Gradescope

t11902210 張一凡

Question1(+2 pts): Visualize distribution of features accross different classes.

1.Please make t-SNE plot the distribution of early, middle, final stage.

a. Evaluate the model on training dataset, collect features and labels

b. Make 3 t-SNE plots of the following training phase:

i. early stage

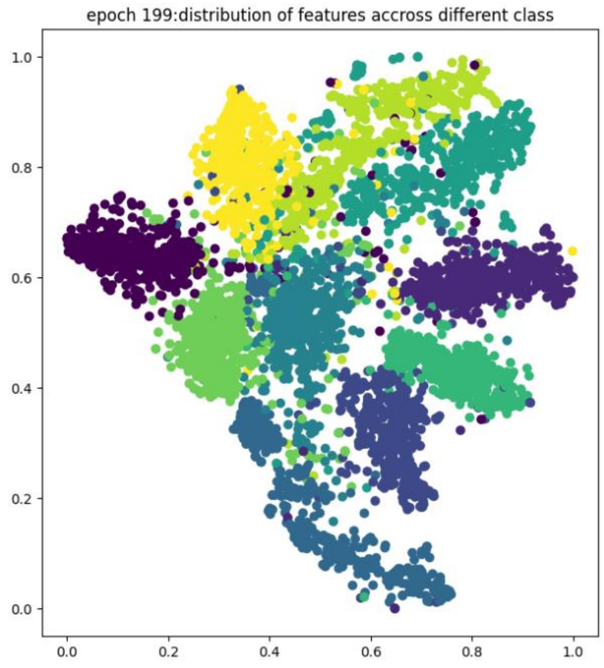
ii. middle stage

iii. final stage

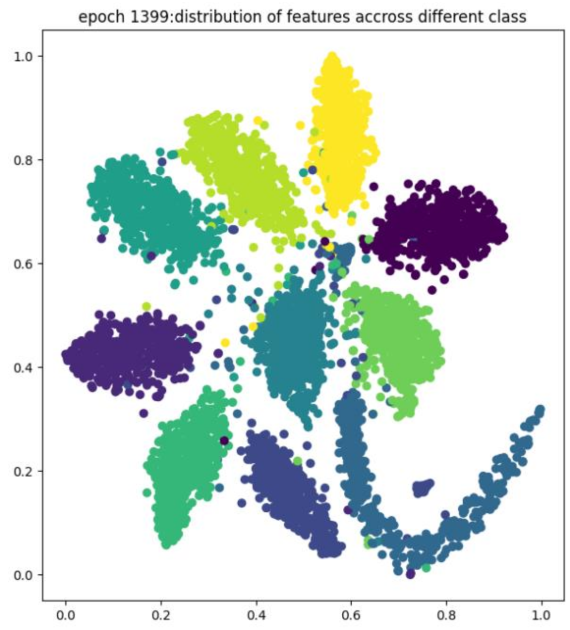
Answer:

For the first question, I have fully introduced the t-SNE plot of the distribution of early, middle, and final stages related to distribution of features across different classes. The code for generating images is the final part of this question.(I used the image generated by my first version of code, so it may be slightly different from the image generated by the last submitted code)

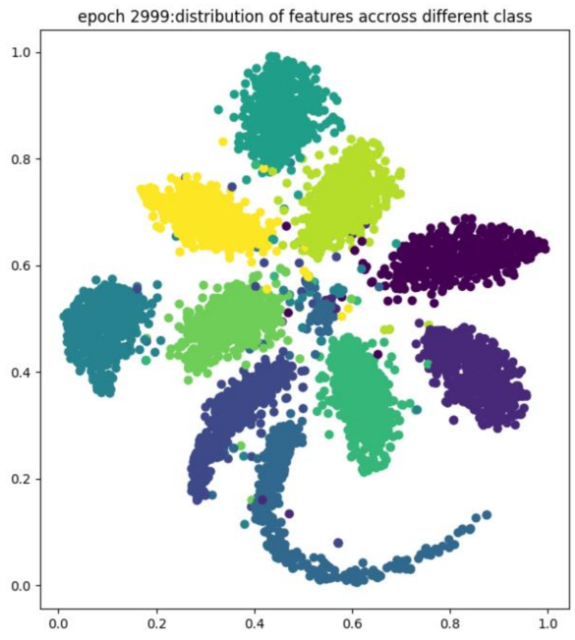
i. early stage (epoch=199/3000)



ii. middle stage (epoch=1399/3000)

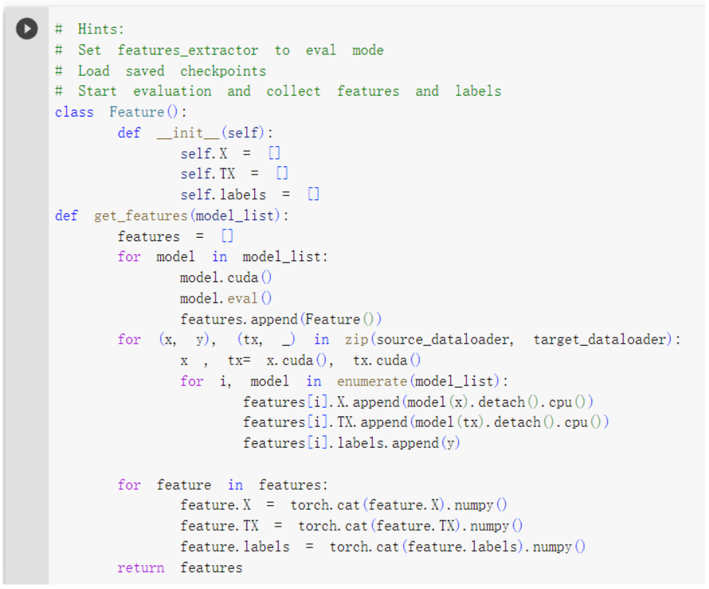


iii. final stage (epoch=2999/3000)



The following is the code for generating images that I have shown. As this code generates the images from this question and the second question together, the code is not divided and is displayed together.

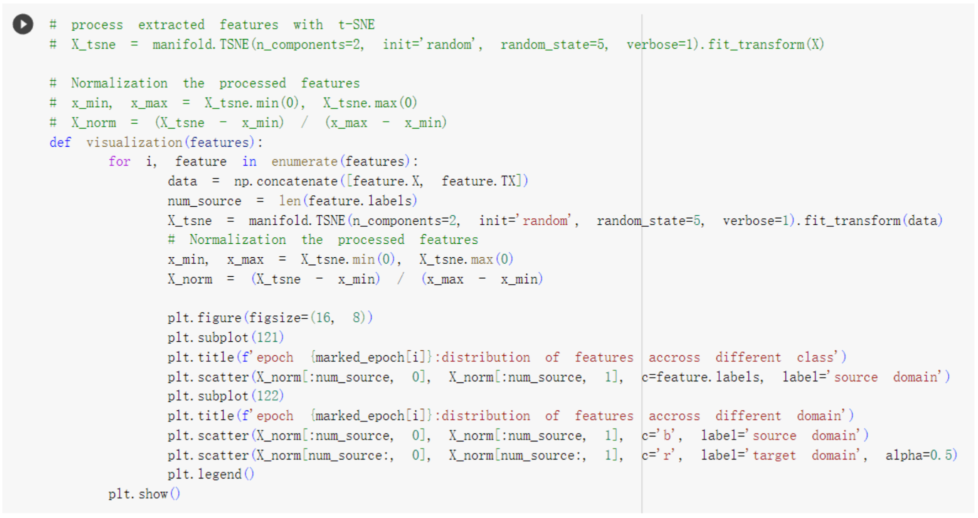
Step1: Load checkpoint and evaluate to get extracted features



```python
# Hints:
# Set features_extractor to eval mode
# Load saved checkpoints
# Start evaluation and collect features and labels
class Feature():
        def __init__(self):
                self.X = []
                self.TX = []
                self.labels = []
def get_features(model_list):
        features = []
        for model in model_list:
                model.cuda()
                model.eval()
                features.append(Feature())
        for (x, y), (tx, _) in zip(source_dataloader, target_dataloader):
                x , tx= x.cuda(), tx.cuda()
                for i, model in enumerate(model_list):
                        features[i].X.append(model(x).detach().cpu())
                        features[i].TX.append(model(tx).detach().cpu())
                        features[i].labels.append(y)

        for feature in features:
                feature.X = torch.cat(feature.X).numpy()
                feature.TX = torch.cat(feature.TX).numpy()
                feature.labels = torch.cat(feature.labels).numpy()
        return features
```

Step2: Apply t-SNE and normalize

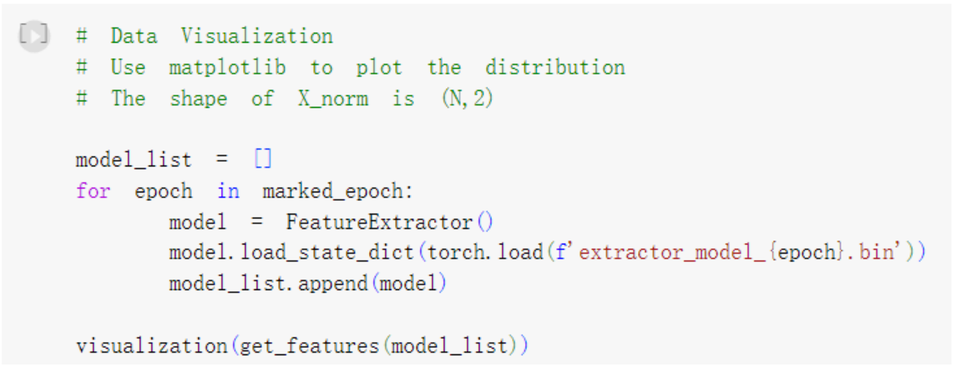


```python
# process extracted features with t-SNE
# X_tsne = manifold.TSNE(n_components=2, init='random', random_state=5, verbose=1).fit_transform(X)

# Normalization the processed features
# x_min, x_max = X_tsne.min(0), X_tsne.max(0)
# X_norm = (X_tsne - x_min) / (x_max - x_min)
def visualization(features):
        for i, feature in enumerate(features):
                data = np.concatenate([feature.X, feature.TX])
                num_source = len(feature.labels)
                X_tsne = manifold.TSNE(n_components=2, init='random', random_state=5, verbose=1).fit_transform(data)
                # Normalization the processed features
                x_min, x_max = X_tsne.min(0), X_tsne.max(0)
                X_norm = (X_tsne - x_min) / (x_max - x_min)

                plt.figure(figsize=(16, 8))
                plt.subplot(121)
                plt.title(f'epoch {marked_epoch[i]}:distribution of features accross different class')
                plt.scatter(X_norm[:num_source, 0], X_norm[:num_source, 1], c=feature.labels, label='source domain')
                plt.subplot(122)
                plt.title(f'epoch {marked_epoch[i]}:distribution of features accross different domain')
                plt.scatter(X_norm[:num_source, 0], X_norm[:num_source, 1], c='b', label='source domain')
                plt.scatter(X_norm[num_source:, 0], X_norm[num_source:, 1], c='r', label='target domain', alpha=0.5)
                plt.legend()
        plt.show()
```

Step3: Visualization with matplotlib



```python
# Data Visualization
# Use matplotlib to plot the distribution
# The shape of X_norm is (N,2)

model_list = []
for epoch in marked_epoch:
        model = FeatureExtractor()
        model.load_state_dict(torch.load(f'extractor_model_{epoch}.bin'))
        model_list.append(model)

visualization(get_features(model_list))
```

Reference link for visualization code：
https://blog.csdn.net/iwill323/article/details/128067213

2. Explain and analyze the distribution of features of three stages.

a. Hint: Is this a good feature extractor for classification task? Why or Why not?

Answer:

In the early, mid, and final stages of training, the distribution of features may vary depending on the specific context and characteristics of the dataset. However, the following are my analyses of the early, mid, and late stages based on the output of the image from the code:

Early stage:

In the early stages of training, the model was just starting to learn patterns and representations from the data, so the features between different categories were quite chaotic. I chose the 199th out of 3000 epochs for output display. Although it can be seen that the differentiation of each feature is somewhat embryonic, it is generally a sense of interlocking. This is because the distribution of different features is relatively diverse, and the relationships between features in the same category are relatively dispersed, Not able to find one's own category well. Overall, this part of the model is still exploring the data and attempting to capture the potential relationship between features and labels for differentiation, and the distribution may exhibit a higher degree of uncertainty and inconsistency. This stage is not a good Domain Adaptation task feature extractor.

Middle stage:

As the training progresses, the model gradually converges and refines the division of label features. I selected the 1399th out of 3000 epochs for intermediate representation output display. In the mid-term, the distribution of features in different categories is becoming increasingly shaped and dispersed, while features in the same category are becoming more concentrated. This model has learned the features and relationships of different category meanings, thus achieving a clearer distribution of features between categories and clustering of similar categories. These features may exhibit clearer patterns and associations with category labels, indicating that the model is better understanding the data for "classification" and "clustering". However, the distance between different categories is still relatively close, and some category features still have obvious boundaries, while the features of the same category still need to be more concentrated. In this stage, it can be used as a Domain Adaptation task feature extractor, but it needs to be improved.

Final stage:

In the final stage of training, the model underwent extensive learning and refinement, and I selected the 2999th out of 3000 epochs for the final representative output display (which is basically the last one). The distribution of features in different categories becomes more dispersed, and the distribution of features in the same category becomes more compact and dense. This indicates that after such processing, samples belonging to different categories have basically completed "classification" and "clustering" and entered the designated blocks. In this stage, it can be used as a good Domain Adaptation task feature extractor

Quesion2 (+2pts): Visualize distribution of features across different domains.

1. Please plot the distribution of early, middle, final stage.

a. Evaluate the model on source dataset and target dataset, collect feature and labels

b. Make 3 plots of the following training phase:

i. early stage

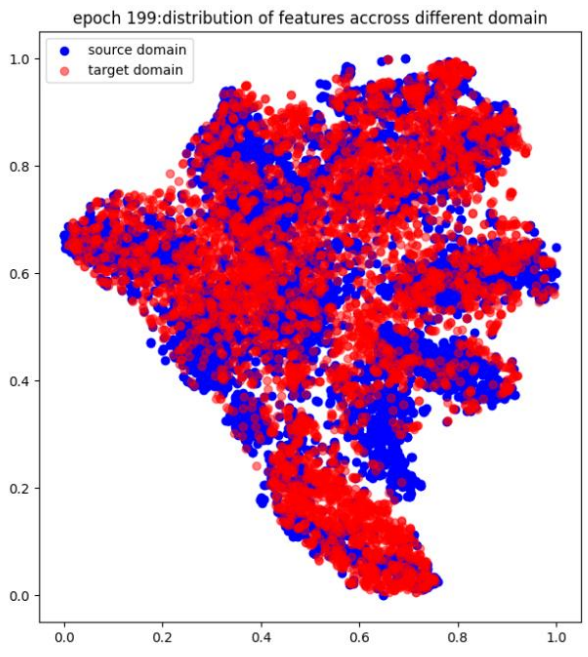
ii. middle stage

iii. final stage

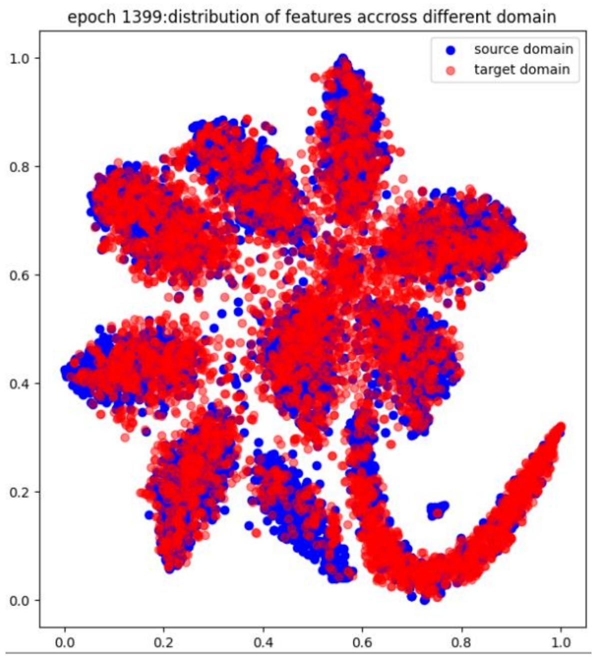
Answer:

For the second question, I have fully presented the distribution of early, middle, and final stage images related to the distribution of features across different domains below, and the code for generating the images is also the final part of this question (Because the images of the two questions were generated together, the same code was used as the first question). (I used the image generated by my first version of code, so it may be slightly different from the image generated by the last submitted code)

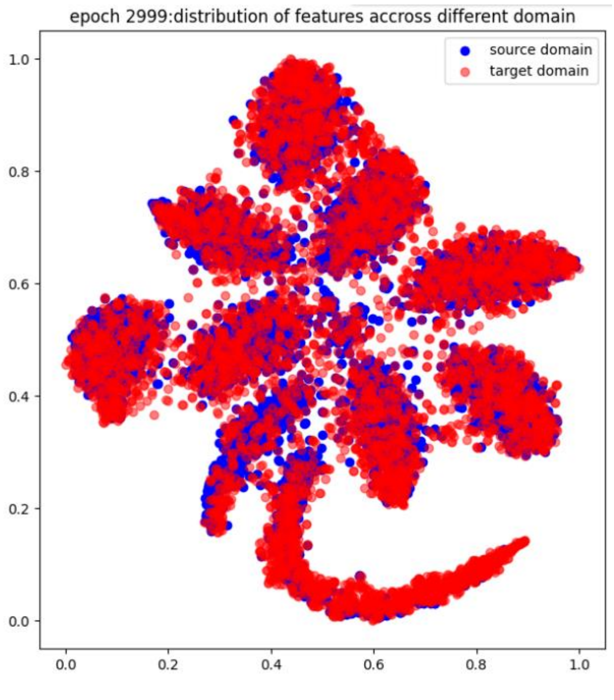
i. early stage (epoch=199/3000)



ii. middle stage (epoch=1399/3000)

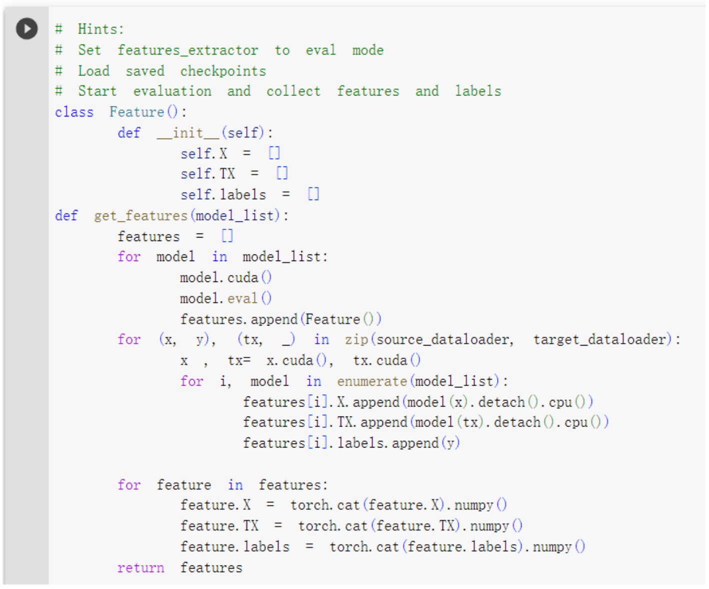


iii. final stage (epoch=2999/3000)



The following is the code I have shown for generating images. Since this code generates the image of the problem and the first problem together, it is not segmented, but displayed together (the same code as the previous problem).

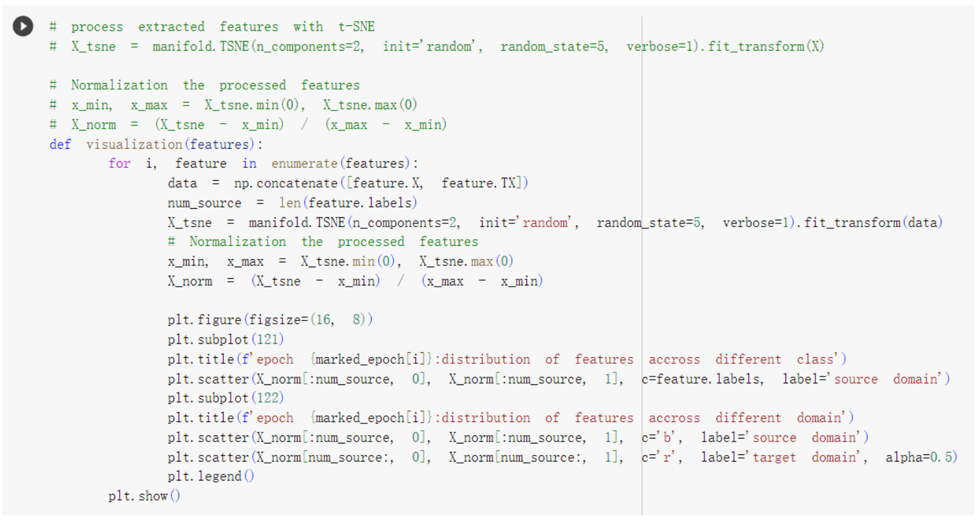
Step1: Load checkpoint and evaluate to get extracted features



```python
# Hints:
# Set features_extractor to eval mode
# Load saved checkpoints
# Start evaluation and collect features and labels
class Feature():
    def __init__(self):
        self.X = []
        self.TX = []
        self.labels = []
def get_features(model_list):
    features = []
    for model in model_list:
        model.cuda()
        model.eval()
        features.append(Feature())
    for (x, y), (tx, _) in zip(source_dataloader, target_dataloader):
        x , tx= x.cuda(), tx.cuda()
        for i, model in enumerate(model_list):
            features[i].X.append(model(x).detach().cpu())
            features[i].TX.append(model(tx).detach().cpu())
            features[i].labels.append(y)

    for feature in features:
        feature.X = torch.cat(feature.X).numpy()
        feature.TX = torch.cat(feature.TX).numpy()
        feature.labels = torch.cat(feature.labels).numpy()
    return features
```

Step2: Apply t-SNE and normalize

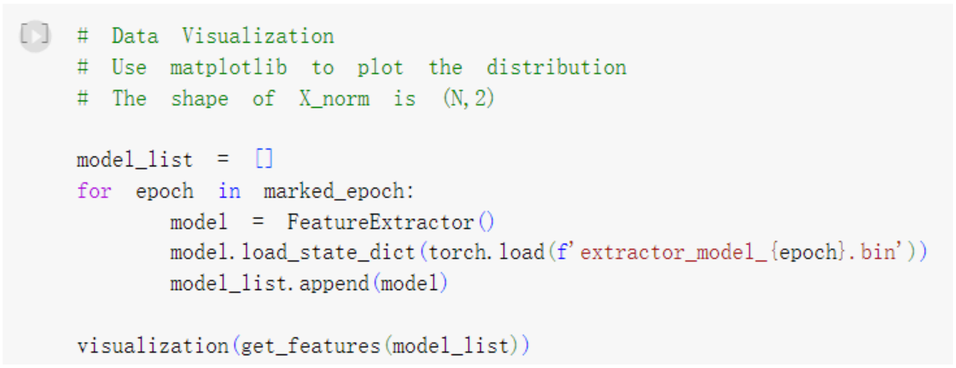


```python
# process extracted features with t-SNE
# X_tsne = manifold.TSNE(n_components=2, init='random', random_state=5, verbose=1).fit_transform(X)

# Normalization the processed features
# x_min, x_max = X_tsne.min(0), X_tsne.max(0)
# X_norm = (X_tsne - x_min) / (x_max - x_min)
def visualization(features):
    for i, feature in enumerate(features):
        data = np.concatenate([feature.X, feature.TX])
        num_source = len(feature.labels)
        X_tsne = manifold.TSNE(n_components=2, init='random', random_state=5, verbose=1).fit_transform(data)
        # Normalization the processed features
        x_min, x_max = X_tsne.min(0), X_tsne.max(0)
        X_norm = (X_tsne - x_min) / (x_max - x_min)

        plt.figure(figsize=(16, 8))
        plt.subplot(121)
        plt.title(f'epoch {marked_epoch[i]}:distribution of features accross different class')
        plt.scatter(X_norm[:num_source, 0], X_norm[:num_source, 1], c=feature.labels, label='source domain')
        plt.subplot(122)
        plt.title(f'epoch {marked_epoch[i]}:distribution of features accross different domain')
        plt.scatter(X_norm[:num_source, 0], X_norm[:num_source, 1], c='b', label='source domain')
        plt.scatter(X_norm[num_source:, 0], X_norm[num_source:, 1], c='r', label='target domain', alpha=0.5)
        plt.legend()
    plt.show()
```

Step3: Visualization with matplotlib



```python
# Data Visualization
# Use matplotlib to plot the distribution
# The shape of X_norm is (N,2)

model_list = []
for epoch in marked_epoch:
    model = FeatureExtractor()
    model.load_state_dict(torch.load(f'extractor_model_{epoch}.bin'))
    model_list.append(model)

visualization(get_features(model_list))
```

Reference link for visualization code：
https://blog.csdn.net/iwill323/article/details/128067213

2. Explain and analyze the distribution of features of three training phases.

a. Hint: Is this a good feature extractor for domain adaption task? Why or Why not?

Answer:

The effectiveness of the feature extractor for domain adaptive tasks depends on the alignment of its feature distribution between the source and target domains. The following are the early, mid-term, and final results and analyses that I will introduce separately:

Early stage:

In the early stages, if the feature distributions of the source and target domains are significantly different, it indicates that the initial feature extractor may not be suitable for domain adaptation. I chose the 199th out of 3000 epochs for observation. From the independent distribution of red and blue, there are many areas where the two do not overlap, indicating that the feature distributions between the two domains are relatively different. The lack of alignment or overlap indicates that the model is striving to capture domain invariant representations, so currently this stage is not a good feature extractor for domain adaptation task

Middle stage:

If the feature distribution begins to exhibit alignment or clustering in the intermediate stage, it indicates that the feature extractor is gradually learning domain invariant representations. I selected the 1399th of 3000 epochs for observation, and from the independent distribution of red and blue, their respective category distributions have gradually formed, with some overlap in the red and blue areas (such as the area at the bottom right), However, the overall overlap is still quite chaotic (such as a large fuzzy area in the middle on the right), indicating that the feature distribution between the two domains is getting closer to the same distribution, but there are still many flaws. This indicates that the model is making progress in adapting features to the target domain, so at this stage, it is moving towards feature extractor for domain adaptation task. However, if the current extraction does not meet the requirements.

Final stage:

In the final stage, if the feature distribution shows a high degree of alignment or overlap between the source and target domains, it indicates successful domain adaptation. I selected 2999th out of 3000 epochs for observation, and from the independent distribution of red and blue, their respective category distributions have been fully formed. The red and blue regions overlap well, and scattered points are also sparse. The model has learned to extract features that are invariant to domain offset, which is desirable for effective adaptation. Therefore, this result is currently a good feature extractor for domain adaptation task.