1. Data Processing

(a)

i

Mean: [118.263, 118.004, 118.594]

Std: [66.307, 69.039, 75.332]

ii.

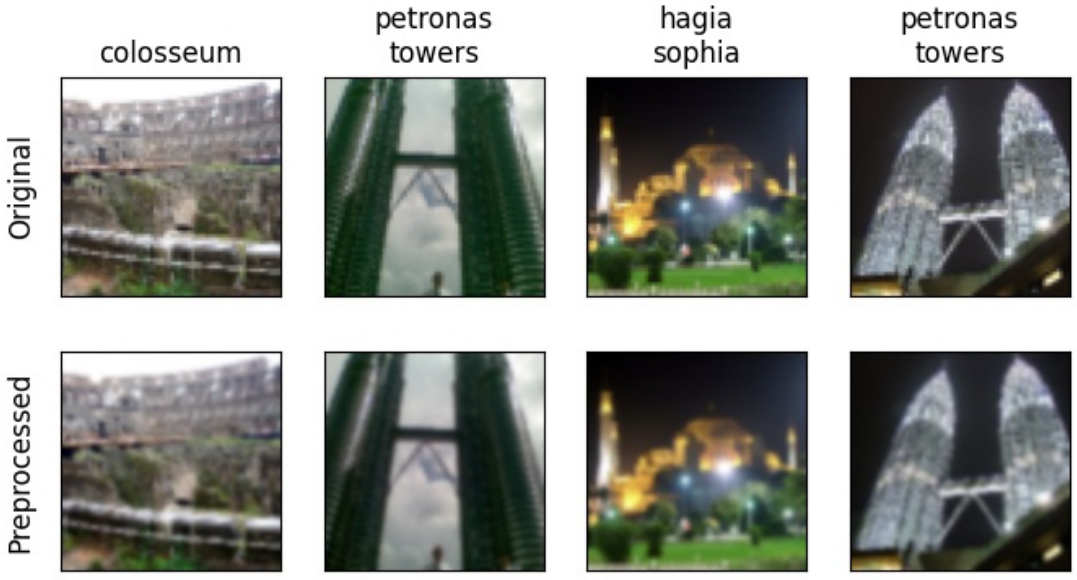
① By extracting the mean and standard deviation from each color channel, we can normalize each channel of the image. and thus reducing the possibility of overfitting.

Other data partitions are uncommon methods, and are useless for normalizing the image.

(2) processing the image with their mean and std is a linear process (shift the data by the value of mean and then scale it).

Therefore, the image still contains the original information. There's no information loss by processing the image with mean and standard deviation. And this will be beneficial for the model to take outliers into account. After normalization, outliers won't have great impact on the predicting result.

(b)



2. Convolutional Neural Networks

a.

Layer 1: 16 filters

each filter size 5×5×3

bias: 1

⇒ 16 × (5×5×3+1) = 16 × (76) = 1216

Layer 2: 0

Layer 3: 64 filters

filter size: 5×5×16

bias: 1

⇒ 64 × (5×5×16+1) = 25664

Layer 4: 0

Layer 5: 8 filters.

filter size: 5×5×64

bias 1

⇒ 8 × (5×5×64+1) = 12808

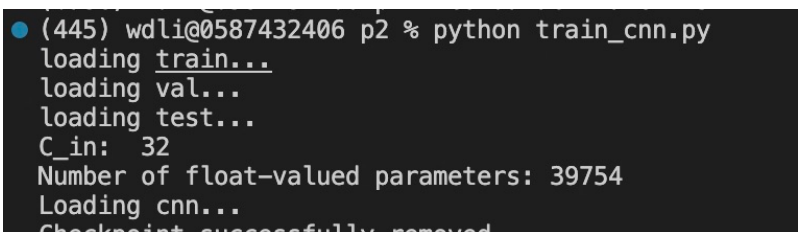
Layer 6: 1 bias: 32 inputs, 2 outputs

(32+1) ×2 = 66

In total: 1216 + 25664 + 12808 + 66

= 39754

The photo to the right verifies this conclusion.



(f)

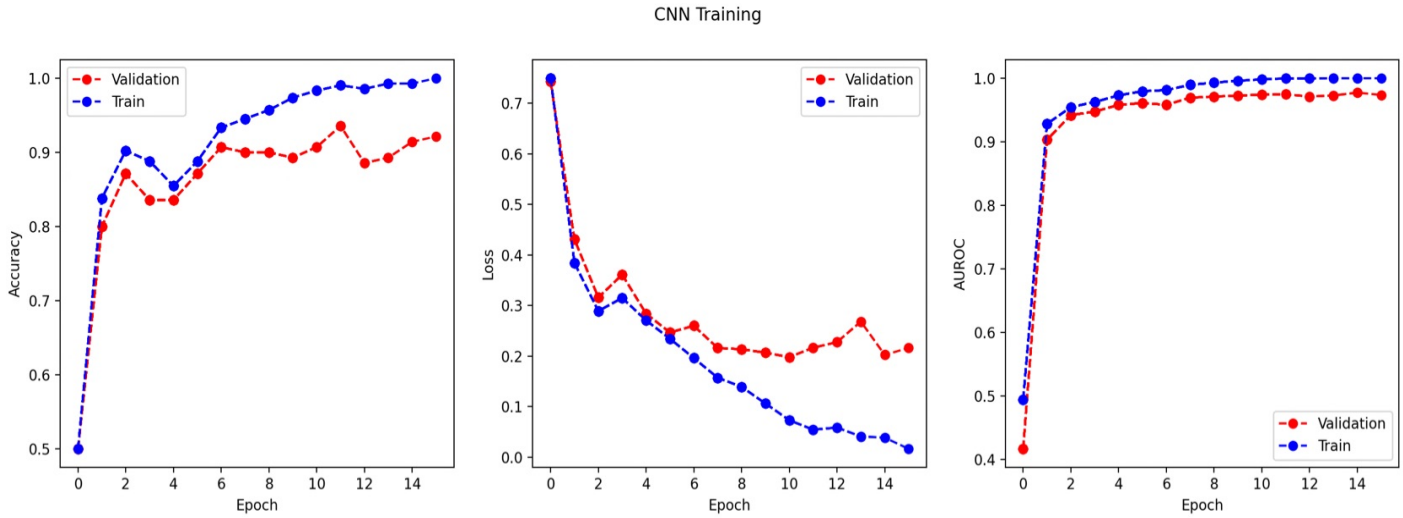
(i)

1. The reason why it doesn't monotonically decreasing is that the model may ~~be~~ overfit when epoch is high, thus validation error doesn't always decrease when epoch increases

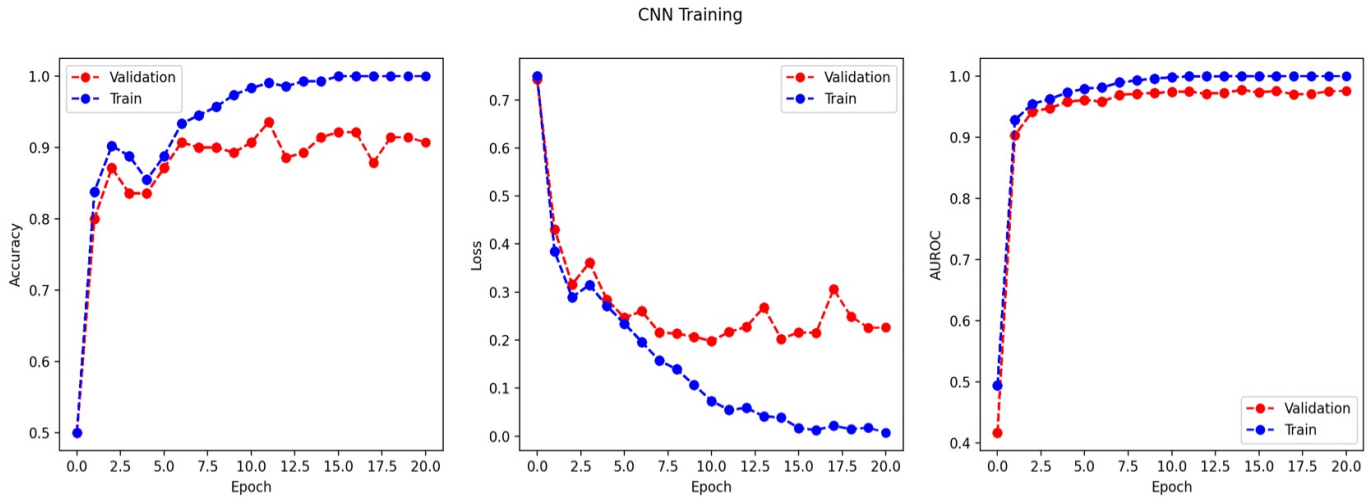
2. the dataset may contain some noise. when we apply mini-batch SGD to it, we may receive some random information. this will result in oscillation in model prediction.

(ii) when patience = 5 model stop training at 15

when patience = 10, model stop training at 20



The above figure is generated by using a patience of 5



The above figure is generated by using a patience of 10

Based on the training graph, which patience value works better and why?

Patience = 5 works better

Compared to loss value when patience = 10, the model that is generated by patience = 5 is less overfitting. And the value when validation loss equals training loss is less than that of model of patience 10.

And since both graph overfits when epoch is high, this shows it's not necessary to use patience=10 since the extra work is meaningless (e.g. no performance increase).

Both of these two model show quite similar validation performance in terms of accuracy, loss and AUROC, but the model of patience=5 needs less time to training(and patience=10 may aggravate overfitting). So, in terms of efficiency, model with patience = 5 is better.

In what scenario might increased patience be better?

When the model fluctuates quite intensely, and the value of the model may suddenly drop a lot due to the instability of the model. For example, when the model comes across a local minima, if the patience is not large enough, the figure will stop, leaving high error because the model doesn't reach global minimum. If the patience is higher, it will keep pending in the calculation for a longer time, which means it will be more possible to reach global minimum.

(iii): The new size is 64 * 2 * 2 = 256

	Epoch	Training AUROC	Validation AUROC
8 filters	10	0.9982	0.9743
64 filters	7	0.9993	0.9694

How does the performance vary as we increase the number of filters from 8 to 64? What might account for this change?

The validation performance decreases a little bit, but the training performance of 64 filters is better than that of 8 filters in terms of AUROC. When the number of filters is 64, we reach minimum loss when Epoch=7, but we reach minimum loss with Epoch=10 when using 8 filters. Although generally, the accuracy, loss and AUROC of both models are quite similar, the model with 64 filters always get lower maximum performance in terms of validation.

This might because of

1. the overfitting. Since the dataset has some noise. 64-sized filter lets the model receive too much “noise”, this may lead to overfitting.
2. too complex model. 64-sized model might be too complex structure for this dataset, so the model might overfit the data.

(g)

(i)

	Training	Validation	Testing
Accuracy	0.9833	0.9071	0.625
AUROC	0.9982	0.9743	0.6827

(ii)

Yes.
From the comparison between training performance and validation performance, we can notice that the training accuracy is much greater than validation accuracy (higher for about 0.075) and the AUROC of training model is also greater than that of validation model (for about 0.01). Although the AUROC score does not show far exceeding, the accuracy score of training and validation model changes much.

(iii)

Compare the validation and test performance.
From the table, we can notice that the testing performance is much lower than validation performance. They are not very similar.

What trend do you see here?
It seems the test performance is much lower than validation performance. And validation performance is lower than training performance. So, in terms of performance(accuracy & AUROC), from higher to lower: Training > Validation > Testing.

Hypothesize a possible explanation for such a trend.
This might due to the bias learned from the training dataset. The performance of the model shows that the training performance is slightly greater than validation performance. However, the training performance and validation performance is much greater than testing performance. This shows that our model learned some not important features from the training set that are not generalizable on testing set.

3. Visualizing what the CNN has learned

(a).

$$L^1 = ReLU\left(\sum_{k=1}^{2} \alpha_k^1 A_k^1\right)$$

$$= ReLU\left[\sum_{k=1}^{2} \left(\frac{1}{Z}\sum_i\sum_j \frac{\partial y^1}{\partial A_{ij}^k}\right) \cdot A_k^1\right]$$

$$\alpha_1^1 = \frac{1}{Z}\sum_i\sum_j \frac{\partial y^1}{\partial A_{ij}^1} = \frac{1}{16}\cdot(-1+1-2-1-1+1+1+1+2+2)$$

$$= \frac{3}{16}$$

$$\alpha_2^1 = \frac{1}{Z}\sum_i\sum_j \frac{\partial y^1}{\partial A_{ij}^2} = \frac{1}{16}\cdot(1+2+2+2+2+1+1-1-2-1)$$

$$= \frac{7}{16}$$

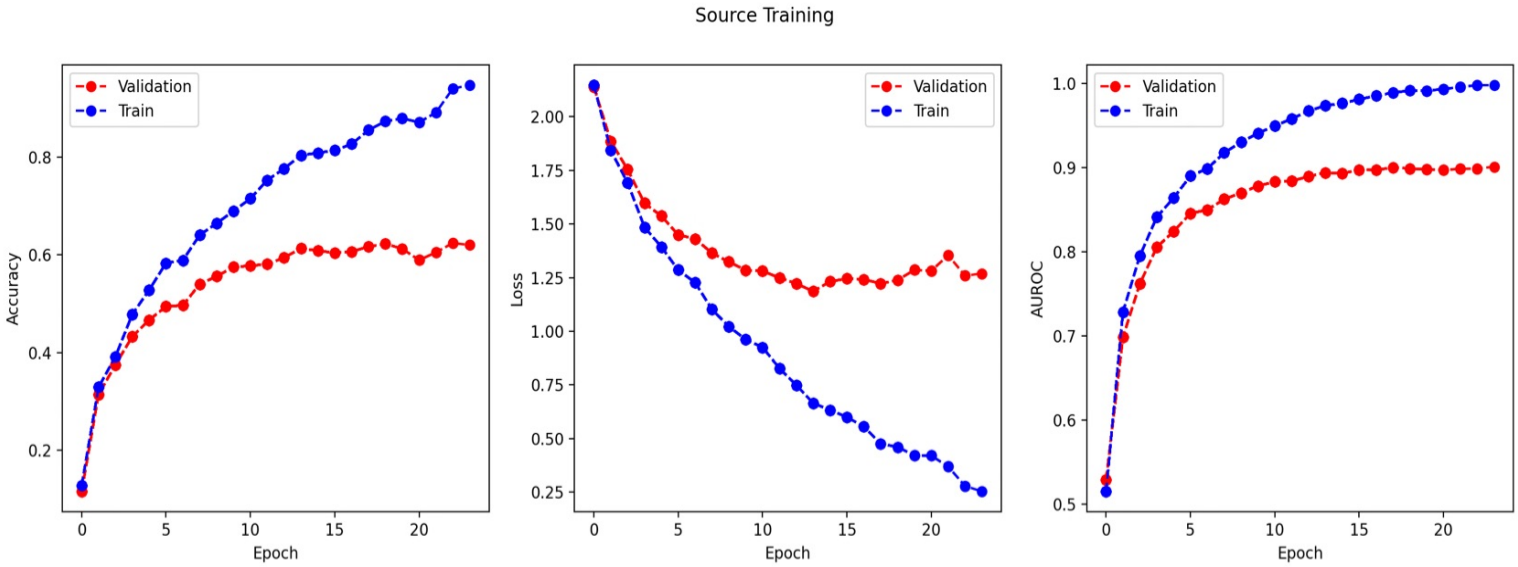
$$L^1 = ReLU\left(\sum_{k=1}^{2} \alpha_k^1 A^k\right) = \alpha_1^1 A^1 + \alpha_2^1 A^2$$

$$= \begin{pmatrix} \frac{5}{8} & \frac{5}{8} & \frac{13}{16} & \frac{5}{8} \\ \frac{17}{16} & \frac{5}{4} & \frac{17}{16} & \frac{7}{8} \\ \frac{7}{8} & \frac{17}{16} & \frac{7}{16} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

(b)
From the problem, we are told that the lighter colored area of the image is more important for the model to distinguish. Because these lighter areas are generally distributed on blue objects like sky, blue clothes. Also, some lighter area is also distributed on places that transited from one object to another, like roof, column, and window of the building.

(c)
Yes.
From the picture, I notice that the lighter part does not have a very precise distribution on these buildings, most of lighter area even distributed on objects like sky and humans. Considering that the model has not bad validation performance, I can confirm that the model learned some bias that is not important (non-generalizable) features from training set. The blue objects can be considered as bias.

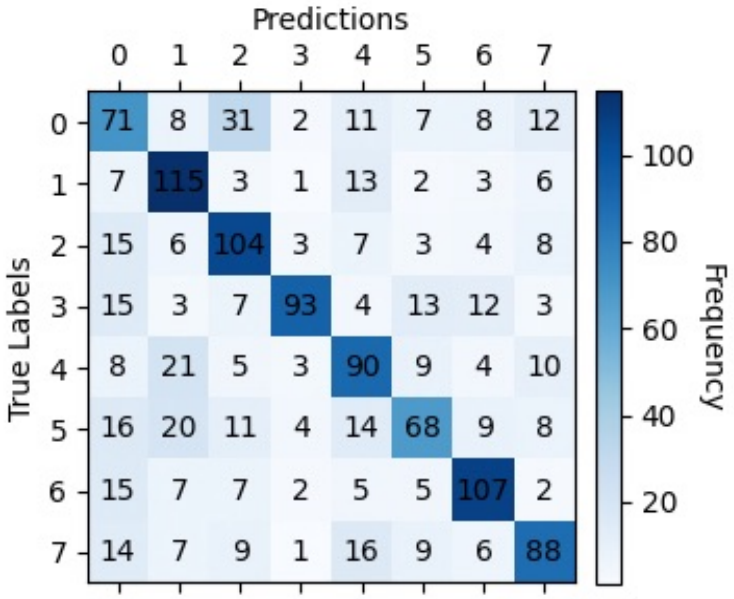
4.1 c



Epoch Number corresponding to the lowest validation loss: 13

4.1 d

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



1-Petronas Towers is predicted the most accurate, while 5-Berlin Cathedral is predicted the least accurate. I get the conclusion from the above graph. The formula for accuracy is

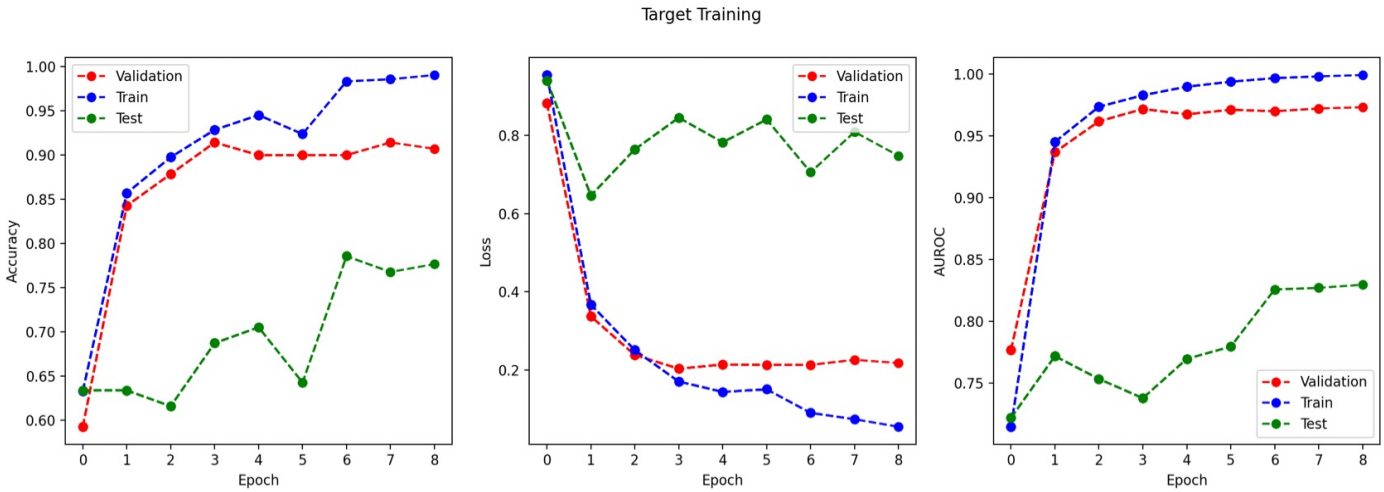
- Accuracy = True Predictions / All Predictions

Since for all predictions, the denominator is equal. To compare the accuracy of each landmark, we simply need to compare the number of true labels. From the graph, we know that the number on the diagonal means true predictions. Among them, 1-Petronas Towers has highest true prediction number while 5-Berlin Cathedral has lowest true prediction number.

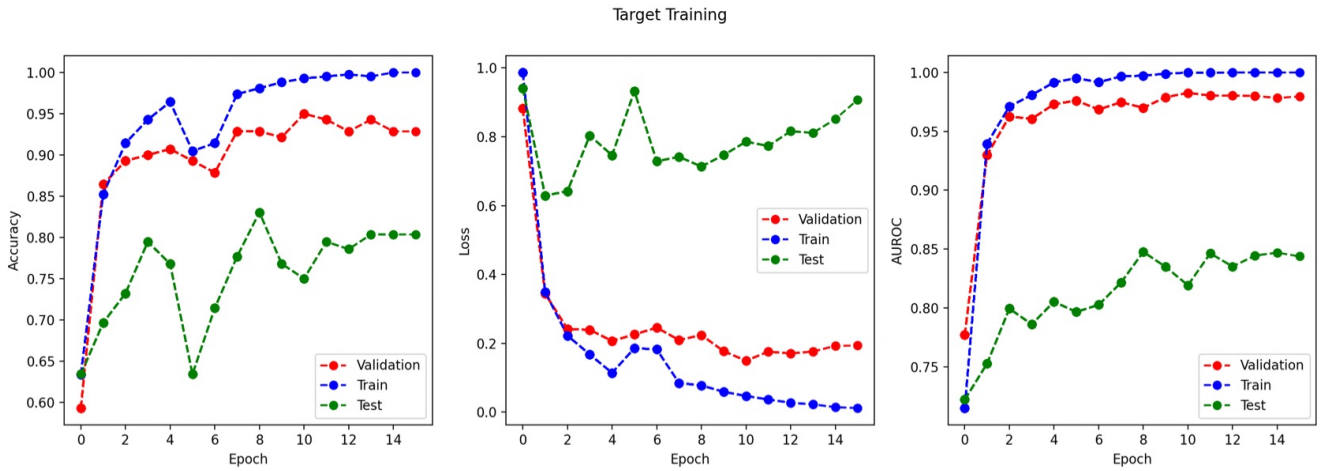
For the reason why this might be the case, I think (1) maybe Berlin Cathedral is not distinct from its background. Its roof is blue, which is quite close to the color of the sky. However the Petronas Towers has some distinct features like two identical buildings and one bridge in between. This distinguish itself from other buildings.(2) Maybe the training data for Berlin cathedral is not quite similar to that in the testing data set, so the data is not quite representative. So the model may learn some features that is not quite important in testing set.

4.1 f

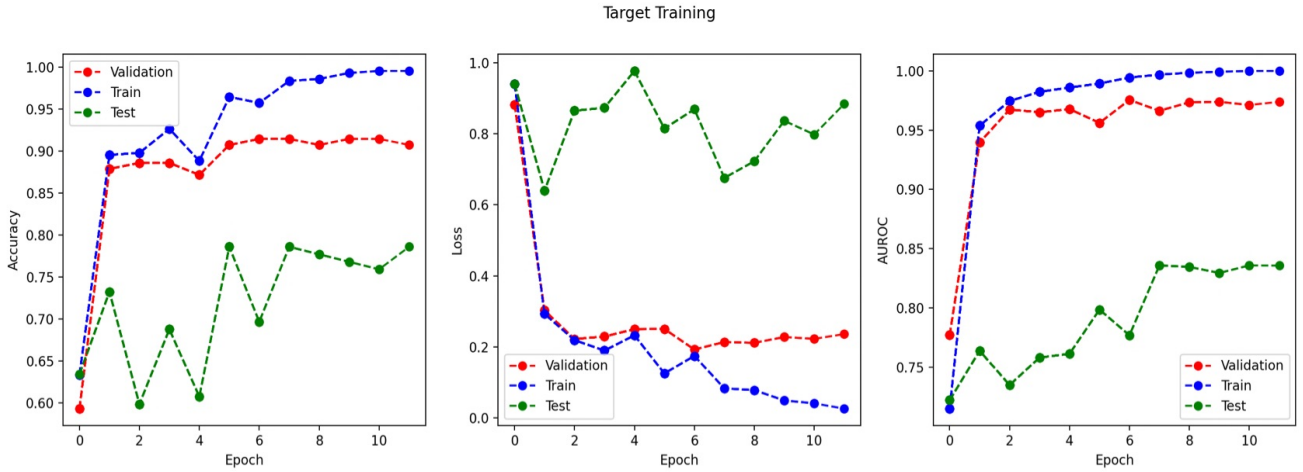
	AUROC		
	TRAIN	**VAL**	**TEST**
Freeze all CONV layers (Fine-tune FC layer)	0.9985	0.9747	0.8383
Freeze first two CONV layers (Fine-tune last CONV and FC layers)	0.9944	0.9755	0.7768
Freeze first CONV layers (Fine-tune last 2 conv, and fc layers)	0.9999	0.9827	0.8192
Freeze no layers (Fine-tune all layers)	0.983	0.9718	0.7379
No pertaining or Transfer Learning(Section 2(g) performance)	0.9982	0.9743	0.6827



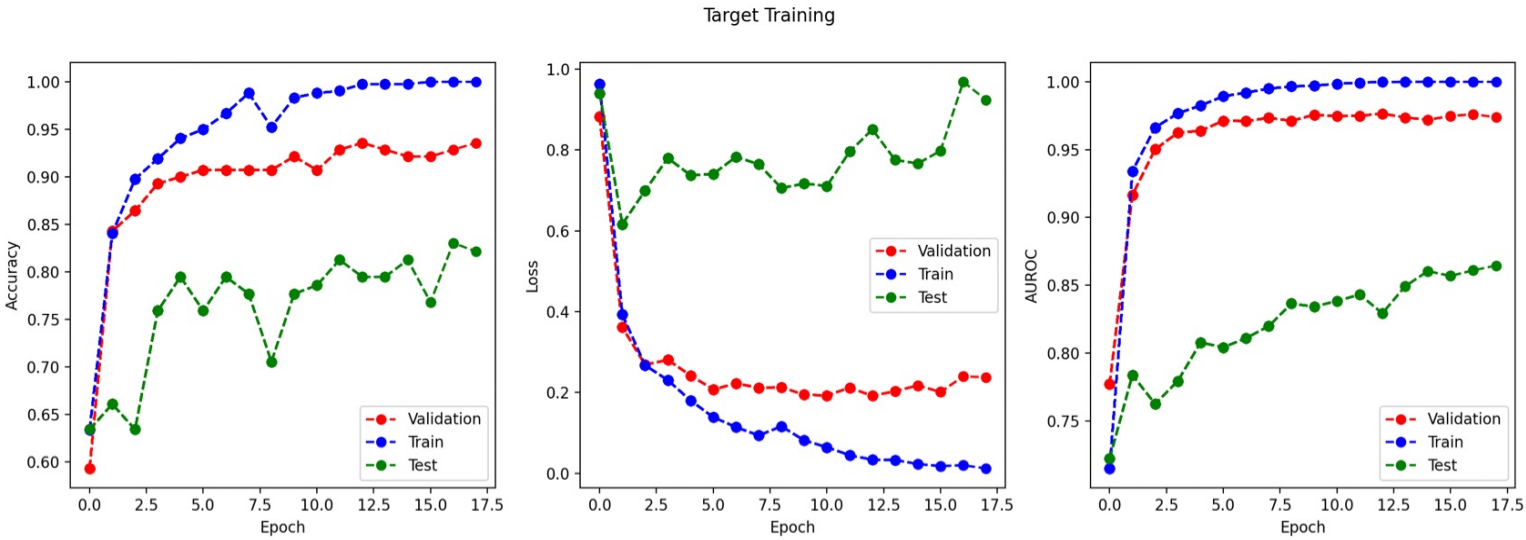
Freeze no layers



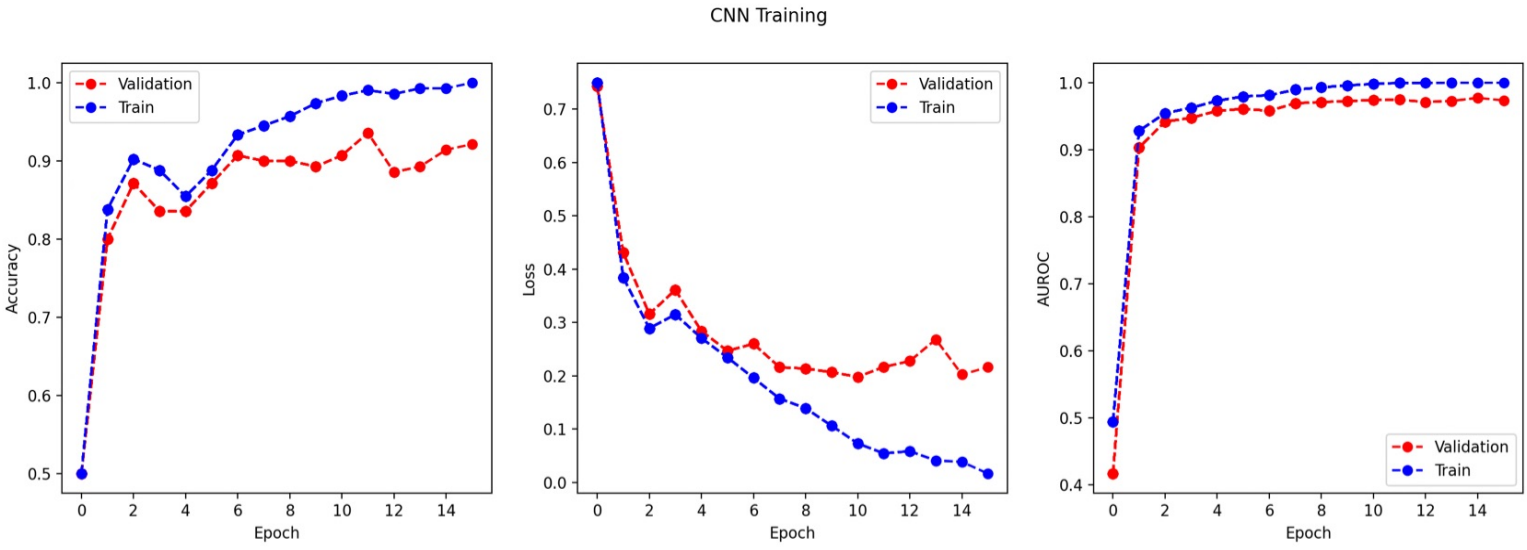
Freeze one layer



Freeze two layers



Freeze three layers



No pertaining or Transfer Learning(section 2(g))

Examine the results of using transfer learning

The performance is better than before. From the table, we can notice that the testing AUROC increases compared to the model that freezes no layers. And we can notice that testing AUROC increases when more layers are frozen (despite the case when first two layers are frozen). And all results of the first four rows are better than the last row, which uses the most primitive method and not using pertaining or transfer learning.

Does transfer learning help? Was the source task helpful? Why do you hypothesize it was (or wasn't) ?

From the table, we know that all first four rows have greater testing AUROC than the last row, which corresponds to the model that does not using transfer learning and source task. Although all four models have similar training and validation performance, the last model has worst testing performance, and this means transfer learning and source task are helpful.

How does freezing all convolutional layers compared to freezing just a subset, versus freezing none?

From the table, we find that the testing AUROC of the model that freezes all convolutional layers has best performance. This indicates freezing all convolutional layers achieves better performance than models that simply freeze a subset or nothing. And we can also compare the models that freeze a subset with the model that freezes nothing. The comparison shows that models that freeze just a subset still achieve better AUROC score than model that freezes nothing.

Why do you think this might occur?

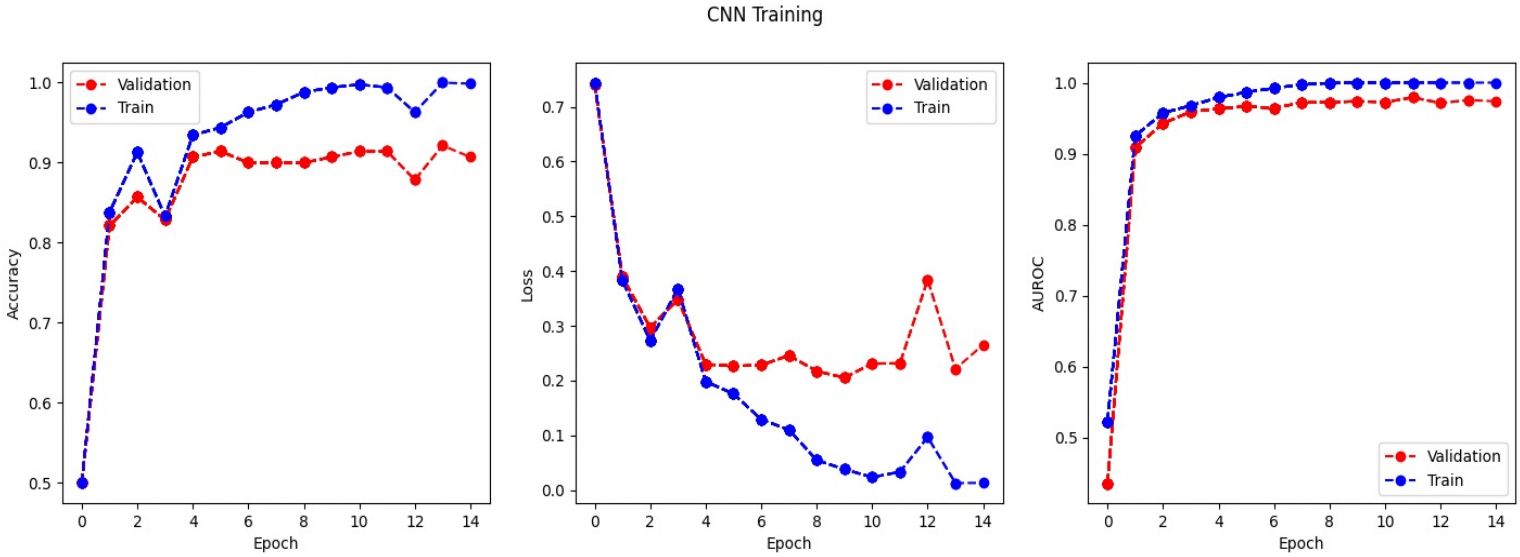
Because the source model is trained from the dataset that contains all 8 layers, the parameters in its convolutional layers might contain less bias than the model that trained on dataset that only has images of Pantheon and Homburg Imperial Palace. The more frozen convolutional layers, the more uesful and generalizable weights contained by the model. So, in the table, we can notice that for the testing AUROC, models that froze more layers have better performance when classifying data. In addition, if we freeze more layers, the model may contain fewer learnable parameters. This will reduce model complexity and can help solve overfitting to some extent.

However, we notice that the training error and validation error of the model whose convolutional was frozen is lower than that of the primitive model. This is because the primitive model tends to receive more information from training data and validation data and thus contains some bias in these two data sets.

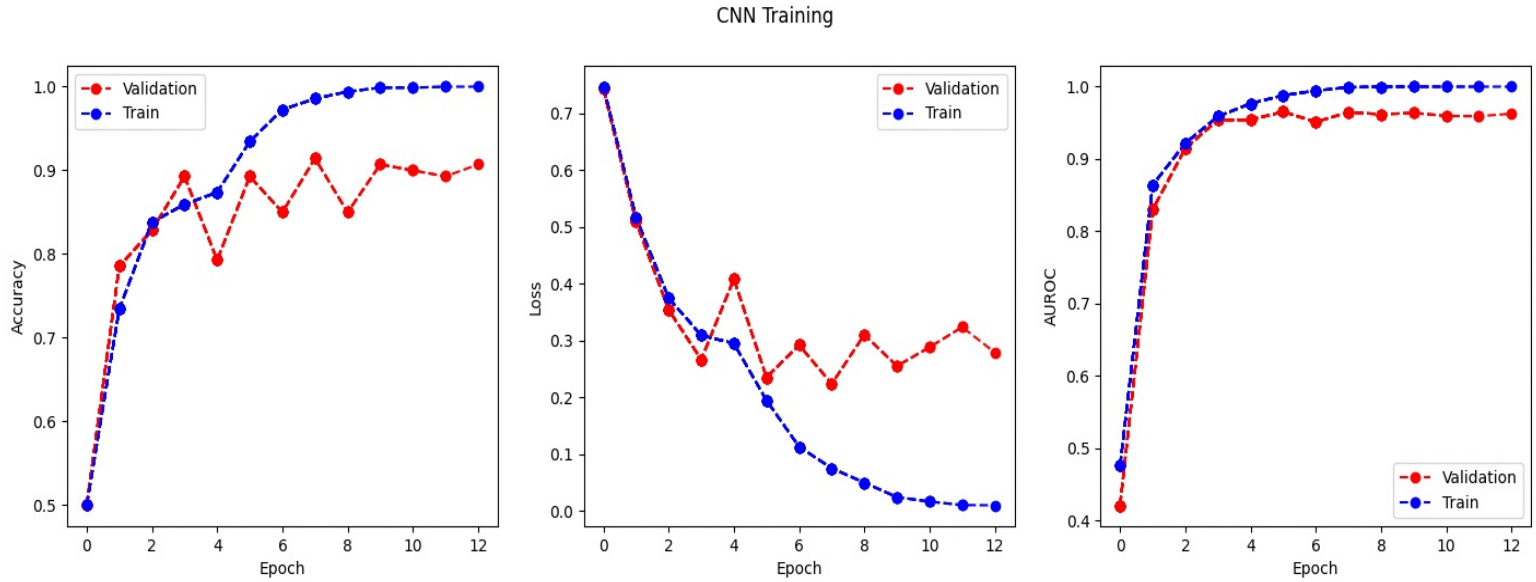
4.2 b

	AUROC		
	TRAIN	VAL	TEST
Rotation (keep original)	0.9998	0.9749	0.7602
Grayscale (keep original)	0.9994	0.9663	0.7618
Grayscale (discard original)	0.9946	0.911	0.8042
No augmentation	0.9982	0.9743	0.6827

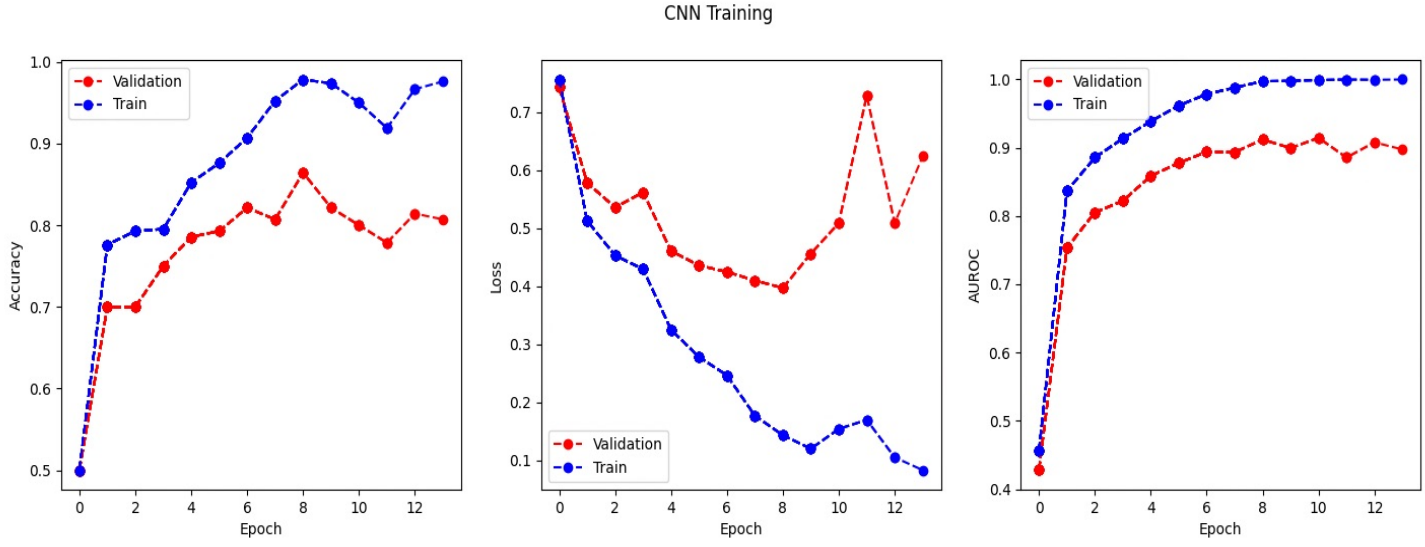
Rotation, keep original:



Grayscale, keep original:



Grayscale, discard original:



4.2 c

Compared to model that does not use any augmentation, The validation and training performances of the models that use rotation and grayscale are generally lower. For the training score, rotation, grayscale(keep original) has higher score than no augmentation. However, the model that uses grayscale but discards original images gets lowest training and validation score but highest testing score.

Maybe this is because the rotation and grayscale makes the model receive more information, instead of some misleading features like blue sky. Because of this, the bias in the training set is lower than the original training data set. The model receive less bias, so the model can detect more generalizable features. Therefore, the validation error is lower than the model that learned from the not-augmented dataset, but the testing error is higher than the primitive model.

Challenge

In this project, I build a convolutional neural network based on residual neural network 18 from scratch, and applied several methods including data augmentation, regularization, feature selection and transfer learning to test on models and improve the performance. Finally, I choose to use ResNet18, Epoch=56, trained on augmented data, regularization, and transfer learning. The maximum validation AUROC is 0.9826, greatest testing AUROC is 0.9917

Model Evaluation

Since this challenge part only grades AUROC score of our predicted model, I choose AUROC as a main criteria of model evaluation. But it does not mean I do not see loss and accuracy in the whole process. In the later part, whenever there is a experiment on certain methods, I will show the result by presenting an image comprised of three criteria: accuracy, loss and AUROC, and report the maximum AUROC score explicitly.

Model architecture

From the slides, it's shown that ResNet achieved best performance in 2014 ImageNet Challenge. Therefore, I think this is a good model for me to reference.

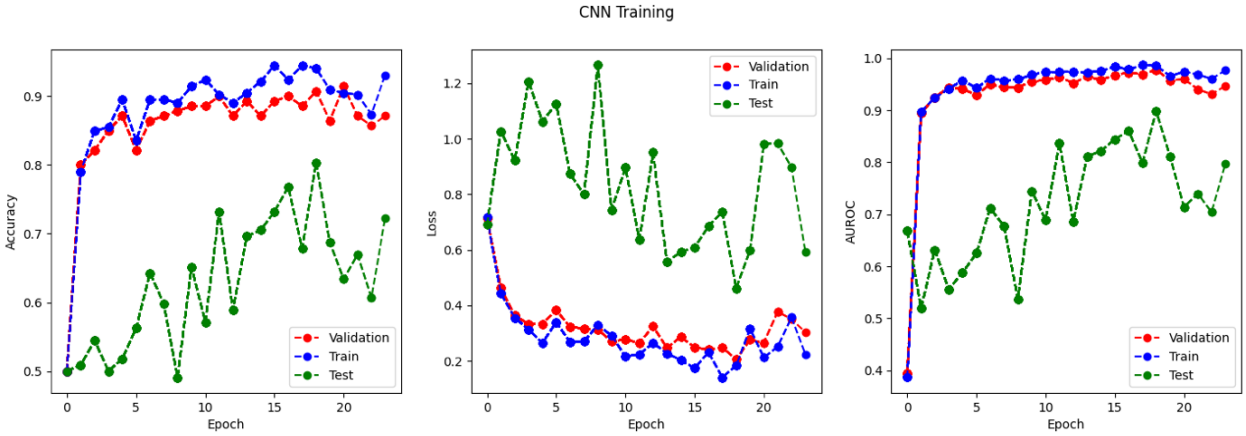
In the `challenge.py` , I implemented a model based on ResNet architecture. It has 18 layers in total, while each layer was normally distributed with $\mu = 0.0$, $\sigma = \frac{1}{\sqrt{5 \times 5 \times num\_input\_channels}}$.

In the transition of convolutional layers and linear layers, I used an average pooling layer. This can let me avoid calculating the image dimension changes in convolutional layers.

The following image shows the result of my first training and validation.

- Criterion: cross entropy loss
- Optimizer: adam
- Learning rate: 0.001

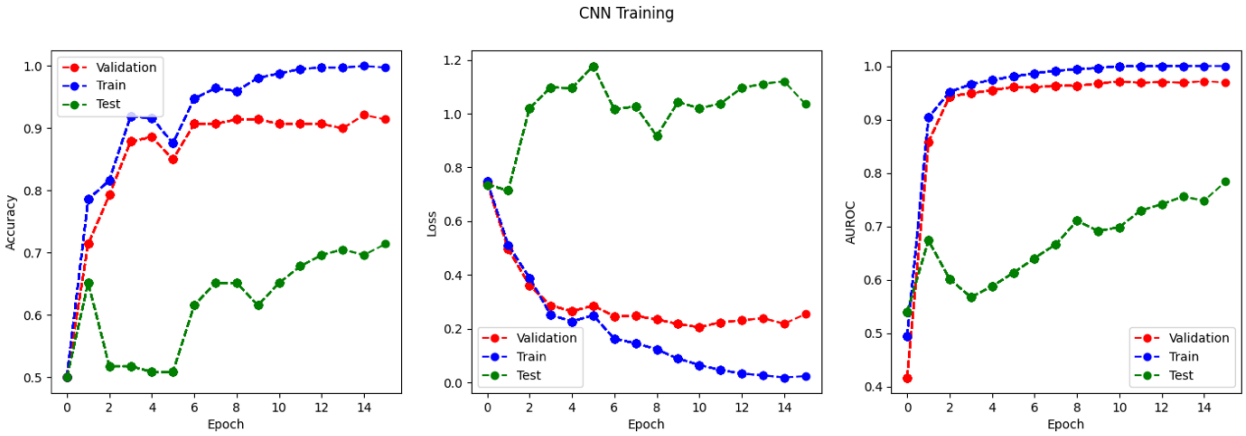
- Patience: 5
- Batch size: 32



From the figure, we know that

- the greatest validation AUROC is 0.9535, epoch 7
- the lowest validation error is 0.3132, epoch 2
  This is not a good result, so we need to make more improvements.

For comparison, I also implemented the original model that we used in the previous part.
The result is as follows:



We can figure out that the ResNet18 has better performance, but it needs a lot of time to train. In the later sections, I will focus on the original model, but when some methods has improvements, I will apply them into resNet.

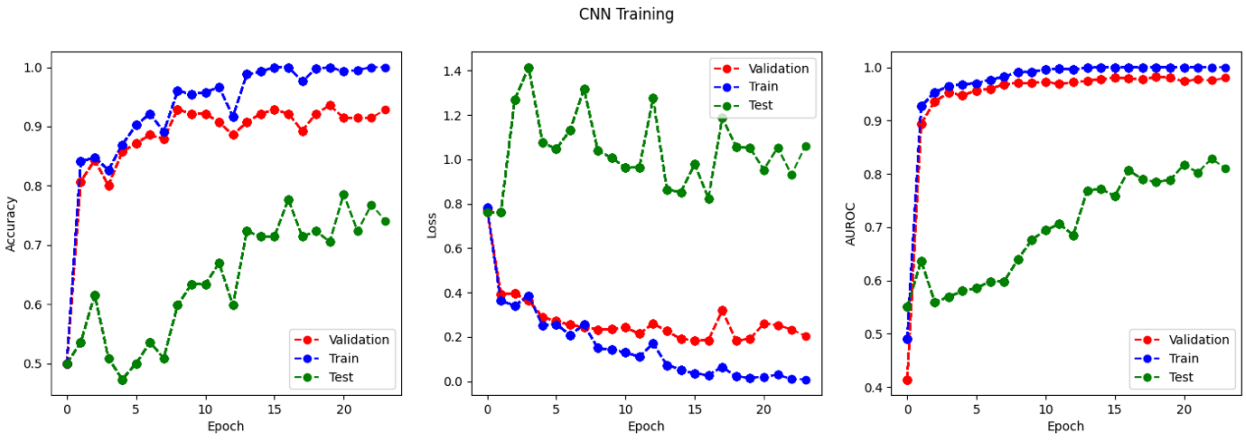
Regularization (Weight decay, dropout, etc.)

In this section, I will apply some regularization methods. Regularization can be known as a way to adding additional constraints to the optimization problem in order to prevent overfitting. From the above raw test, we can intuitively sense that the model receives lots of noises in the process, and the model overfit when epoch is very big. So, it's reasonable to apply regularization in mitigate the effect of overfitting.

Dropout

First, we can employ dropout. I first by applying each dropout after each convolutional layers.

And the result is as follows:



The maximum AUROC is 0.8281, this is not a good performance, so we need to continue improve it.

However, since our original model has better performance than the most original model. we will keep this methods.

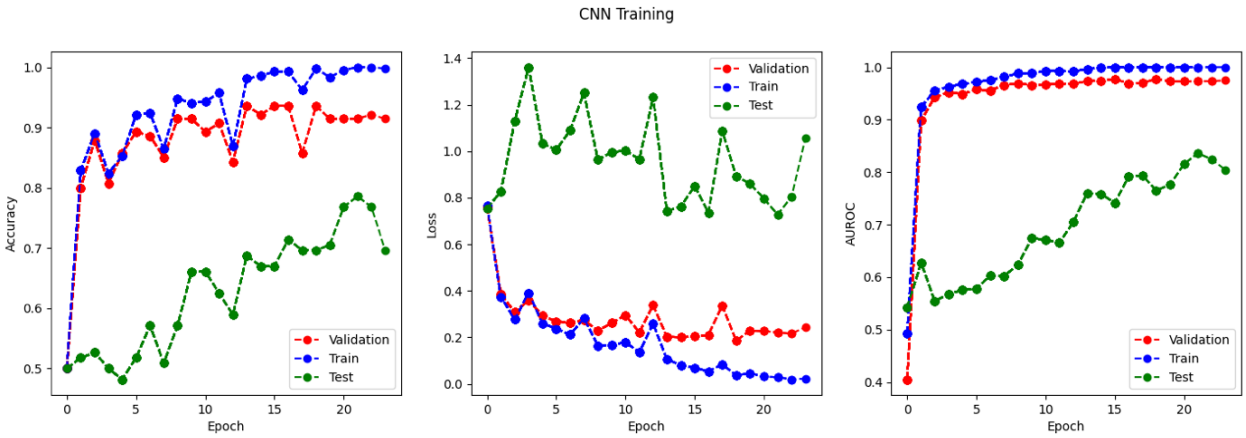
Weight Decay

As shown in the following code, we can add weight decay to mitigate the effect of overfitting.

```
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3, weight_decay=0.01)
```

This principle behind it is quite like what we use L2 norm regularization in support vector machine, and the value we input means the importance of this L2 norm.

After applying this method, we can get the following result.

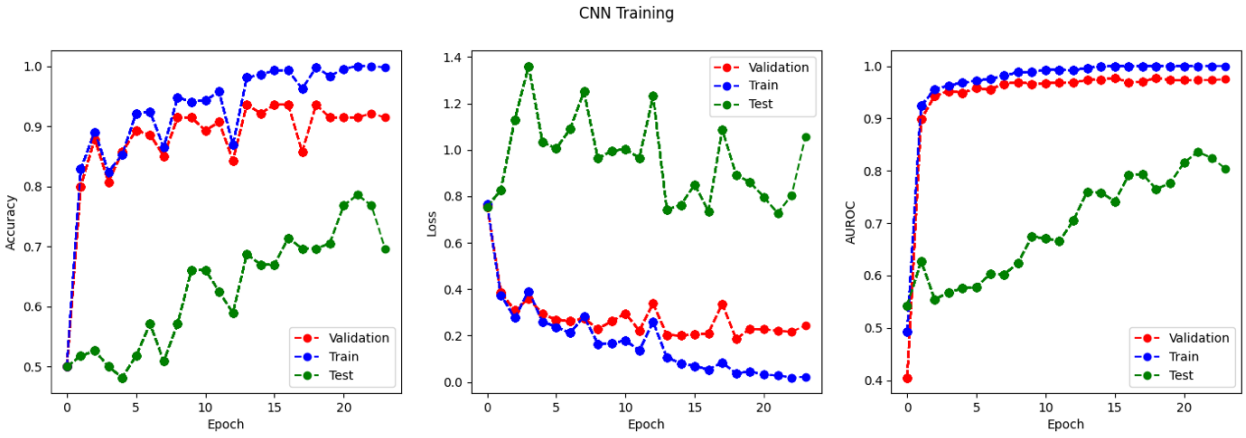


The maximum test AUROC is 0.8361, which is better than last trial. So I decide to keep it.

Batch Normalization

Given the improvements brought by the above two methods, we are sure that regularization can indeed help the model to increase performance.

So, now I decided to apply batchNormal to this model. Batch Normalization normalizes the inputs of each layer in a mini-batch, which helps stabilize and speed up the training process. Here's the performance after I implement the Batch Normalization to the model.



The maximum AUROC is 0.9021! It's a great improvement!

After implementing dropout, weight decay and batch normalization, the final code is shown as below.

```
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=(5,5), stride=(2,2), padding=2)
        self.bn1 = nn.BatchNorm2d(16)
        self.conv2 = nn.Conv2d(16, 64, kernel_size=(5,5), stride=(2,2), padding=2)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 8, kernel_size=(5,5), stride=(2,2), padding=2)
        self.bn3 = nn.BatchNorm2d(8)
        self.dropout1 = nn.Dropout(p=0.1)
        self.dropout2 = nn.Dropout(p=0.2)
        self.dropout3 = nn.Dropout(p=0.3)
        self.pool = nn.MaxPool2d(kernel_size=(2,2), stride=(2,2))
        self.fc1 = nn.Linear(32, 2)
        self.init_weights()
    def forward(self, x):
        N, C, H, W = x.shape

        x = F.relu(self.bn1(self.conv1(x)))
        x = self.dropout1(x)
        x = self.pool(x)
        x = F.relu(self.bn2(self.conv2(x)))
        x = self.dropout2(x)
        x = self.pool(x)
        x = F.relu(self.bn3(self.conv3(x)))
        x = x.reshape(N, 32)
        x = self.fc1(x)

        return x
```

And we can apply some similar strategies to ResNet18 model.

Transfer Learning

The transfer learning is a good way to let our current model receive more information. Since our original dataset has 8 classes in total, we can train the model on 8 classes first, then transfer the learned parameters and freeze them when training new models on dataset of 2 classes. In this way,

we can take advantages of the previous trained model and get better performance.

Since we've already known the power of transfer learning in previous problems, in this part, I will only apply the code on the ResNet18 model.

From section 1, I've already presented my handwritten code of ResNet18. In order to do transfer learning on this model, I need to set the `num_classes` equal to 8 first, and then store the code into `p2/model/challenge_source.py`. And I also add `train_challenge_source.py` to train `challenge_source.py` directly. Finally, in the `train_challenge.py`, I add an `if` statement, asking if user would like to use transfer learning:

```
do_transfer_learning = input("use transfer learning?y/n\n")
```

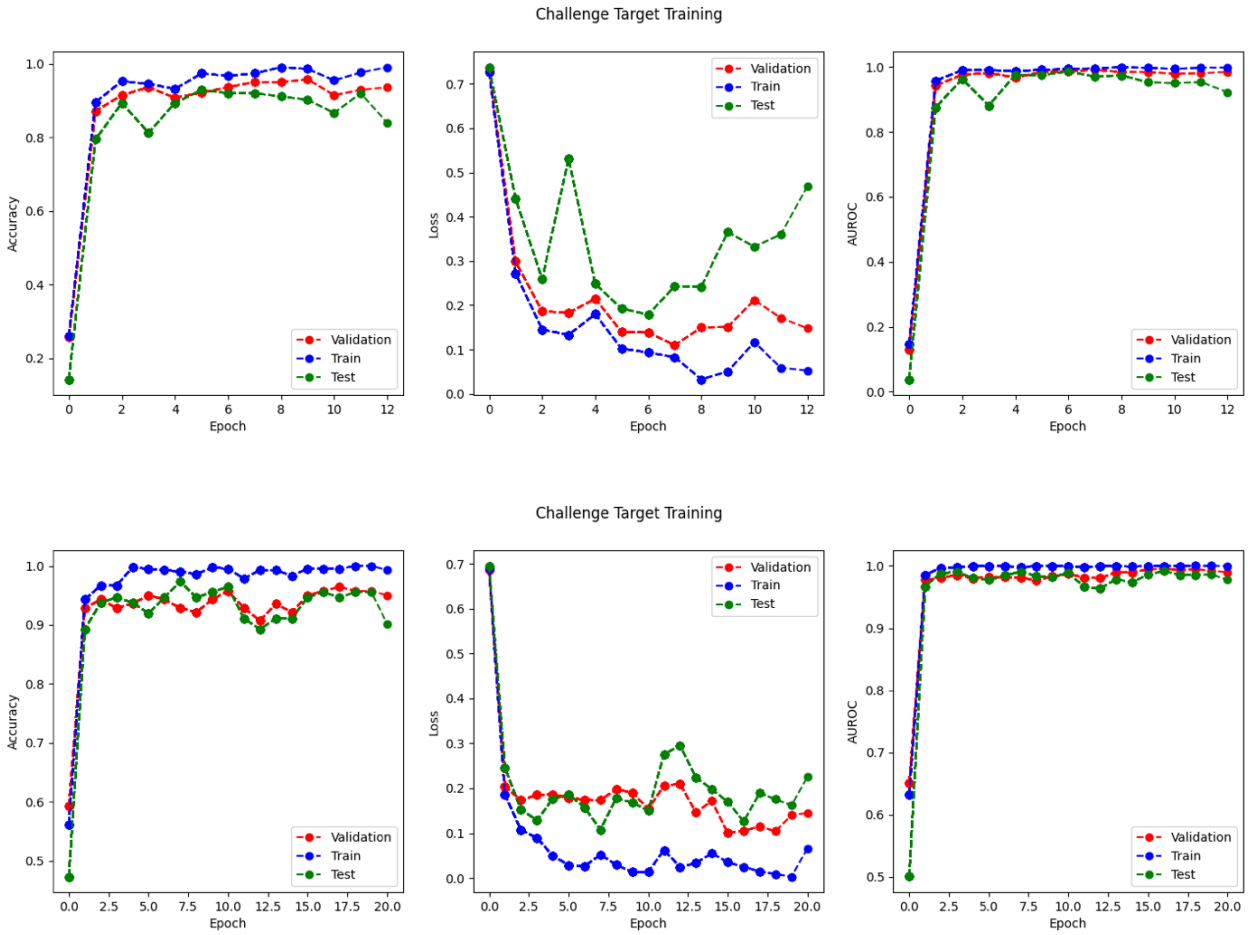
If the user replies `y`, then this function will direct to transfer learning part, which has the code as follows:

```
freeze_none = getResNet18_source()
print("Loading source ...")
freeze_none, _, _ = restore_checkpoint(
    freeze_none, config("challenge_source.checkpoint"), force=True, pretrain=T
)


freeze_whole = copy.deepcopy(freeze_none)
freeze_layers(freeze_whole, 10)

# modify the last layer:
num_class = 2
freeze_none.fc = torch.nn.Linear(freeze_none.fc.in_features, num_class)
freeze_whole.fc = torch.nn.Linear(freeze_whole.fc.in_features, num_class)


train(tr_loader, va_loader, te_loader, freeze_none, "./checkpoints/challenge_t
train(tr_loader, va_loader, te_loader, freeze_whole, "./checkpoints/challenge_
```

Now, I'd like to briefly discuss the meaning of the above code.

- In `restore_checkpoint` , we can restore the checkpoint we learned from `train_challenge_source.py` .
- In `freeze_layers` function, we can specify how many layers to freeze.
- In `freeze_whole.fc = torch.nn.Linear(freeze_whole.fc.in_features, num_class)` , we modify the final layer (the linear layer) to have the output of 2 since we have 2 classes.

After training all the above steps, we can get the following results:



The maximum AUROC is 0.994 and 0,9917, respectively. This is a huge improvement!!!

Data Augment

like what we have done in previous part, we can apply data augmentation to our dataset so that our model can receive more information.

In this section, I decided to use `torchvision.transform` . The following are code implementations.

Rotate

Instead of using fixed rotation like what we have done in previous part, I used random rotation so that the image can be more representative.

```
random_rotation = transforms.RandomRotation(degrees=(-20, 20))
```

Clip, Flip

Clipping and flipping should not change the types of images, this is one of the most popular data augmentation methods.

```
randomVerticalFlip = transforms.RandomVerticalFlip()
random_horizontal_flip = transforms.RandomHorizontalFlip()
random_resized_crop = transforms.RandomResizedCrop(
    (64, 64), scale=(0.1, 1), resized=(0.5, 2))
```

color changing

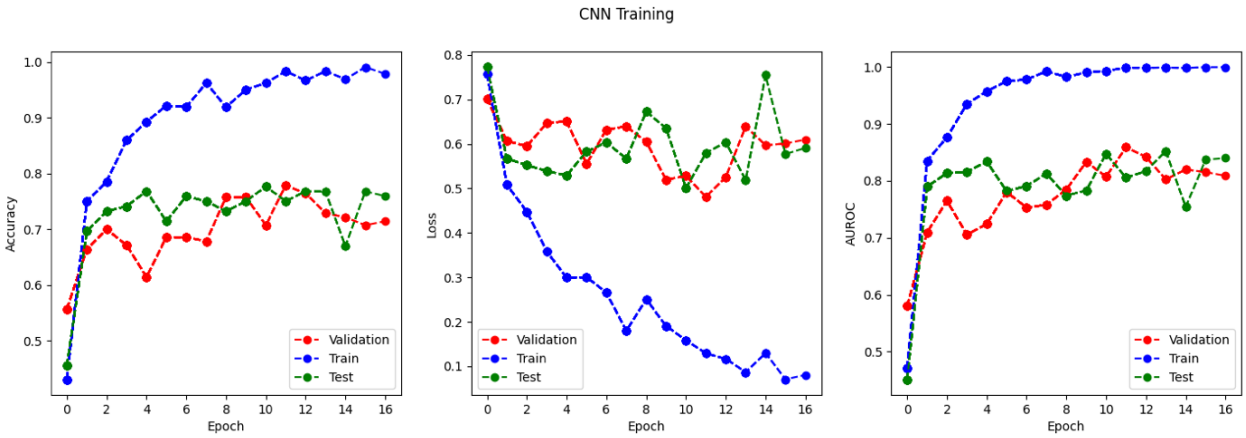
We can also change the color of the images. It contain 4 aspects: Brightness, Contrast, Saturation and Hue.

```
torchvision.transforms.ColorJitter(
    brightness=0.5, contrast=0.5, saturation=0.5, hue=0.5
)
```

Finally, we can add all of the above codes together. The final code is like this:

```
transform_shape = tortransforms.Compose([
    tortransformers.RandomRotation(degrees=(-20, 20)),
    tortransforms.RandomResizedCrop(
        (64, 64), scale=(0.3, 1), retio=(0.5, 2)),
    tortransforms.RandomHorizontalFlip(),
    tortransforms.ToTensor(),
])

transformer_color = tortransformers.ColorJitter(
  brightnenss=0.5, contrast=0.5, saturation=0.5, hue=0.5
)
```

Due to the time limit, I will only test this method on our base model.



It turns out that the performance of the model does not have much improvement. But it's still worth a try.

Feature Selection

In this section, I want to use feature selection to highlight some features in the image so that our model can pay more attention to them. I decided to use edge detection. It's an important feature in image because it tells the shape of a building and is always the boundary of this building. With the help of this, we can eliminate the effect of noisy in the image and make our model concentrate on the shape of a building.

Given that the sky is usually white and the building is usually black (gray) in image, I decide to add

gray color to the detected edge.

The code is as follows:

```
import cv2
import glob

dataset_dir = "data/image_backup"

image_file_paths = []

for file_path in glob.glob(dataset_dir + "/*.png"):
    image_file_paths.append(file_path)

for image_file_path in image_file_paths:

    image = cv2.imread(image_file_path)
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray_image, 150, 255, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(image, contours, -1, (128, 128, 128), 1)
    cv2.imwrite(image_file_path, image)
```

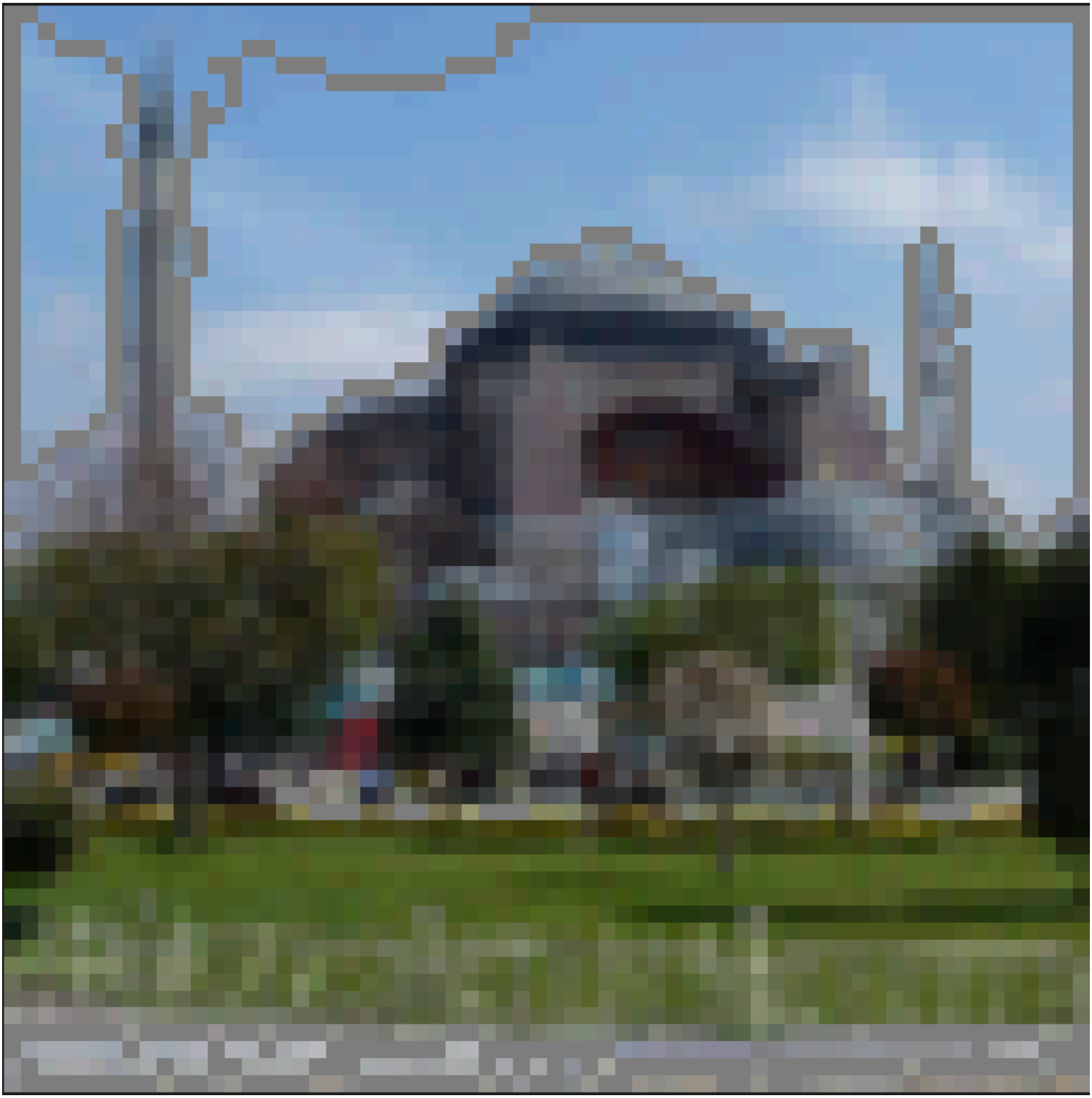
In the above code, I employ some functions in opencv to help me detect the edge of some images. When the edges of a image is detected, the program will add a gray contour of 1 pixel size to it. This actually serves as a way to highlight these features.

One example is like this:

From the image, we can notice that the building and window has gray edges around it.

But sometimes, this implementation may not be a good idea, because this edge detection method sometime has errors.

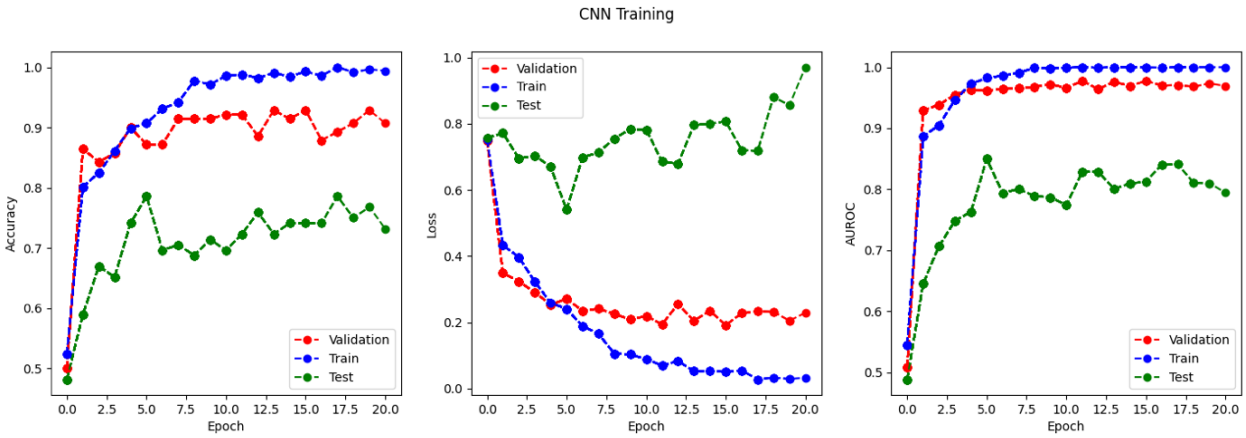
For example, we can see this picture:



The sky has some gray line, which is an error that shouldn't happen.

The code is as follows:

```
import cv2
import glob

dataset_dir = "data/image_backup"

image_file_paths = []

for file_path in glob.glob(dataset_dir + "/*.png"):
    image_file_paths.append(file_path)

for image_file_path in image_file_paths:

    image = cv2.imread(image_file_path)
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray_image, 150, 255, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(image, contours, -1, (128, 128, 128), 1)
    cv2.imwrite(image_file_path, image)
```



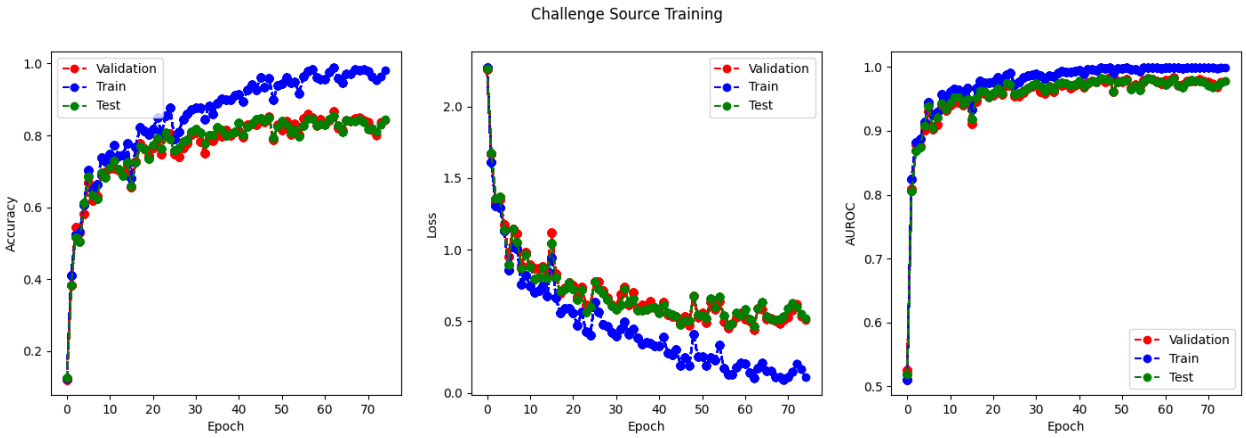
The maximum AUROC is 0.8406. It's a good number. But given the deviation in prediction, I think this method may introduce too much noise into the dataset.

Conclusion

Based on the above analysis, I'd like to apply data augmentation, transfer learning and regularization

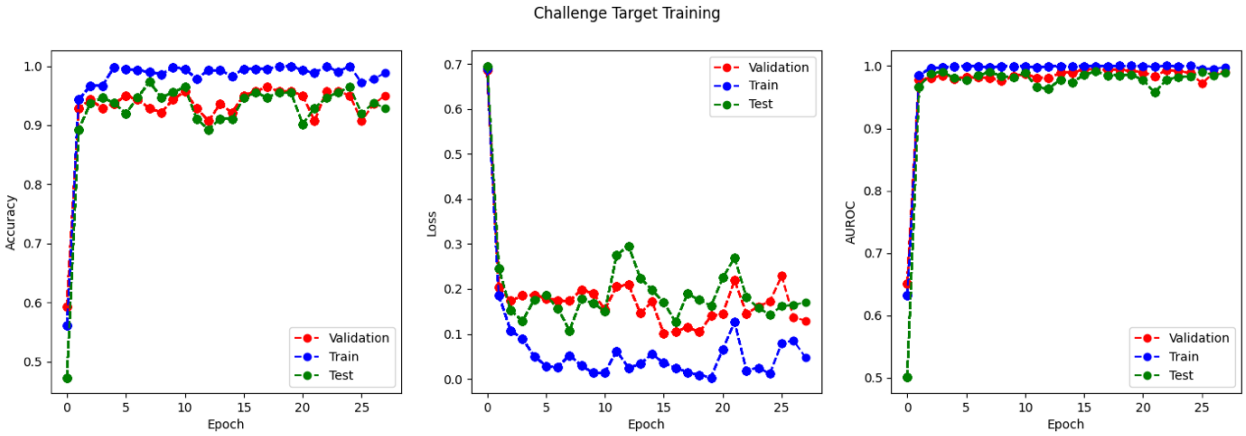
to my handwritten resnet18.

The source training result is:



The maximum epoch is 56: 0.9826

The final performance of my target model is:



The best AUROC score is: 0.9917

```
# argument_data.py
"""
EECS 445 - Introduction to Machine Learning
Fall 2023 - 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.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
        """
        # TODO: implement _rotate(img)
        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.
```

```
            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: implement _grayscale(img)
        grayscale_img = np.mean(img, axis=2)
        grayscale_img = np.round(grayscale_img).astype(np.uint8)
        grayscale_img = np.stack([grayscale_img] * 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 r
    :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", "ta
```

```
    )
    augment_partitions = set(args.partitions)

    # TODO: change `augmentations` to specify which augmentations to apply
    # augmentations = [Grayscale(), Rotate()]
    augmentations = [Grayscale()]
    # augmentations = [Rotate()]

    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:]))



# challenge_augment_data.py
"""
EECS 445 - Introduction to Machine Learning
Fall 2023 - 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
import torchvision.transforms as tortransformers
from imageio.v3 import imread, imwrite

import cv2

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
        """
        # TODO: implement _rotate(img)
        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: implement _grayscale(img)
        grayscale_img = np.mean(img, axis=2)
        grayscale_img = np.round(grayscale_img).astype(np.uint8)
        grayscale_img = np.stack([grayscale_img] * 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 r
    :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 []
```

```
    res = [img] if original else []
    for i in range(n):
        new = img
        for transform in transforms:
            new = transform(new)
        res.append(new)
    return res

transform_shape = tortransformers.Compose([
    tortransformers.RandomRotation(degrees=(-7, 7)),
    # tortransformers.RandomResizedCrop(
    #     (64, 64), scale=(0.7, 1), ratio=(0.5, 2)),
    tortransformers.RandomHorizontalFlip(),
    # tortransformers.ToTensor(),
])

transformer_color = tortransformers.ColorJitter(
  brightness=0.5, contrast=0.5, saturation=0.5, hue=0.5
)

def main(args):
    """Create augmented dataset."""

    use_feature_selection = input("do you want to use feature selection? y/n\n")

    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 = [Grayscale(), Rotate()]
    # augmentations = [Grayscale(), tortransformers.ToPILImage(), transformer_color
    augmentations = [tortransformers.ToPILImage(),
                     transform_shape, transformer_color]
    # augmentations = [Rotate()]

    writer.writeheader()
    os.makedirs(f"{args.datadir}/challenge_augmented/", exist_ok=True)
    for f in glob.glob(f"{args.datadir}/challenge_augmented/*"):
        print(f"Deleting {f}")
        os.remove(f)
    for row in reader:
```

```
        for row in reader:
            if row["partition"] not in augment_partitions:
                imwrite(
                    f"{args.datadir}/challenge_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}/challenge_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:]))
```

```
# challenge_target.py
"""
EECS 445 - Introduction to Machine Learning
Fall 2023 - 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
import torchvision.transforms as tortransformers
from imageio.v3 import imread, imwrite

import cv2

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
        """
        # TODO: implement _rotate(img)
        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: implement _grayscale(img)
        grayscale_img = np.mean(img, axis=2)
        grayscale_img = np.round(grayscale_img).astype(np.uint8)
        grayscale_img = np.stack([grayscale_img] * 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 r
    :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

transform_shape = tortransformers.Compose([
    tortransformers.RandomRotation(degrees=(-7, 7)),
    # tortransformers.RandomResizedCrop(
    #     (64, 64), scale=(0.7, 1), ratio=(0.5, 2)),
    tortransformers.RandomHorizontalFlip(),
```

```
    # tortransformers.ToTensor(),
])

transformer_color = tortransformers.ColorJitter(
  brightness=0.5, contrast=0.5, saturation=0.5, hue=0.5
)

def main(args):
    """Create augmented dataset."""

    use_feature_selection = input("do you want to use feature selection? y/n\n")

    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 = [Grayscale(), Rotate()]
    # augmentations = [Grayscale(), tortransformers.ToPILImage(), transformer_color
    augmentations = [tortransformers.ToPILImage(), transform_shape, transformer_col
    # augmentations = [Rotate()]

    writer.writeheader()
    os.makedirs(f"{args.datadir}/challenge_augmented/", exist_ok=True)
    for f in glob.glob(f"{args.datadir}/challenge_augmented/*"):
        print(f"Deleting {f}")
        os.remove(f)
    for row in reader:
        if row["partition"] not in augment_partitions:
            imwrite(
                f"{args.datadir}/challenge_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}/challenge_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:]))



# confusion_matrix.py
"""
EECS 445 - Introduction to Machine Learning
Fall 2023 - Project 2

Generate confusion matrix graphs.
"""

import torch
import numpy as np
from dataset import get_train_val_test_loaders
from model.source import Source
from train_common import *
```

```
from utils import config
import utils
import os
import itertools
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt


def gen_labels(loader, model):
    """Return true and predicted values."""
    y_true, y_pred = [], []
    for X, y in loader:
        with torch.no_grad():
            output = model(X)
            predicted = predictions(output.data)
            y_true = np.append(y_true, y.numpy())
            y_pred = np.append(y_pred, predicted.numpy())
    return y_true, y_pred


def plot_conf(loader, model, sem_labels, png_name):
    """Draw confusion matrix."""
    y_true, y_pred = gen_labels(loader, model)
    cm = confusion_matrix(y_true, y_pred)
    fig, ax = plt.subplots()
    cax = ax.matshow(cm, cmap=plt.cm.Blues, interpolation="nearest")
    cbar = fig.colorbar(cax, fraction=0.046, pad=0.04)
    cbar.set_label("Frequency", rotation=270, labelpad=10)
    for (i, j), z in np.ndenumerate(cm):
        ax.text(j, i, z, ha="center", va="center")
    plt.gcf().text(0.02, 0.4, sem_labels, fontsize=9)
    plt.subplots_adjust(left=0.5)
    ax.set_xlabel("Predictions")
    ax.xaxis.set_label_position("top")
    ax.set_ylabel("True Labels")
    plt.savefig(png_name)


def main():
    """Create confusion matrix and save to file."""
    tr_loader, va_loader, te_loader, semantic_labels = get_train_val_test_loaders(
        task="source", batch_size=config("source.batch_size")
    )

    model = Source()
```

```
    print("Loading source...")
    model, epoch, stats = restore_checkpoint(model, config("source.checkpoint"))

    sem_labels =
        "0 - Colosseum\n1 - Petronas Towers\n2 - Rialto Bridge\n3 - Museu Nacional

    # Evaluate model
    plot_conf(va_loader, model, sem_labels, "conf_matrix.png")


if __name__ == "__main__":
    main()




# dataset_challenge.py
"""
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


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"""
            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."""
            X1 = X - self.image_mean
            X2 = X1 / self.image_std
            return X2




    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_pal
        """
        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)



# dataset.py
"""
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


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"""
        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."""
        X1 = X - self.image_mean
        X2 = X1 / self.image_std
        return X2




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_pal
        """
        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)
```

```
# feature_selection.py
import cv2
import glob

dataset_dir = "data/challenge_augmented"

image_file_paths = []

for file_path in glob.glob(dataset_dir + "/*.png"):
    image_file_paths.append(file_path)

for image_file_path in image_file_paths:

    image = cv2.imread(image_file_path)
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray_image, 150, 255, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(thresh,
                                   cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(image, contours, -1, (128, 128, 128), 1)
    cv2.imwrite(image_file_path, image)



# predict_challenge.py
"""
EECS 445 - Introduction to Machine Learning
Fall 2023 - Project 2

Predict Challenge
    Runs the challenge model inference on the test dataset and saves the
    predictions to disk
    Usage: python predict_challenge.py --uniqname=<uniqname>
"""

import argparse
import torch
import numpy as np
import pandas as pd
import utils
from dataset import get_challenge
from model.challenge import Challenge
from train_common import *
from utils import config
```

```
import utils
from sklearn import metrics
from torch.nn.functional import softmax


def predict_challenge(data_loader, model):
    """
    Runs the model inference on the test set and outputs the predictions
    """
    y_score = []
    for X, y in data_loader:
        output = model(X)
        y_score.append(softmax(output.data, dim=1)[:, 1])
    return torch.cat(y_score)

def main(uniqname):
    """Train challenge model."""
    # data loaders
    if check_for_augmented_data("./data"):
        ch_loader, get_semantic_label = get_challenge(
            task="target",
            batch_size=config("challenge.batch_size"), augment = True
        )
    else:
        ch_loader, get_semantic_label = get_challenge(
            task="target",
            batch_size=config("challenge.batch_size"),
        )

    model = Challenge()

    # Attempts to restore the latest checkpoint if exists
    model, _, _ = restore_checkpoint(model, config("challenge.checkpoint"))

    # Evaluate model
    model_pred = predict_challenge(ch_loader, model)
    print("saving challenge predictions...\n")
    pd_writer = pd.DataFrame(model_pred, columns=["predictions"])
    pd_writer.to_csv(uniqname + ".csv", index=False, header=False)


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--uniqname", required=True)
    args = parser.parse_args()
```

```
    main(args.uniqname)
```

```
# rng_control.py
import torch
import numpy as np
import random

SEED = 42

np.random.seed(SEED)

torch.manual_seed(SEED)
random.seed(SEED)
torch.use_deterministic_algorithms(True)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

```
# test_cnn.py
"""
EECS 445 - Introduction to Machine Learning
Fall 2023 - Project 2

Test CNN
    Test our trained CNN from train_cnn.py on the heldout test data.
    Load the trained CNN model from a saved checkpoint and evaulates using
    accuracy and AUROC metrics.
    Usage: python test_cnn.py
"""

import torch
import numpy as np
import random
from dataset import get_train_val_test_loaders
from model.target import Target
from train_common import *
from utils import config
import utils

import rng_control
```

```
def main():
    """Print performance metrics for model at specified epoch."""
    # Data loaders
    tr_loader, va_loader, te_loader, _ = get_train_val_test_loaders(
        task="target",
        batch_size=config("target.batch_size"),
    )

    # Model
    model = Target()

    # define loss function
    criterion = torch.nn.CrossEntropyLoss()

    # Attempts to restore the latest checkpoint if exists
    print("Loading cnn...")
    model, start_epoch, stats = restore_checkpoint(
        model, config("target.checkpoint"))

    axes = utils.make_training_plot()

    # Evaluate the model
    evaluate_epoch(
        axes,
        tr_loader,
        va_loader,
        te_loader,
        model,
        criterion,
        start_epoch,
        stats,
        include_test=True,
        update_plot=False,
    )


if __name__ == "__main__":
    main()



# train_challenge.py
"""
EECS 445 - Introduction to Machine Learning
Fall 2023 - Project 2
```

```
Train Challenge
    Train a convolutional neural network to classify the heldout images
    Periodically output training information, and saves model checkpoints
    Usage: python train_challenge.py
"""
import torch
import numpy as np
import random
from dataset import get_train_val_test_loaders
# from model.challenge import Challenge
from model.challenge import *
from train_common import *
from challenge_target import *
# import challenge_source as chas
from model.challenge_source import *
from utils import config
import utils
import copy

def freeze_layers(model, size=0):
    """
    Args:
        model (challenge): a model built in challenge.py
        size (int, optional): the size of layers to be frozen. Defaults to 0.
    Due to the time limit, I only implemented
    1. half of the conv layers are freezed: size=1
    2. the whole conv layers are freezed: size=2
    """
    if size <= 0:
        return
    num = size
    for param in model.parameters():
        if num == 0:
            break
        param.requires_grad = False
        num -= 0.5

def train(tr_loader, va_loader, te_loader, model, model_name, num_layers=0):
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

    # Attempts to restore the latest checkpoint if exists
    print("Loading challenge target model with type", num_layers, "frozen")
    model, start_epoch, stats = restore_checkpoint(
        model, model_name
```

```
        model, model_name
    )

    axes = utils.make_training_plot("Challenge Target Training")

    # Evaluate the randomly initialized model
    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 va
    patience = 12
    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)

        # TODO: Implement early stopping
        curr_count_to_patience, global_min_loss = early_stopping(
            stats, curr_count_to_patience, global_min_loss
        )
        #
        epoch += 1
    print("Finished Training")
    # Save figure and keep plot open
    utils.save_challenge_training_plot()
    utils.hold_training_plot()


def main():
```

```
def main():
    # 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"),
        )

    do_transfer_learning = input("use transfer learning?y/n\n")
    if do_transfer_learning == 'n':
        use_which_model = input("use which model? resnet/original\n")
        if use_which_model == 'original':
            model = Challenge2()
        if use_which_model == 'resnet':
            model = getResNet18()

        # TODO: Define loss function and optimizer. Replace "None" with the appropr
        criterion = torch.nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(model.parameters(),
                                     lr=1e-3, weight_decay=0.009)

        # Attempts to restore the latest checkpoint if exists
        print("Loading challenge...")
        model, start_epoch, stats = restore_checkpoint(
            model, config("challenge.checkpoint")
        )

        axes = utils.make_training_plot()

        # Evaluate the randomly initialized model
        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 patienc
        patience = 5
        curr_count_to_patience = 0
```

```
        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,
                            config("challenge.checkpoint"), stats)

            # TODO: Implement early stopping
            curr_count_to_patience, global_min_loss = early_stopping(
                stats, curr_count_to_patience, global_min_loss
            )
            #
            epoch += 1
        print("Finished Training")
        # Save figure and keep plot open
        utils.save_challenge_training_plot()
        utils.hold_training_plot()
    else:
        # freeze_none = getResNet18_target()
        freeze_none = getResNet18_source()
        print("Loading source ...")
        freeze_none, _, _ = restore_checkpoint(
            freeze_none, config("challenge_source.checkpoint"),
            force=True, pretrain=True
        )

        freeze_whole = copy.deepcopy(freeze_none)
        freeze_layers(freeze_whole, 10)

        # modify the last layer:
        num_class = 2
        freeze_none.fc = torch.nn.Linear(freeze_none.fc.in_features, num_class)
        freeze_whole.fc = torch.nn.Linear(freeze_whole.fc.in_features, num_class)
        # freeze_layers.fc = torch.nn.Linear(freeze_layers.fc.in_features, num_clas
```

```
        # train(tr_loader, va_loader,
        # te_loader, freeze_none, "./checkpoints/challenge_target0/", 0)
        train(tr_loader, va_loader,
              te_loader, freeze_whole, "./checkpoints/challenge_target1/", 1)

    # Model
    # model = Challenge()
    # resnet_8class, _, _ = restore_checkpoint(
    #     freeze_none,
    #     config("challenge_source.checkpoint"), force=True, pretrain=True
    # )

    # resnet_8class = nn.Sequential(*list(resnet_8class.children())[:-1])

    # add own classifier
    # num_class = 2
    # classifier = nn.Sequential(
    #     nn.Flatten(),
    #     nn.Linear(512, num_classes)  # 512 is the number of features in the ResNe
    # )

    # Combine pre-trained ResNet and new classifier
    # model = nn.Sequential(resnet_8class, classifier)




if __name__ == "__main__":
    main()



# train_challenge_source.py
"""
EECS 445 - Introduction to Machine Learning
Fall 2023 - Project 2
Train Source CNN
    Train a convolutional neural network to classify images.
    Periodically output training information, and saves model checkpoints
    Usage: python 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 *
import model.challenge_source as mcs
from utils import config
import utils

import rng_control


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 = mcs.getResNet18_source()

    # TODO: Define loss function and optimizer. Replace "None" with the appropriate
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.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...")
    clear_checkpoint(config("challenge_source.checkpoint"))
    model, start_epoch, stats = restore_checkpoint(
        model, config("challenge_source.checkpoint"))

    axes = utils.make_training_plot("Challenge Source Training")

    # Evaluate the randomly initialized model
    evaluate_epoch(
        axes,
        tr_loader,
```

```
        va_loader,
        te_loader,
        model,
        criterion,
        start_epoch,
        stats,
        multiclass=True,
        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 va
    patience = 12
    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,
            include_test=True
        )

        # Save model parameters
        save_checkpoint(model, epoch + 1,
                        config("challenge_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()



# train_cnn.py
"""
EECS 445 - Introduction to Machine Learning
Fall 2023 - Project 2

Train CNN
    Train a convolutional neural network to classify images
    Periodically output training information, and save model checkpoints
    Usage: python train_cnn.py
"""
import torch
import numpy as np
import random
from dataset import get_train_val_test_loaders
from model.target import Target
from train_common import *
from utils import config
import utils

import rng_control

def main():
    """Train CNN and show training plots."""
    # Data loaders
    if check_for_augmented_data("./data"): # if "augmented_landmarks.csv" exists ir
        # if go into this if statement, shows the user decides to use augmented dat
        tr_loader, va_loader, te_loader,
            _ = get_train_val_test_loaders(
            task="target", batch_size=config("target.batch_size"), augment=True
        )
    else: # do not want to use augmented data
        tr_loader, va_loader, te_loader, _ = get_train_val_test_loaders(
            task="target",
```

```
        batch_size=config("target.batch_size"),
    )
# Model
model = Target() # target is a class(convolutional neural network)

# TODO: Define loss function and optimizer. Replace "None" with the appropriate
criterion = torch.nn.CrossEntropyLoss() # use cross entropy loss function
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3) # use adam optimizer

print("Number of float-valued parameters:",
        count_parameters(model)) # output number of learnable parameters in the

# Attempts to restore the latest checkpoint if exists
print("Loading cnn...")

# add clear_checkpoint for 1.f.iii >>>
clear_checkpoint(config("target.checkpoint")) # remove all the original checkpo
# <<<

model, start_epoch, stats = restore_checkpoint(
    model, config("target.checkpoint")) # restore model from checkpoint if exis

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. Replace "None" with the patience va
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()



# train_common.py
"""
EECS 445 - Introduction to Machine Learning
Fall 2023  - 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
```

```
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`.
    We save the model parameters after each epoch. The periodic saving of model par
    checkpointing. Checking is an important technique for training large models in
    if a hardware failure occurs due to a power outage or our code fails for whatev
    don't want to lose all of our progress.
    """
    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 di
    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
```

```
            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:
```

```
        if cuda:
            checkpoint = torch.load(filename)
        else:
            # Load GPU model on CPU
            checkpoint = torch.load(filename,
                                    map_location=lambda storage, loc: storage)

        print("the check point is:\n", checkpoint)

        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_patience by one if new loss is not less than global_min_loss
        Otherwise, update global_min_loss with the current val loss, and reset curr_cou

        Returns: new values of curr_patience and global_min_loss
        """
        # TODO implement early stopping
```

```
    # TODO: Implement early stopping
    if stats[-1][1] >= global_min_loss:
        curr_count_to_patience += 1
    else:
        global_min_loss = stats[-1][1]
        curr_count_to_patience = 0

    return curr_count_to_patience, global_min_loss


def evaluate_epoch(
    axes,
    tr_loader,
    val_loader,
    te_loader,
    model,
    criterion,
    epoch,
    stats,
    include_test=False,
    update_plot=True,
    multiclass=False,
):
    """Evaluate the `model` on the train and validation set."""
    """
    pass the entire validation set (in batches) through the network
    and get the model's predictions, and compare these with the true
    labels to get and evaluation metric
    """

    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`
    Definition: within one epoch, we pass batches of training examples through the
        use back propagation to compute gradients, and update model weights using t
    """
    for i, (X, y) in enumerate(data_loader):
        optimizer.zero_grad()
        predict = model(X)
        loss = criterion(predict, y)
        loss.backward()
        optimizer.step()
```

```
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(|

    Returns:
        the predicted class output as a PyTorch Tensor
        the set of outputs is called logits
    """
    pred = []
    for i in range(logits.shape[0]):
        ans = np.argmax(logits[i])
        pred.append(ans)
    return torch.tensor(pred)



# train_source.py
"""
EECS 445 - Introduction to Machine Learning
Fall 2023 - Project 2
Train Source CNN
    Train a convolutional neural network to classify images.
    Periodically output training information, and saves model checkpoints
    Usage: python 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

import rng_control


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. Replace "None" with the appropriate
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.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...")
clear_checkpoint(config("source.checkpoint"))
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: Define patience for early stopping. Replace "None" with the patience va
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()



# train_target.py
"""
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."""
    # if num_layers == 1:
    #     model.conv1.requires_grad = False
    # if num_layers == 2:
    #     model.conv1.requires_grad = False
    #     model.conv2.requires_grad = False
    # if num_layers == 3:
    #     model.conv1.requires_grad = False
    #     model.conv2.requires_grad = False
    #     model.conv3.requires_grad = False
    if num_layers <= 0:
        return
    new_num_layers = num_layers
    for name, param in model.named_parameters():
        if new_num_layers == 0:
            break
        param.requires_grad = False
        new_num_layers -= 0.5

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

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

    axes = utils.make_training_plot("Target Training")

    evaluate_epoch(
```

```
        axes,
        tr_loader,
        va_loader,
        te_loader,
        model,
        criterion,
        start_epoch,
        stats,
        include_test=True,
    )

    # initial val loss for early stopping
    global_min_loss = stats[0][1]

    # TODO: Define patience for early stopping. Replace "None" with the patience va
    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() # class:2
    print("Loading source...")
    freeze_none, _, _ = restore_checkpoint( # get trained parameters here
        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/",


if __name__ == "__main__":
    main()



# utils.py
"""
```

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

import os
import numpy as np
import matplotlib.pyplot as plt


def config(attr):
    """
    Retrieves the queried attribute value from the config file. Loads the
    config file on first call.
    """
    if not hasattr(config, "config"): # hasattr: return 1 if config has attribute '
        with open("config.json") as f:
            config.config = eval(f.read())
    node = config.config
    for part in attr.split("."):
        node = node[part]
    return node


def denormalize_image(image):
    """Rescale the image's color space from (min, max) to (0, 1)"""
    ptp = np.max(image, axis=(0, 1)) - np.min(image, axis=(0, 1))
    return (image - np.min(image, axis=(0, 1))) / ptp


def hold_training_plot():
    """
    Keep the program alive to display the training plot
    """
    plt.ioff()
    plt.show()


def log_training(epoch, stats):
    """Print the train, validation, test accuracy/loss/auroc.

    Each epoch in `stats` should have order
        [val_acc, val_loss, val_auc, train_acc, ...]
    Test accuracy is optional and will only be logged if stats is length 9.
    """
```

```
    splits = ["Validation", "Train", "Test"]
    metrics = ["Accuracy", "Loss", "AUROC"]
    print("Epoch {}".format(epoch))
    for j, split in enumerate(splits):
        for i, metric in enumerate(metrics):
            idx = len(metrics) * j + i
            if idx >= len(stats[-1]):
                continue
            print(f"\t{split} {metric}:{round(stats[-1][idx],4)}")


def make_training_plot(name="CNN Training"):
    """Set up an interactive matplotlib graph to log metrics during training."""
    plt.ion()
    fig, axes = plt.subplots(1, 3, figsize=(20, 5))
    plt.suptitle(name)
    axes[0].set_xlabel("Epoch")
    axes[0].set_ylabel("Accuracy")
    axes[1].set_xlabel("Epoch")
    axes[1].set_ylabel("Loss")
    axes[2].set_xlabel("Epoch")
    axes[2].set_ylabel("AUROC")

    return axes


def update_training_plot(axes, epoch, stats):
    """Update the training plot with a new data point for loss and accuracy."""
    splits = ["Validation", "Train", "Test"]
    metrics = ["Accuracy", "Loss", "AUROC"]
    colors = ["r", "b", "g"]
    for i, metric in enumerate(metrics):
        for j, split in enumerate(splits):
            idx = len(metrics) * j + i
            if idx >= len(stats[-1]):
                continue
            # __import__('pdb').set_trace()
            axes[i].plot(
                range(epoch - len(stats) + 1, epoch + 1),
                [stat[idx] for stat in stats],
                linestyle="--",
                marker="o",
                color=colors[j],
            )
        axes[i].legend(splits[: int(len(stats[-1]) / len(metrics))])
```

```
    plt.pause(0.00001)


def save_cnn_training_plot():
    """Save the training plot to a file."""
    plt.savefig("cnn_training_plot.png", dpi=200)


def save_tl_training_plot(num_layers):
    """Save the transfer learning training plot to a file."""
    if num_layers == 0:
        plt.savefig("TL_0_layers.png", dpi=200)
    elif num_layers == 1:
        plt.savefig("TL_1_layers.png", dpi=200)
    elif num_layers == 2:
        plt.savefig("TL_2_layers.png", dpi=200)
    elif num_layers == 3:
        plt.savefig("TL_3_layers.png", dpi=200)


def save_source_training_plot():
    """Save the source learning training plot to a file."""
    plt.savefig("source_training_plot.png", dpi=200)


def save_challenge_training_plot():
    """Save the challenge learning training plot to a file."""
    plt.savefig("challenge_training_plot.png", dpi=200)



# visualize_cnn.py
"""
EECS 445 - Introduction to Machine Learning
Fall 2023  - Project 2
Grad-CAM Visualization
    This script generates a heat map on top of the original image for 20
    sample images in the dataset to help visualize what is learned by
    the convolutional networks.

    Output files will be titled CNN_viz1_<number>.png

    Usage: python visualize_cc.py

Original credit to:
```

```
Author:   Kazuto Nakashima
URL:      http://kazuto1011.github.io
Created:  2017-05-26
"""

import numpy as np
import torch
from matplotlib import pyplot as plt
import matplotlib.cm as cm
from dataset import get_train_val_test_loaders
from model.target import Target
from model.source import Source
from train_common import *
from utils import config
import utils
from collections import OrderedDict
from torch.nn import functional as F
from imageio.v3 import imread


def save_gradcam(gcam, original_image, axarr, i):
    cmap = cm.viridis(np.squeeze(gcam.numpy()))[..., :3] * 255.0
    raw_image = (
        (
            (original_image - original_image.min())
            / (original_image.max() - original_image.min())
        )
        * 255
    ).astype("uint8")
    gcam = (cmap.astype(np.float64) + raw_image.astype(np.float64)) / 2
    axarr[1].imshow(np.uint8(gcam))
    axarr[1].axis("off")
    plt.savefig("CNN_viz1_{}.png".format(i), dpi=200, bbox_inches="tight")


class _BaseWrapper(object):
    """
    Please modify forward() and backward() according to your task.
    """

    def __init__(self, model, fmaps=None):
        super(_BaseWrapper, self).__init__()
        self.device = next(model.parameters()).device
        self.model = model
        self.handlers = []  # a set of hook function handlers
```

```
    def _encode_one_hot(self, ids):
        one_hot = torch.zeros_like(self.logits).to(self.device)
        one_hot.scatter_(1, ids, 1.0)
        return one_hot

    def forward(self, image):
        """
        Simple classification
        """
        self.model.zero_grad()
        self.logits = self.model(image)
        if type(self.logits) is tuple:
            self.logits = self.logits[0]
        self.probs = torch.nn.Sigmoid()(self.logits)
        return self.probs.sort(dim=1, descending=True)

    def backward(self, ids):
        """
        Class-specific backpropagation
        Either way works:
        1. self.logits.backward(gradient=one_hot, retain_graph=True)
        2. (self.logits * one_hot).sum().backward(retain_graph=True)
        """
        one_hot = self._encode_one_hot(ids)
        self.logits.backward(gradient=one_hot, retain_graph=True)

    def generate(self):
        raise NotImplementedError

    def remove_hook(self):
        """
        Remove all the forward/backward hook functions
        """
        for handle in self.handlers:
            handle.remove()


class BackPropagation(_BaseWrapper):
    def forward(self, image):
        self.image = image
        self.image.requires_grad = False
        return super(BackPropagation, self).forward(self.image)

    def generate(self):
```

```
        gradient = self.image.grad.clone()
        self.image.grad.zero_()
        return gradient


class GradCAM(_BaseWrapper):
    """
    "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localizati
    https://arxiv.org/pdf/1610.02391.pdf
    Look at Figure 2 on page 4
    """

    def __init__(self, model, candidate_layers=None):
        super(GradCAM, self).__init__(model)
        self.fmap_pool = OrderedDict()
        self.grad_pool = OrderedDict()
        self.candidate_layers = candidate_layers  # list

        def forward_hook(key):
            def forward_hook_(module, input, output):
                # Save featuremaps
                a = output.detach().cpu()
                self.fmap_pool[key] = a
                del output
                del a
                torch.cuda.empty_cache()

            return forward_hook_

        def backward_hook(key):
            def backward_hook_(module, grad_in, grad_out):
                # Save the gradients correspond to the featuremaps
                a = grad_out[0].detach().cpu()
                self.grad_pool[key] = a
                torch.cuda.empty_cache()

            return backward_hook_

        # If any candidates are not specified, the hook is registered to all the la
        for name, module in self.model.named_modules():
            if self.candidate_layers is None or name in self.candidate_layers:
                self.handlers.append(
                    module.register_forward_hook(forward_hook(name)))
                self.handlers.append(
                    module.register_full_backward_hook(backward_hook(name))
```

```
                )

    def find(self, pool, target_layer):
        if target_layer in pool.keys():
            return pool[target_layer]
        else:
            raise ValueError("Invalid layer name: {}".format(target_layer))

    def _compute_grad_weights(self, grads):
        return F.adaptive_avg_pool2d(grads, 1)

    def forward(self, image):
        self.image = image
        self.image_shape = image.shape[2:]
        return super(GradCAM, self).forward(self.image)

    def generate(self, target_layer):
        fmaps = self.find(self.fmap_pool, target_layer)
        grads = self.find(self.grad_pool, target_layer)
        weights = self._compute_grad_weights(grads)
        gcam = torch.mul(fmaps, weights).sum(dim=1, keepdim=True)
        gcam = F.relu(gcam)
        gcam = F.interpolate(
            gcam, self.image_shape, mode="bilinear", align_corners=False
        )
        B, C, H, W = gcam.shape
        gcam = gcam.view(B, -1)
        gcam -= gcam.min(dim=1, keepdim=True)[0]
        gcam /= gcam.max(dim=1, keepdim=True)[0]
        gcam = gcam.view(B, C, H, W)
        return gcam


device = torch.device("cpu")


def get_image(img_num):
    img_path = "data/images/" + img_num + ".png"
    img = imread(img_path)
    return img


def visualize_input(img_num, axarr):
    xi = get_image(img_num)
    axarr[0].imshow(utils.denormalize_image(xi))
```

```
    axarr[0].axis("off")


def visualize_layer1_activations(img_num, i, axarr):
    xi = get_image(img_num)
    xi = xi.transpose(2, 0, 1)
    xi = torch.from_numpy(xi).float()
    xi = xi.view((1, 3, 64, 64))
    bp = BackPropagation(model=model)
    gcam = GradCAM(model=model)
    target_layer = "conv1"
    target_class = 1
    _ = gcam.forward(xi)
    gcam.backward(ids=torch.tensor([[target_class]]).to(device))
    regions = gcam.generate(target_layer=target_layer)
    activation = regions.detach()
    save_gradcam(
        np.squeeze(activation),
        utils.denormalize_image(np.squeeze(xi.numpy()).transpose(1, 2, 0)),
        axarr,
        i,
    )


if __name__ == "__main__":
    # Attempts to restore from checkpoint
    print("Loading cnn...")
    model = Target()
    model, start_epoch, _ = restore_checkpoint(
        model, config("target.checkpoint"), force=True
    )

    tr_loader, va_loader, te_loader, _ = get_train_val_test_loaders(
        task="target",
        batch_size=config("target.batch_size"),
    )

    # img_list contains the ids for a sample of images from the training set
    img_list = ['01105', '00024', '03545', '05934', '03165',
                '05354', '01295', '00914', '03035', '07984',
                '09125', '05654', '05125', '05174', '04685',
                '12235', '00975', '01074', '02995', '01134']
    for i, img_num in enumerate(img_list):
        plt.clf()
        f, axarr = plt.subplots(1, 2)
```

```
        visualize_input(img_num, axarr)
        visualize_layer1_activations(img_num, i, axarr)
        plt.close()



# visualize_data.py
"""
EECS 445 - Introduction to Machine Learning
Fall 2023  - Project 2
Visualize Landmarks
    This will open up a window displaying randomly selected training
    images. The label of the image is shown. Click on the figure to
    refresh with a set of new images. You can save the images using
    the save button. Close the window to break out of the loop.

    The success of this script is a good indication that the data flow
    part of this project is running smoothly.

    Usage: python visualize_data.py
"""
import matplotlib.pyplot as plt
import numpy as np
import os
import platform
import pandas as pd

from dataset import resize, ImageStandardizer, LandmarksDataset
from imageio.v3 import imread
from utils import config, denormalize_image

OUT_FILENAME = "visualize_data.png"

training_set = LandmarksDataset("train")
training_set.X = resize(training_set.X)
standardizer = ImageStandardizer()
standardizer.fit(training_set.X)

metadata = pd.read_csv(config("csv_file"))
metadata = metadata[metadata["partition"] != "challenge"].reset_index(drop=True)

N = 4
fig, axes = plt.subplots(nrows=2, ncols=N, figsize=(2 * N, 2 * 2))

pad = 3
axes[0, 0].annotate(
```

```
axes[0, 0].annotate(
    "Original",
    xy=(0, 0.5),
    xytext=(-axes[0, 0].yaxis.labelpad - pad, 0),
    xycoords=axes[0, 0].yaxis.label,
    textcoords="offset points",
    size="large",
    ha="right",
    va="center",
    rotation="vertical",
)
axes[1, 0].annotate(
    "Preprocessed",
    xy=(0, 0.5),
    xytext=(-axes[1, 0].yaxis.labelpad - pad, 0),
    xycoords=axes[1, 0].yaxis.label,
    textcoords="offset points",
    size="large",
    ha="right",
    va="center",
    rotation="vertical",
)

for ax in axes.flatten():
    ax.set_xticks([])
    ax.set_yticks([])


def display_imgs():
    rand_idx = np.random.choice(np.arange(len(metadata)), size=N, replace=False)
    X, y = [], []
    for idx in rand_idx:
        filename = os.path.join(config("image_path"),
                                metadata.loc[idx, "filename"])
        X.append(imread(filename))
        y.append(metadata.loc[idx, "semantic_label"])

    for i, (xi, yi) in enumerate(zip(X, y)):
        axes[0, i].imshow(xi)
        axes[0, i].set_title(yi.replace("_", "\n"))

    X_ = resize(np.array(X))
    X_ = standardizer.transform(X_)
    for i, (xi, yi) in enumerate(zip(X_, y)):
        axes[1, i].imshow(denormalize_image(xi),
                          interpolation="bicubic")
```

```
                                interpolation='bicubic')


if platform.system() == "Linux" and "DISPLAY" not in os.environ:
    print(
        f'No window server found, switching \
            to writing first {N} images to file "{OUT_FILENAME}"'
    )
    display_imgs()
    fig.savefig(OUT_FILENAME, bbox_inches="tight")
    exit(0)

print(
    "I will display some images. \
        Click on the figure to refresh. Close the figure to exit."
)

while True:
    display_imgs()
    plt.draw()
    if plt.get_fignums():
        print(plt.get_fignums())
    else:
        print(None)
    fig.savefig(OUT_FILENAME, bbox_inches="tight")
    if plt.waitforbuttonpress(0) == None:
        break

print("OK, bye!")
```