts** Incorrect / Incomplete Response

3.2 **Q3 b** **3** / 3 pts

- ✔ **+ 3 pts** Fully Correct explanation
- **+ 1.5 pts** Explains other features
- **+ 0 pts** Incorrect / Incomplete Response

3.3 **Q3 c** **2** / 2 pts

- ✔ **+ 2 pts** Fully Correct
- **+ 0 pts** Incorrect / Incomplete Response

Question 4

Q4.1 Transfer Learning **12** / 14 pts

4.1 **Q4.1 c** **2** / 2 pts

- ✔ **+ 2 pts** Fully Correct
- **+ 1 pt** Correct plot and trends
- **+ 1 pt** Correct epoch selected
- **+ 0 pts** Incorrect / Incomplete Response

4.2 **Q4.1 d** **4** / 4 pts

- ✔ **+ 4 pts** Fully Correct
- **+ 1 pt** Correct confusion matrix
- **+ 1 pt** Correct landmark with highest accuracy
- **+ 1 pt** Correct landmark with least accuracy
- **+ 1 pt** Valid analysis/justification
- **+ 0 pts** Incorrect / Incomplete Response

4.3 **Q4.1 f** **6** / 8 pts

- **+ 8 pts** Fully Correct
- ✔ **+ 1 pt** Correct plot for No frozen layers
- ✔ **+ 1 pt** Correct plot for First layer frozen
- ✔ **+ 1 pt** Correct plot for First two layers frozen
- ✔ **+ 1 pt** Correct plot for All three layers frozen
- ✔ **+ 2 pts** Correct explanation for transfer learning
- **+ 2 pts** Correct trend in freezing layers with test AUROC
- **+ 0 pts** Incorrect / Incomplete Response

Question 5

Q4.2 Data Augmentation **11** / 11 pts

5.1 **Q4.2 b.iii** **8** / 8 pts

✔ **+ 8 pts** Fully Correct

+ 1 pt Same performance in 2g

+ 1 pt Grayscale (discard original) has smallest difference between validation and test AUROC

+ 2 pts Grayscale (discard original) has best test AUROC

+ 4 pts Correct training plots

+ 0 pts Incorrect / blank

5.2 **Q4.2 c** **3** / 3 pts

✔ **+ 3 pts** Correct

+ 0 pts Incorrect / page not selected

Question 6

Q5 Challenge - Report **7** / 9 pts

+ 9 pts Fully Correct

✔ **+ 1 pt** Discuss regularization techniques and justification

+ 1 pt Discuss preprocessing and justification

✔ **+ 1 pt** Discuss model architecture and changes made

✔ **+ 1 pt** Discuss hyperparameter selection

+ 1 pt Discuss transfer learning source and target task and setup

✔ **+ 1 pt** Discuss data augmentation and justification

✔ **+ 1 pt** Discuss metrics for evaluation

✔ **+ 2 pts** Write-up demonstrates effort into experimenting and modifying problem setup

+ 0 pts Minimal effort

+ 0 pts Incorrect / Incomplete Response

Question 7

Q6 Code Appendix **10** / 10 pts

✔ **+ 10 pts** Fully Correct

+ 0 pts Incorrect / Incomplete Response

Questions assigned to the following page: 1.1, 1.2, and 1.3

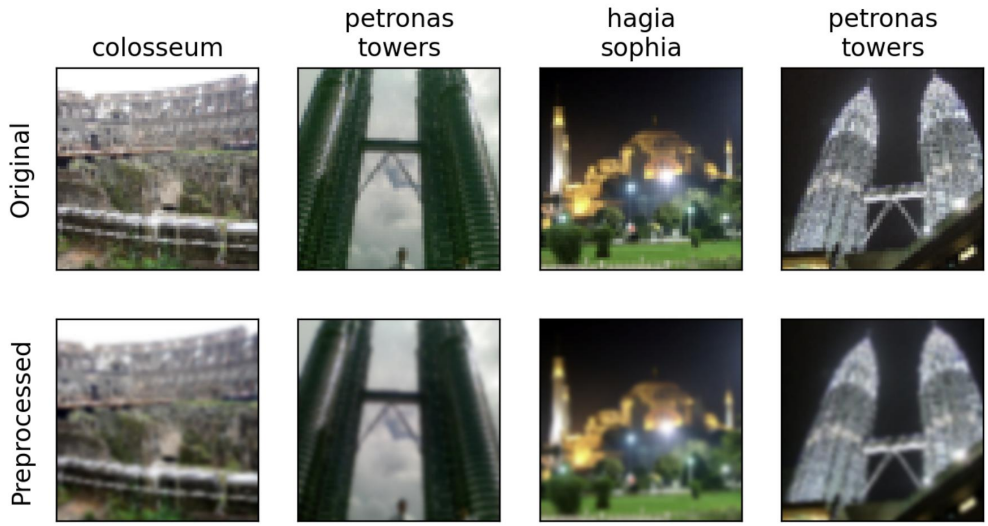
Question 1

A)

i) The mean values for the RGB channels, as learned from the entire training partition, are: R channel has a mean of 118.263, G channel has a mean of 118.004, and the B channel has a mean of 118.594. Correspondingly, the standard deviations for these channels are: R channel is 66.307, G channel is 69.039, and B channel is 75.332.

ii) Extracting the mean and standard deviation from the training set, rather than from the other data partitions, is crucial for a few reasons. First, it prevents data leakage. If we use statistics from the entire dataset, which includes our validation and test sets, our model could inadvertently gain information about these sets, leading to an unrealistically optimistic assessment of its performance. It's imperative that our model remains uninformed about the validation and test datasets to ensure that our evaluations are unbiased. Secondly, this approach ensures our model generalizes well. In real-world applications, when we deploy models, we'll normalize new, unseen data points using statistics from the training data. Hence, our validation and testing during the development should mirror this scenario. Lastly, using the training set's statistics ensures that our normalization process remains consistent. If we were to normalize using statistics from the entire dataset, these numbers would shift each time we split our dataset differently, which would be an unstable and inconsistent approach.

B)



Question assigned to the following page: 2.1

Question 2

A)

Filter Size * Number of Filters * Depth and include the Bias per Filter as well as parameters for the fully connected layer

5*5 (16*3 + 16*64 + 8*64) + 16 + 64 + 8 + 2 + 64 = 39754

B)

C)

D)

E)

Question assigned to the following page: 2.2

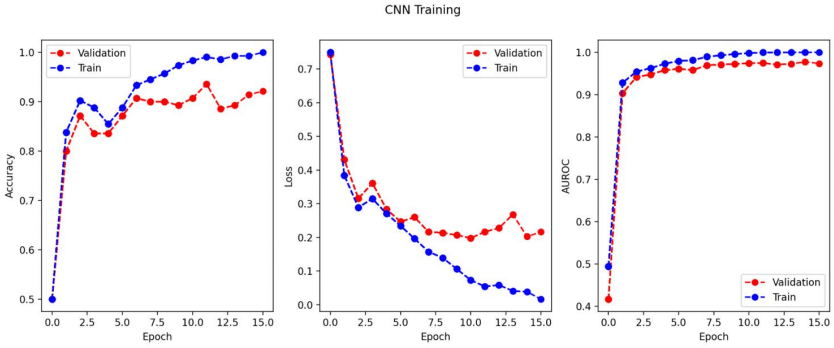
F)

i) Overfitting ~ as the training progresses, the model might start to overfit and rely on specific, chance attributes of the training data more than is actually correct in a theoretical manner, especially if the model is complex with many parameters. This can lead to a scenario where the training loss continues to decrease because the model is fitting more closely to the training data, but the validation loss starts to increase since the model is not generalizing well to new, unseen data. In the provided plots, it can be observed that the training loss consistently decreases over the epochs, but the validation loss decreases initially and then increases. This is a classic scenario for overfitting.

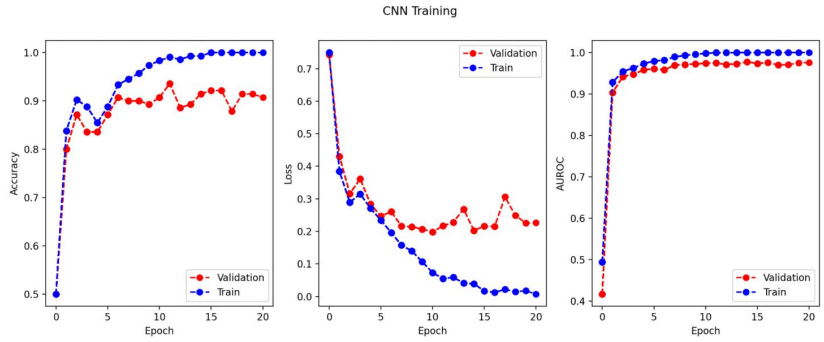
Learning Rate and Optimization Issues ~ the learning rate might be set too high, causing the model to overshoot the optimal values in the loss landscape, leading to fluctuations in the validation loss. Alternatively, if the learning rate is too low, the model might not learn effectively, causing slow or no improvement in the validation loss. It's also possible that the optimizer might get stuck in local minima or saddle points, causing the validation loss not to decrease monotonically.

Questions assigned to the following page: 2.3 and 2.4

ii) With a patience of five, it stopped at epoch 15. With a patience of ten, it stops at epoch 20.
Below is the graph of patience = 5
With a patience of ten it stops at Epoch 20 rather than 15 ~ if we evaluate better based on the patience, then both perform equally well since they end based off the same epoch, epoch 10. Increased Patience would work in a data-environment where the training is highly random and where the training hyperplane is highly non-convex, leading to lots of local minima which must be escaped over larger number of epochs to further approximate the global minimum.



Below is the graph of patience = 10



iii)

New Size : 256

Questions assigned to the following page: 2.4, 2.5, 2.6, and 2.7

	EPOCH	Train. AUROC	Val. AUROC
8 filters	10	0.9982	0.9743
64 filters	7	0.9993	0.9694

Differences in plots : The plot for the architecture with 64 filters has training and validation loss diverge much more significantly much sooner ~ it begins to significantly diverge at the third epoch and by the seventh the difference in loss is .2. The plot for the architecture with 8 filters doesn't significantly diverge until the seventh epoch and only gets to a divergence of .2 at the fifteenth training epoch. This suggests that the 64 filter model is too complex and is almost immediately causing overfitting.

G)

i)

	Training	Validation	Testing
Accuracy	0.9833	0.9071	0.625
AUROC	0.9982	0.9743	0.6827

ii) The difference is not-negligible but also not sufficiently significant to definitively state that there is overtraining. This could be valid performance, we would need to see the trend of validation performance over more training cycles to see if it is increasing. I would say there is not evidence of overtraining present.

iii) The validation and test performance are significantly different, indicating that there is a large gap between what is represented with our training and validation data, and our test data.

Questions assigned to the following page: 3.1, 3.2, and 3.3

Question 3

A) $\alpha_1^1 = \frac{1}{16}\sum\sum\frac{dy^c}{dA_{ij}} = \frac{1}{16}(-1+1-2-1-1+1+1+1+2+2) = \frac{3}{16}$

$\alpha_2^1 = \frac{1}{16}\sum\sum\frac{dy^c}{dA_{ij}} = \frac{1}{16}(1+2+2+2+2+2+1+1-1-2-1) = \frac{7}{16}$

$L^1 = ReLU(\sum \alpha_1 A^1)$

$B = [[1,1,2,1],[1,2,1,0],[0,1,0,-1],[0,1,-2,-2]]$

$C = [[1,1,1,1],[2,2,2,2],[2,2,1,0],[-1,-1,-1,0]]$

$= ReLU(\frac{3}{16} * B + \frac{7}{16} * C)$

$= [[\frac{5}{8},\frac{5}{8},\frac{13}{16},\frac{5}{8}],[\frac{17}{16},\frac{5}{4},\frac{17}{16},\frac{7}{8}],[\frac{7}{8},\frac{17}{16},\frac{7}{16},\frac{0}{}],[0,0,0,0]]$

B) The CNN appears to be using the horse statue as well as the top of the palace to identify the Hofburg Imperial Palace.

C) Yes, the grad-cam visualizations confirm the performance earlier in the report. The training performance was extremely high, and the differentiation of these two architectures based on the Pantheon's columns and the Hofburg Palace's roofline makes sense as these features are extremely distinguishing between the two of them. However, If the angle of the photos in the test data is such that we do not have a frontal-view of the pantheon or do not have the roof-line of the Hofburg in the photo, then it is likely the model will perform significantly worse.

Question assigned to the following page: 4.1

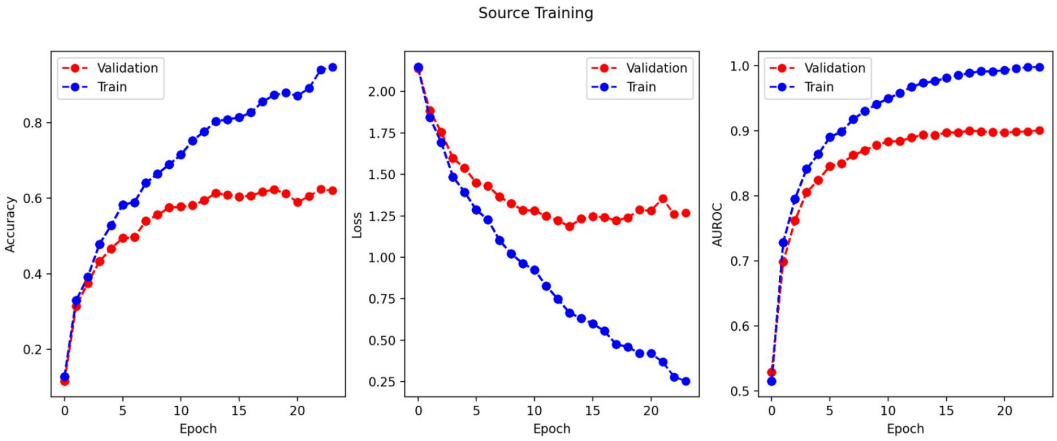
Question 4

Portion 4.1

A) (code)

B) (code)

C)



Epoch : 13

Epoch 13

Validation Accuracy:0.6133

Validation Loss:1.1857

Validation AUROC:0.8941

Train Accuracy:0.8035

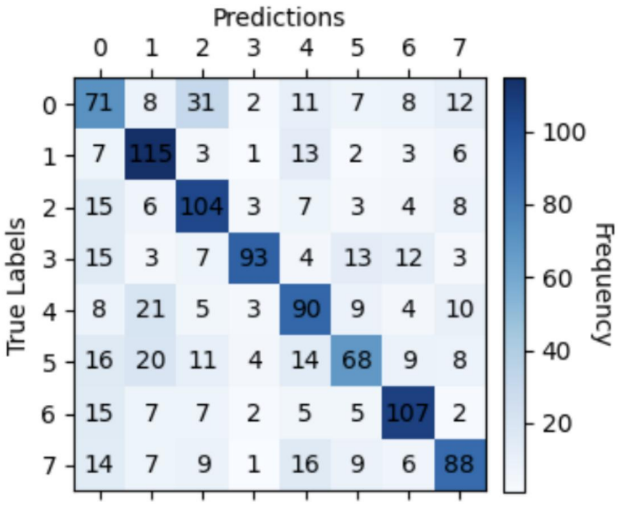
Train Loss:0.6652

Train AUROC:0.9737

Question assigned to the following page: 4.2

D) The Petronas Towers, Rialto Bridge, and Hagia Sophia all have extremely distinct geometries and there is only really one perspective on the landmark, so all the images are not only distinct from the other landmarks but also internally very similar leading to a very high true-positive rate.

0 - Colosseum
1 - Petronas Towers
2 - Rialto Bridge
3 - Museu Nacional d'Art de Catalunya
4 - St Stephen's Cathedral in Vienna
5 - Berlin Cathedral
6 - Hagia Sophia
7 - Gaudi Casa Batllo in Barcelona



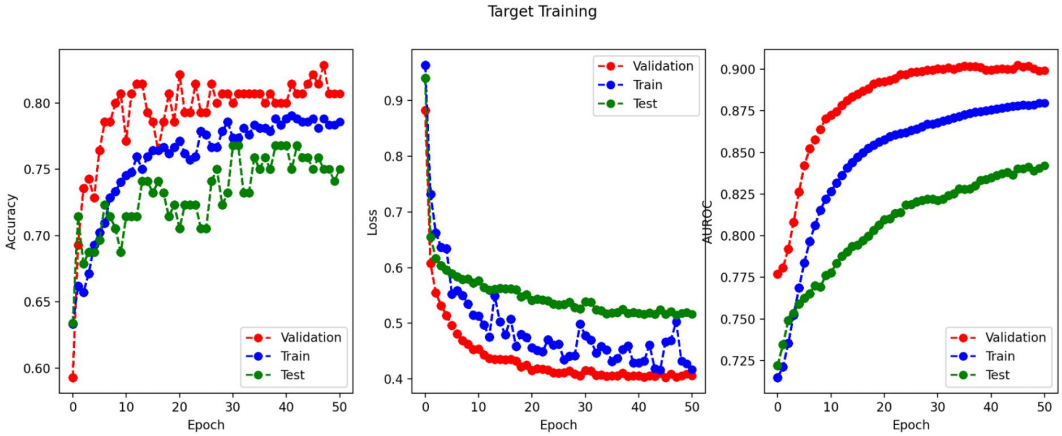
E) (code)

Question assigned to the following page: 4.3

F)

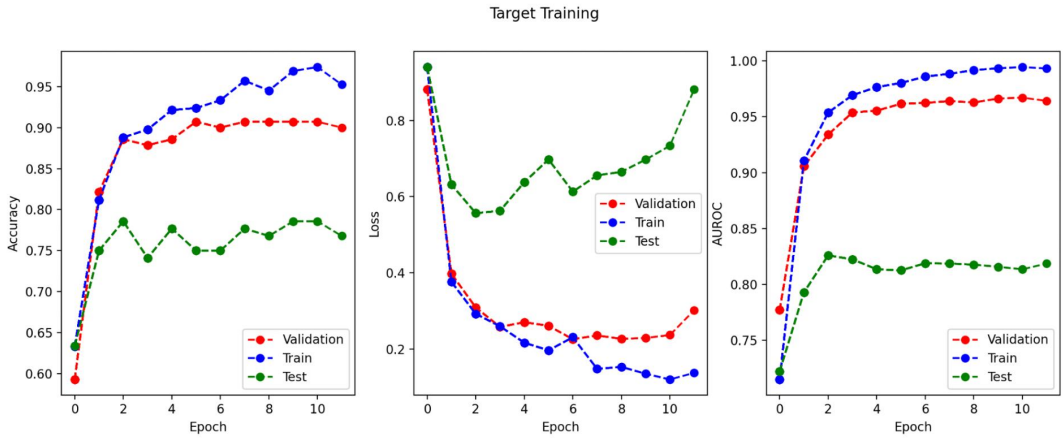
	AUROC	AUROC	AUROC
	TRAIN	TEST	VALIDATION
Freeze All Conv	0.878	0.8402	0.9024
Freeze First 2 Conv	0.986	0.8189	0.9624
Freeze First Conv	0.9999	0.809	0.9773
Freeze None	0.983	0.7379	0.9718
No pretraining or Transfer Learning (2(g))	0.9982	0.6827	0.9743

All Convolutional Layers Frozen

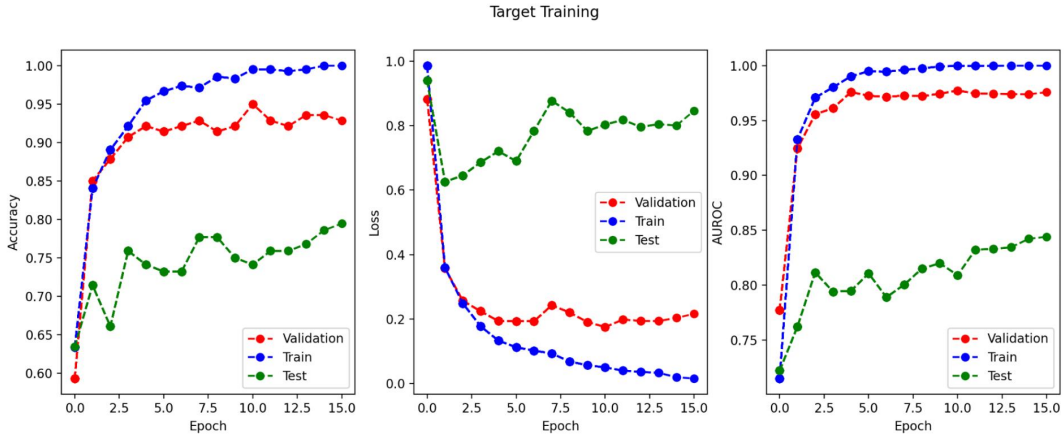


Two Frozen Layers

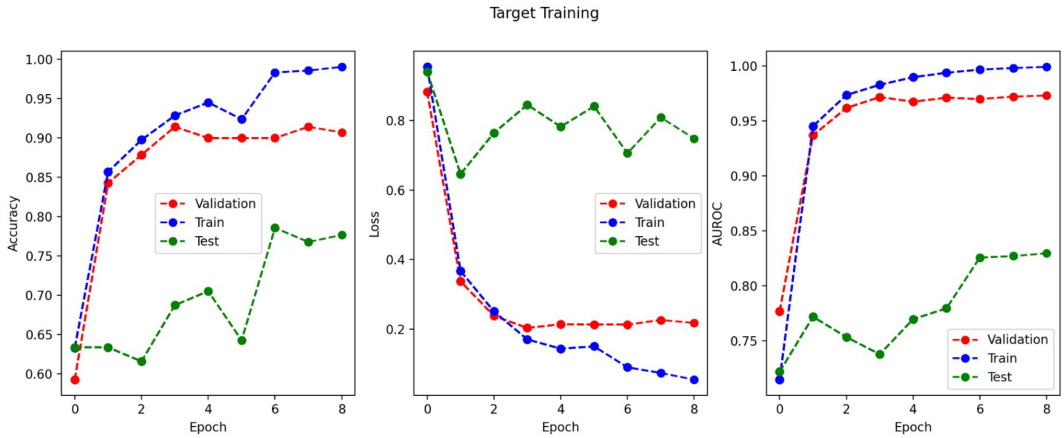
Question assigned to the following page: 4.3



One Frozen Layer



No Frozen Layers



Question assigned to the following page: 4.3

Effect of Transfer Learning ~ When we look at the results, using transfer learning seems to have a significant impact on performance. Models that used some form of transfer learning generally outperformed the model that did not leverage source task data, especially in the test set. This suggests that the source task was indeed helpful. Leveraging pre-existing knowledge (weights) from a different but related task can lead to quicker convergence and potentially better generalization to unseen data.

Examining Freezing Strategies:
- Freeze All Conv ~ This approach has the lowest train AUROC, indicating that when we freeze all convolutional layers, the model may not be capturing all the intricate patterns from the data. However, its test AUROC is relatively high, which may indicate a good generalization to unseen data.
- Freeze First 2 Conv ~ There's a noticeable drop in test performance compared to freezing all conv layers, suggesting some overfitting might be occurring.
- Freeze First Conv ~ Almost perfect training score suggests potential overfitting, though the validation score remains high. Test performance dropped slightly compared to the previous scenario.
- Freeze None ~ Here, all layers are trainable. This strategy seems to be overfitting the most as seen by a relatively high train score but a much lower test score.

Why this might occur
- Freezing layers means the model retains pre-trained features from the source task. When all layers are frozen, the model relies entirely on the source task's features. As layers become trainable, the model adjusts to the target data, which might lead to overfitting if the training data is not representative enough or if the model is too complex.
- Freezing just a subset allows the model to retain some general features (from the frozen layers) while fine-tuning others to the specific task. This can be seen as a balance between leveraging prior knowledge and adapting to new data.
- Not freezing any layer allows complete adaptation to the new task but might also mean losing some useful features from the source task.

Conclusion Transfer learning helps in leveraging information from related tasks. However, the extent to which we should adapt these pre-trained models (how many layers we should freeze or make trainable) depends on the similarity between the source and target tasks, as well as the complexity and representativeness of the training data.

Question assigned to the following page: 5.1

Question 4.2

A) (code)

B)

i) (code)

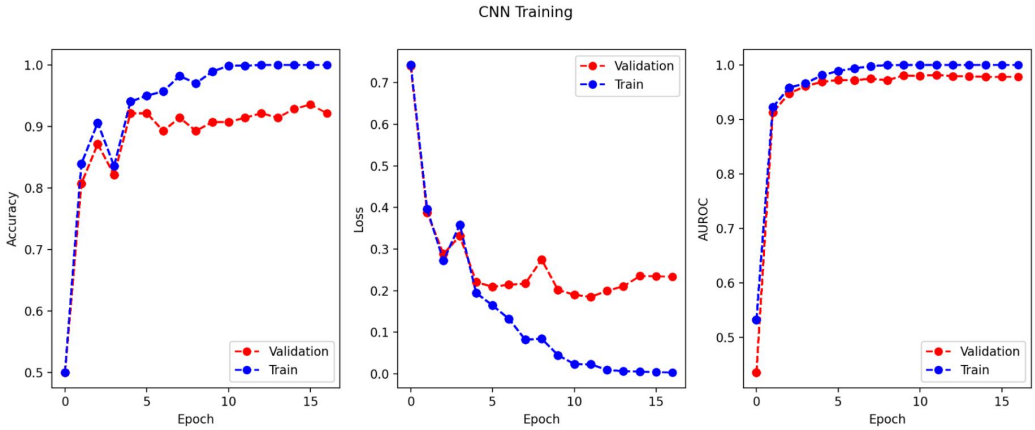
ii) (execution of code)

iii) The training and validation AUROC are mostly the same across the augmented and regular data except for the Grayscale w/o original Auroc which is significantly lower at .87 rather than .97 for the un-modified data. Furthermore, the test AUROC for all augmentations was higher than for the un-modified data, showing that all the augmentations bolstered model efficacy to some extent. It seems that Grayscale excluding the original photos seemed to have the strongest effect, and I'm not surprised based on the fact that this would likely lend itself best to forcing the model to ascertain geometries and shapes rather that slight / spurious patterns in color and areas.

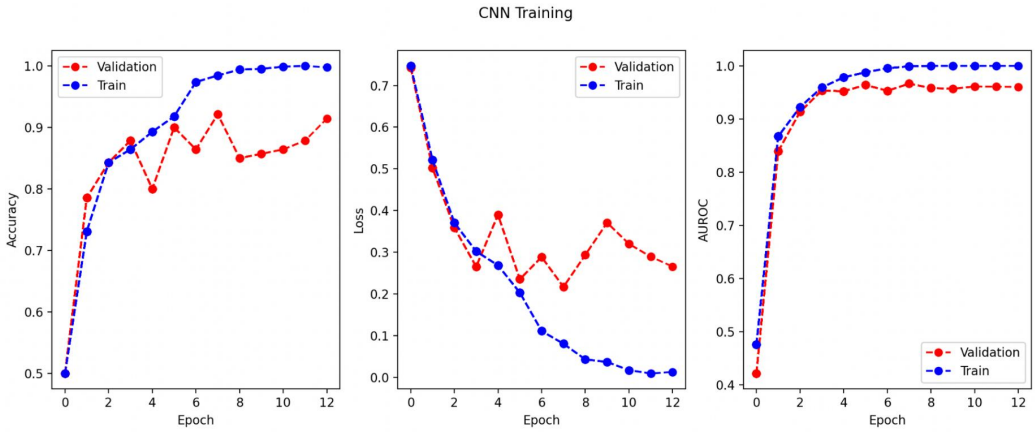
	AUROC	AUROC	AUROC
	Training	Validation	Test
Rotation (original)(11)	1.0	0.9818	0.751
Grayscale (original)(7)	0.9993	0.9665	0.7554
Grayscale(w/o orig)(7)	0.9889	0.8792	0.8198
No Aug. (2(g))	0.9982	0.9743	0.6827

Rotation With Original Images

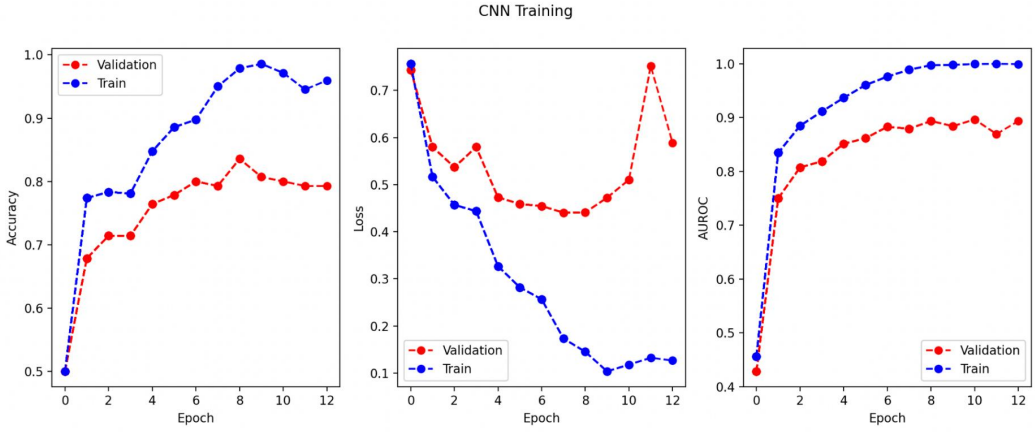
Questions assigned to the following page: 5.1 and 5.2



Grayscale With Original Images



Grayscale Without Original Images



C) Upon analyzing the plots, it's clear that the augmentation techniques influence the CNN's training and validation performances differently. When original images are combined with rotation or grayscale augmentations, there's a pronounced

Questions assigned to the following page: 5.2 and 6

divergence between training and validation metrics, suggesting overfitting. However, when grayscale augmentation is applied without original images, training and validation metrics are more aligned, indicating reduced overfitting. The change in the plots, as these augmentations are applied, likely stems from how the model interacts with the augmented data. Combining original images with augmentations might be reinforcing certain patterns too strongly, leading the model to over-rely on them, resulting in overfitting. On the other hand, grayscale augmentation without original images may be promoting better generalization. Additionally, the rotation might not provide enough diversity if the rotation range closely resembles the original images, which could explain the overfitting observed in its scenario.

Question 5 : Challenge

I will be minimizing for validation loss because I think given that is the grading constraint.

Model Architecture :

My first thought for the challenge section was the architecture of the model. People have spent entire PhDs focusing on specific model architectures, so I went online and found a few, and after attempting to implement several of the architectures, AlexNet was the most feasible given our input dimension is non-standard and smaller than all the implemented models. Beyond model architecture, I will also be using augmented data, and assessing performance across different sets of augmented data, specifically rotated with original, grayscale with original, as well as both with original, and both excluding original. After this, I will modify learning rate.

Based on the below tests across learning rate and weight decay for the modified ALexNet architecture, I chose epoch 34, lr=0.00001, weight_decay=0.005 for the final challenge model. It had the highest validation AUROC across all tested models.

AlexNet

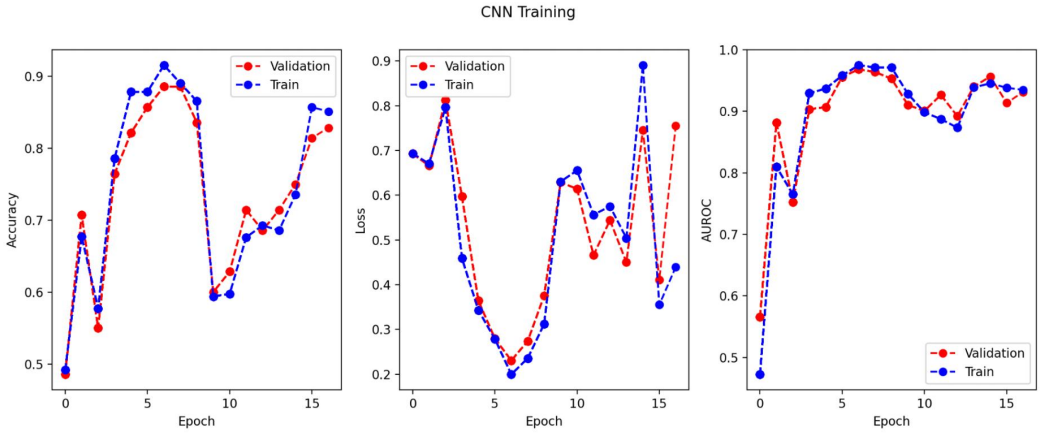
For all below : lr=1e-3, weight_decay=0.001

HyperPerams : patience 10, numClassses = 2
Data : Rotation, Include Original

Epoch 6
Validation Accuracy:0.8857
Validation Loss:0.2306
Validation AUROC:0.9684
Train Accuracy:0.9155

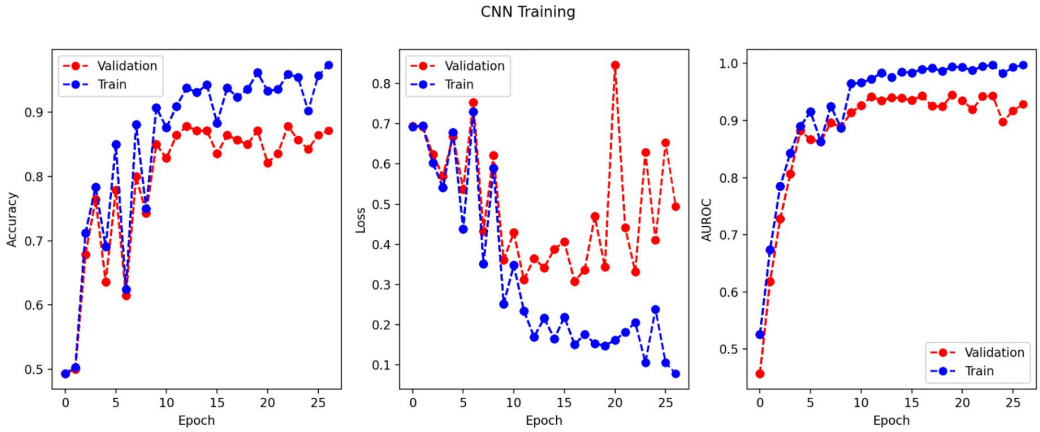
Question assigned to the following page: 6

Train Loss:0.1997
Train AUROC:0.9749



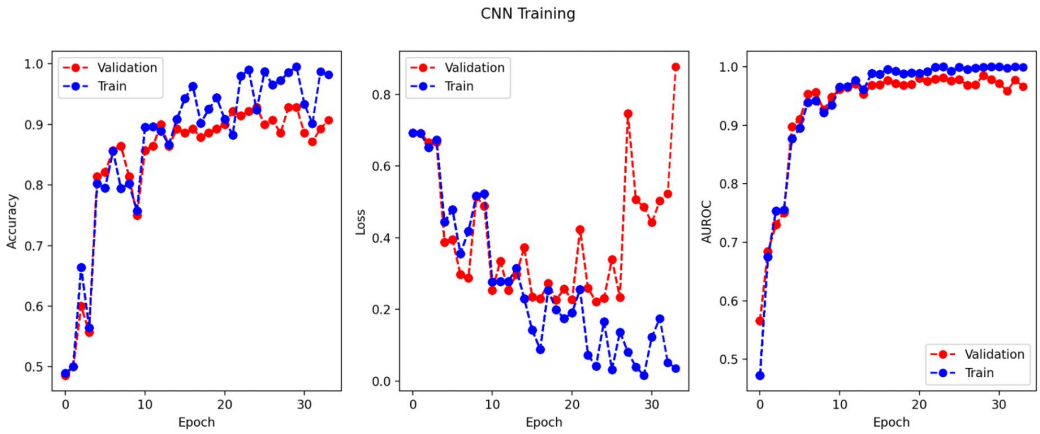
HyperPerams : patience : 10, numClasses = 2
Data : grayscale, exclude original

Epoch 16
Validation Accuracy:0.8643
Validation Loss:0.308
Validation AUROC:0.9433
Train Accuracy:0.9381
Train Loss:0.1508
Train AUROC:0.9895



HyperPerams : patience 10 , numClasses = 2
Data, Grayscale, Rotation, Rotation&Grayscale, originals included

Question assigned to the following page: 6



Epoch 23

- Validation Accuracy:0.9214
- Validation Loss:0.2218
- Validation AUROC:0.981
- Train Accuracy:0.9905
- Train Loss:0.0417
- Train AUROC:0.9997

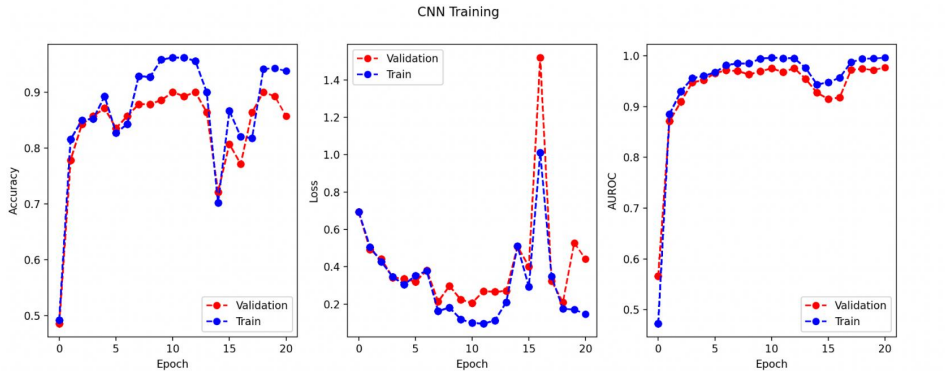
Now that we have determined including the broadest scale of augmented data leads to the highest validation auroc, I will test across a range of learning rates and weight decays

lr=0.001, weight_decay=0.0005

Epoch 10

- Validation Accuracy:0.9
- Validation Loss:0.2067
- Validation AUROC:0.9755
- Train Accuracy:0.9619
- Train Loss:0.1
- Train AUROC:0.9965

Question assigned to the following page: 6



lr=0.01, weight_decay=0.0005

Epoch 4

Validation Accuracy:0.5214

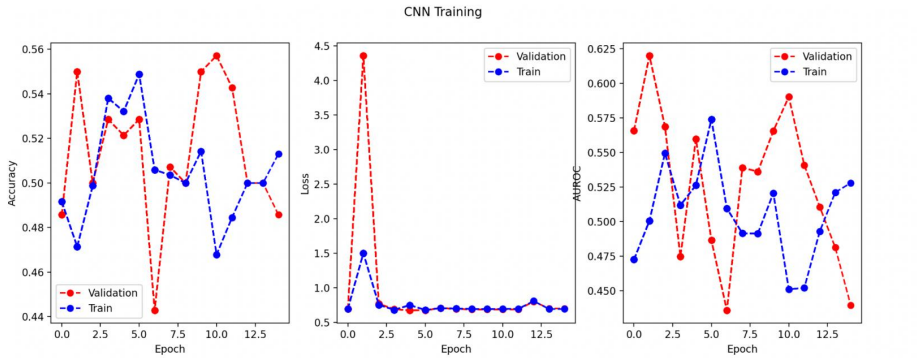
Validation Loss:0.6739

Validation AUROC:0.5598

Train Accuracy:0.5321

Train Loss:0.7501

Train AUROC:0.5263



That was clearly too large based on the initial loss and then massively variant changes in training performance per epoch.

lr=0.0001, weight_decay=0.0005

Epoch 9

Validation Accuracy:0.8857

Validation Loss:0.2012

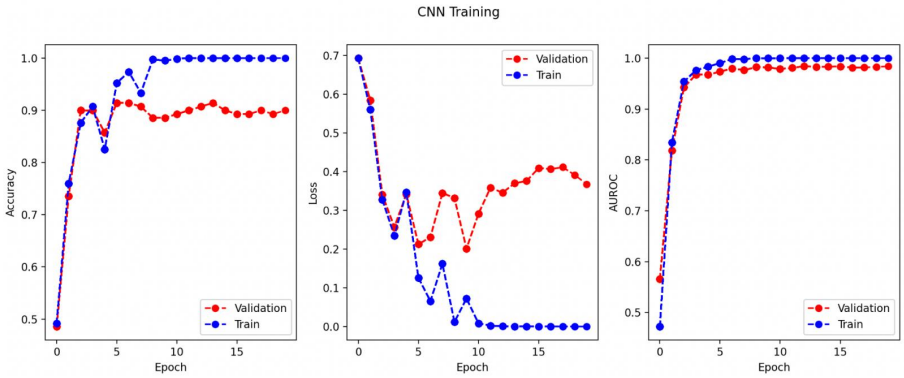
Validation AUROC:0.9816

Train Accuracy:0.9952

Train Loss:0.0727

Train AUROC:0.9998

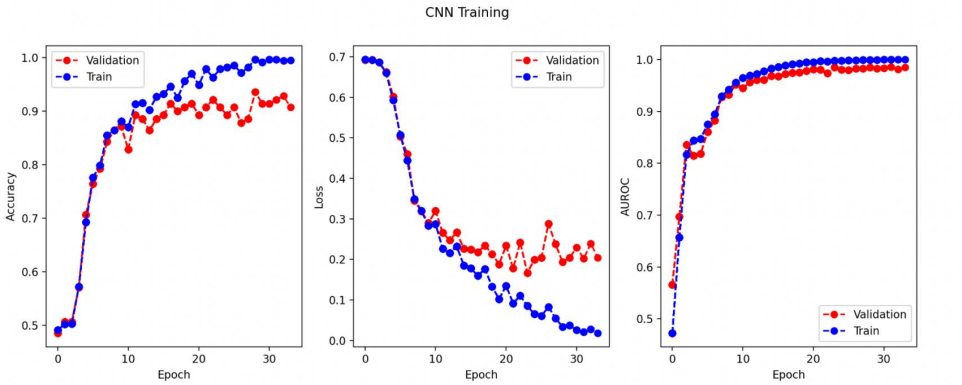
Question assigned to the following page: 6


CNN Training
Accuracy
Loss
AUROC
Epoch
Validation
Train

Question assigned to the following page: 6

lr=0.00001, weight_decay=0.0005
Epoch 23

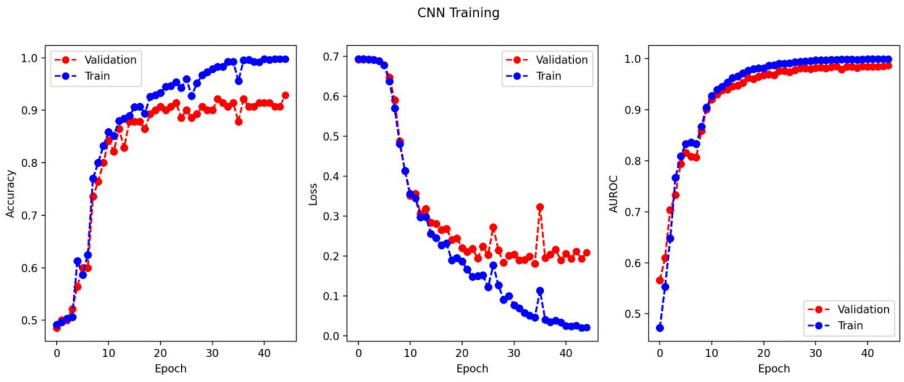
Validation Accuracy:0.9071
Validation Loss:0.1668
Validation AUROC:0.9845
Train Accuracy:0.9786
Train Loss:0.0853
Train AUROC:0.998



Now lets vary weight decay …

lr=0.00001, weight_decay=0.005
Epoch 34

Validation Accuracy:0.9143
Validation Loss:0.1812
Validation AUROC:0.9853
Train Accuracy:0.9929
Train Loss:0.046
Train AUROC:0.998

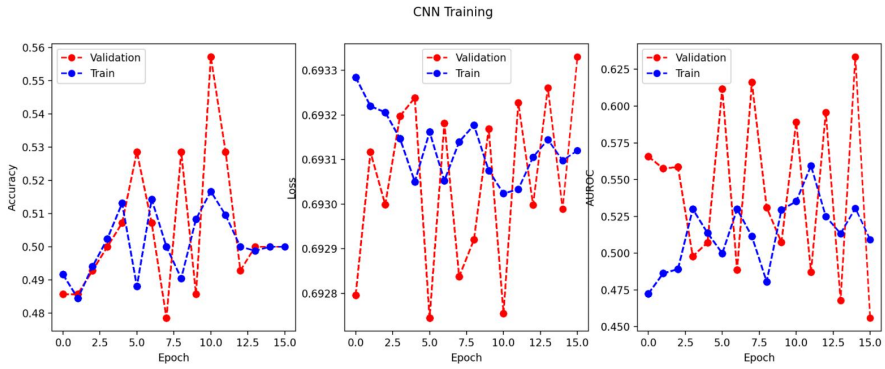


lr=0.00001, weight_decay=0.05

Question assigned to the following page: 6

Epoch 5

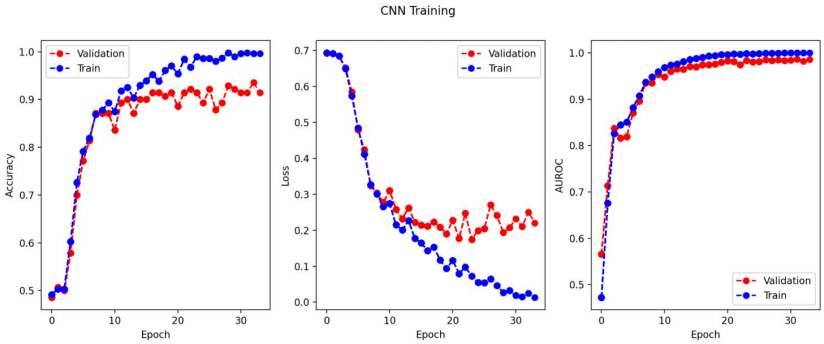
Validation Accuracy:0.5286
Validation Loss:0.6927
Validation AUROC:0.6116
Train Accuracy:0.4881
Train Loss:0.6932
Train AUROC:0.4999



The decay is definitely too high here based on the per epoch variance in performance …

lr=0.00001, weight_decay=0.00005
Epoch 23

Validation Accuracy:0.9143
Validation Loss:0.1744
Validation AUROC:0.9835
Train Accuracy:0.9893
Train Loss:0.0722
Train AUROC:0.9986



No questions assigned to the following page.

No questions assigned to the following page.

CODE APPENDIX

Question assigned to the following page: 7

Dataset.py

```python
"""
EECS 445 - Introduction to Machine Learning
Fall 2023 - Project 2

Landmarks Dataset
    Class wrapper for interfacing with the dataset of landmark 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.v3 import imread
from PIL import Image
from torch.utils.data import Dataset, DataLoader
from utils import config

import rng_control


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 LandmarksDataset 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 LandmarksDataset constructor.
    """
    tr = LandmarksDataset("train", task, **kwargs)
    ch = LandmarksDataset("challenge", task, **kwargs)

    standardizer = ImageStandardizer()
    standardizer.fit(tr.X)
    tr.X = standardizer.transform(tr.X)
    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 LandmarksDatasets and image standardizer.

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

    # Resize
    # 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]
    resized = np.array(resized)

    return resized
```

Question assigned to the following page: 7

```python
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), where N is
    the number of images in the set.
    """

    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.
        Hint: you may find the axis parameter helpful"""

        # Compute mean and standard deviation for each channel across all images
        # X has shape (N, image_height, image_width, color_channel)
        self.image_mean = np.mean(X, axis=(0, 1, 2), dtype=np.float64)
        self.image_std = np.std(X, axis=(0, 1, 2), dtype=np.float64)

    def transform(self, X):
        """Return standardized dataset given dataset X."""

        # Check if fit() was called before
        if self.image_mean is None or self.image_std is None:
            raise ValueError("Must call fit() before transform()")

        # Subtract the mean and divide by the standard deviation for each channel
        X_normalized = (X - self.image_mean) / self.image_std
        return X_normalized


class LandmarksDataset(Dataset):
    """Dataset class for landmark 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 numeric label 1 maps to the semantic label 'hofburg_imperial_pala
        """
        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=Fa
```

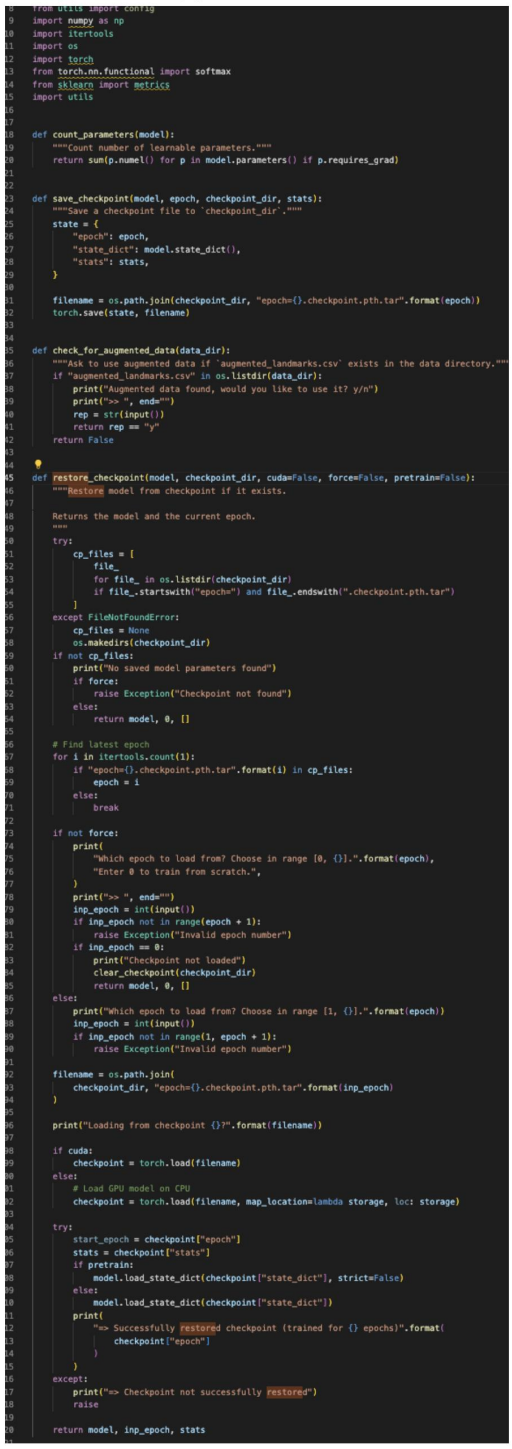
Question assigned to the following page: 7

Target.py

```python
"""
EECS 445 - Introduction to Machine Learning
Fall 2023 - 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):
        """
        Define the architecture, i.e. what layers our network contains.
        At the end of __init__() we call init_weights() to initialize all model parameters (weights and biases)
        in all layers to desired distributions.
        """
        super().__init__()

        # Layer 1: Convolutional Layer 1
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5, stride=2, padding=2)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Layer 3: Convolutional Layer 2
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=64, kernel_size=5, stride=2, padding=2)

        # Layer 5: Convolutional Layer 3
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=8, kernel_size=5, stride=2, padding=2)

        # Layer 6: Fully connected layer 1 (Output layer)
        self.fc_1 = nn.Linear(8 * 2 * 2, 2)

        self.init_weights()

    def init_weights(self):
        """Initialize all model parameters (weights and biases) in all layers to desired distributions"""
        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)

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

    def forward(self, x):
        """
        This function defines the forward propagation for a batch of input examples, by
        successively passing output of the previous layer as the input into the next layer (after applying
        activation functions), and returning the final output as a torch.Tensor object.
        """
        N, C, H, W = x.shape

        # Forward pass
        x = F.relu(self.conv1(x)) # Apply conv1 and ReLU
        x = self.pool(x) # Apply max pooling

        x = F.relu(self.conv2(x)) # Apply conv2 and ReLU
        x = self.pool(x) # Apply max pooling

        x = F.relu(self.conv3(x)) # Apply conv3 and ReLU

        x = x.view(N, -1) # Flatten the output tensor
        x = self.fc_1(x) # Apply fully connected layer

        return x
```

Question assigned to the following page: 7

Train_common.py 1/2



```python
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_landmarks.csv` exists in the data directory."""
    if "augmented_landmarks.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
```

Question assigned to the following page: 7

Train_common.py 2/2

```python
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):
    """
    Implement early stopping based on the validation loss.
    """
    # Extract the latest validation loss from stats
    latest_val_loss = stats[-1][1]

    # If the latest validation loss is greater or equal to the global minimum loss
    if latest_val_loss >= global_min_loss:
        curr_count_to_patience += 1
    else:
        curr_count_to_patience = 0
        global_min_loss = latest_val_loss

    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():
                import ipdb; ipdb.set_trace
                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(tr_loader, model, criterion, optimizer):
    """
    Train the model for one epoch.
    """
    model.train()  # Set the model to training mode

    for batch_idx, (data, target) in enumerate(tr_loader):
        optimizer.zero_grad()  # Zero out any gradient accumulation
        output = model(data)  # Forward pass
        loss = criterion(output, target)  # Compute loss
        loss.backward()  # Backward pass
        optimizer.step()  # Update model weights


def predictions(logits):
    """Determine predicted class index given a tensor of logits.

    Example: Given tensor([[0.2, -0.8], [-0.9, -3.1], [0.5, 2.3]]), return tensor([0, 0, 1])

    Returns:
        the predicted class output as a PyTorch Tensor
    """
    pred = torch.argmax(logits, dim=1)
    return pred
```

Question assigned to the following page: 7

Train_cnn.py

```python
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 rng_control


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()

    # Define loss function and optimizer. Replace "None" with the appropriate definitions.
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(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]

    # Define patience for early stopping. Replace "None" with the patience value.
    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=False,
        )

        # 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()
```

Question assigned to the following page: 7

source.py

```python
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__()

        # Define Convolutional layers
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5, stride=2, padding=2)  # padding set to 2 to achieve 'SAME' effect
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=64, kernel_size=5, stride=2, padding=2)  # padding set to 2 to achieve 'SAME' effect
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=8, kernel_size=5, stride=2, padding=2)  # padding set to 2 to achieve 'SAME' effect

        # Define Max Pooling layer
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        # Define Fully Connected layer
        self.fc1 = nn.Linear(8 * 2 * 2, 8)  # 8 channels, each of size 2x2

        self.init_weights()

    def init_weights(self):
        """Initialize all model parameters (weights and biases) in all layers to desired distributions"""

        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)

        # Initialize the parameters for self.fc1
        nn.init.normal_(self.fc1.weight, 0.0, 1 / sqrt(self.fc1.weight.size(1)))
        nn.init.constant_(self.fc1.bias, 0.0)

    def forward(self, x):
        # Apply the first convolutional layer
        bsz, _, _, _ = x.size()

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

        # Apply the second convolutional layer
        x = self.conv2(x)
        x = F.relu(x)
        x = self.pool(x)

        # Apply the third convolutional layer
        x = self.conv3(x)
        x = F.relu(x)

        # Flatten the tensor
        x = x.view(bsz, -1) # 64 x 32

        # Apply the fully connected layer
        x = self.fc1(x)


        return x

```

Question assigned to the following page: 7

Train_source.py

train_source.py > ...

```python
    # Model
    model = Source()

    # Criterion (loss function)
    criterion = nn.CrossEntropyLoss()

    # Optimizer
    optimizer = optim.Adam(model.parameters(), lr=1e-3, 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")
    #import ipdb; ipdb.set_trace()

    # 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]

    # 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()
```

Question assigned to the following page: 7

Train_target.py

```python
"""
EECS 445 - Introduction to Machine Learning
Fall 2023 - 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
"""

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

import rng_control

def freeze_layers(model, num_layers=0):
    """Stop tracking gradients on selected layers."""
    layers = model.named_parameters()
    for _ in range(num_layers*2):
        layer_pointer = next(layers)
        layer_pointer[1].requires_grad_(False)




def train(tr_loader, va_loader, te_loader, model, model_name, num_layers=0):
    """Train transfer learning model."""
    # TODO: Define loss function and optimizer. Replace "None" with the appropriate definitions.
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(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: Define patience for early stopping. Replace "None" with the patience value.
    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()
```

Question assigned to the following page: 7

Augment_data.py

augment_data.py > ...

```python
import argparse
import csv
import glob
import os
import sys
import numpy as np
from scipy.ndimage import rotate
from imageio.v3 import imread, imwrite

import rng_control


def Rotate(deg=20):
    """Return function to rotate image."""

    def _rotate(img):
        """Rotate a random integer amount in the range (-deg, deg) (inclusive).

        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
        """
        # Generate a random integer degree in the range (-deg, deg)
        rotation_deg = np.random.randint(-deg, deg + 1)

        # Rotate the image and fill the missing pixels with black
        rotated_img = rotate(img, rotation_deg, reshape=False, mode='constant', cval=0)

        return rotated_img

    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
        """
        # Compute the grayscale values
        grayscale_values = np.mean(img, axis=-1).astype(np.uint8)

        # Convert it back to a 3-channel image
        grayscale_img = np.stack([grayscale_values] * 3, axis=-1)

        return grayscale_img

    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
    :original: whether to include the original images in the augmented dataset or not
    :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_landmarks.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 = [Rotate()] # rotation only
    # augmentations = [Grayscale()] #grascale only
    # augmentations = [Rotate(), Grayscale()] #both rotation and 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=True,  # 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:]))
```

-- INSERT --

Question assigned to the following page: 7

Challenge.py

model > challenge.py > Challenge > __init__ > num_classes

```python
"""
EECS 445 - Introduction to Machine Learning
Fall 2023 - 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, num_classes=2):
        super(Challenge, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((2, 2))
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 2 * 2, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x

```