學號：B06901053　系級：電機三　　姓名：謝承延

1. 請描述你實作的模型架構、方法以及 accuracy 為何。其中你的方法必須為 domain adversarial training 系列（就是你的方法必須要讓輸入 training data & testing data 後的某一層輸出 domain 要相近）。(2%)

- Model: DANN
- Kaggle Accuracy:0.76246
- 架構:

	Type	Filter Size	Input size
Feature Extractor	Conv2D	3x3x1x64	32x32x1
	Conv2D	3x3x64x64	32x32x64
	BatchNorm2d		32x32x64
	ReLU		32x32x64
	MaxPool2d		32x32x64
	Conv2D	3x3x64x128	16x16x64
	Conv2D	3x3x128x128	16x16x128
	BatchNorm2d		16x16x128
	ReLU		16x16x128
	MaxPool2d		16x16x128
	Conv2D	3x3x128x256	8x8x128
	Conv2D	3x3x256x256	8x8x256
	Conv2D	3x3x256x256	8x8x256
	BatchNorm2d		8x8x256
	ReLU		8x8x256
	MaxPool2d		8x8x256
	Conv2D	3x3x256x256	4x4x256
	Conv2D	3x3x256x256	4x4x256
	Conv2D	3x3x256x256	4x4x256
	BatchNorm2d		4x4x256
	ReLU		4x4x256
	MaxPool2d		4x4x256
	Conv2D	3x3x256x512	2x2x256
	Conv2D	3x3x512x512	2x2x512
	Conv2D	3x3x512x512	2x2x512
	BatchNorm2d		2x2x512
	ReLU		2x2x512
	MaxPool2d		2x2x512
Label Predictor	Linear(512,512)	512	
	ReLU	512	
	Linear(512,512)	512	
	ReLU	512	
	Linear(512,10)	10	
	Linear(512,512)	512	

Domain Classifier	BatchNorm2d	512
	ReLU	512
	Linear(512,512)	512
	BatchNorm2d	512
	ReLU	512
	Linear(512,512)	512
	BatchNorm2d	512
	ReLU	512
	Linear(512,512)	512
	BatchNorm2d	512
	ReLU	512
	Linear(512,1)	512

- 參數設定:

Batch size:32

Optimizer:Adam

Learning rate:0.001

Num_epoch = 750

Loss = predictor loss - lambda*domain classifier loss

Lambda 照以下式子做調整 其中 gamma 為 10，p 依據當前 epoch 由 0 逐漸變為 1

$$\lambda_p = \frac{2}{1+\exp(-\gamma \cdot p)} - 1,$$

- Data Augmentation:

RadomHorizontalflip,

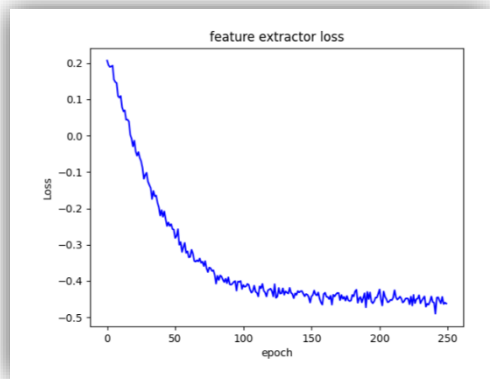
RandomRotation(15),

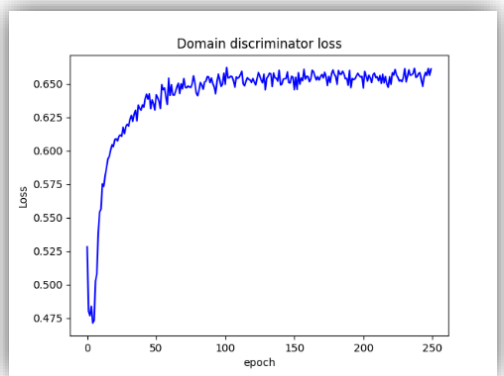
ColorJitter→調整飽和度亮度對比等等

RandomPerspective->透視變換

- Training Loss:

Feature extractor loss:約在-0.48　　Domain Discriminator loss:約為 0.66

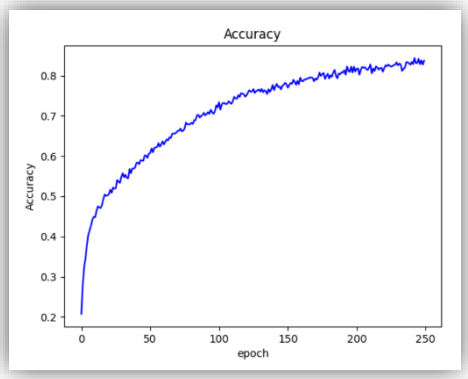


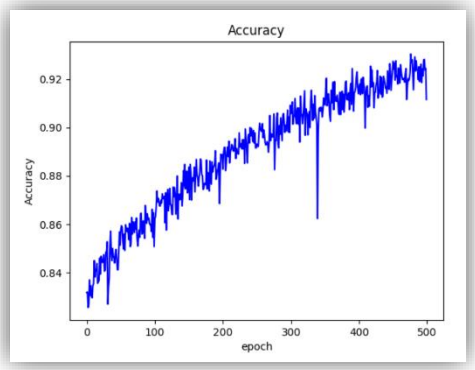


- Train Acc:

大約收斂於 0.94

1~250epoch ：　　250~750epoch:





因為前 250 epoch 和後面 500epoch 是分開訓練的

故分開做圖，可以看到 Accuracy 仍然在持續上升，但考慮到可能會 overfit，最終只停在 epoch750

- 方法描述:

  基本上就是一般的 Domain Adversarial training， 但是加入一些 Data Augmentation

  我的想法是:Domain Adversarial Training 若要成功，則

  1. feature extractor 能夠成功將 source data/target data 的 feature 投影到相同處:

  通過使用第三題的方法，我確定 feature extractor 的確能夠將 source data/target data 投影在相同處，另外使用 loss 做判斷也是一個不錯的方法，我的 model 最終可以達到約 0.68 的 domain classifier loss。

  為何 domain classifier loss 很重要呢，因為假如 domain classifier loss 太小的話，這代表可能 feature extractor 產生的 feature 是無法騙過 domain classifier 的，這意味著 domain classifier 仍然能夠清楚判斷 target data/source data 是屬於不同 domain，造成在 label predictor 在做預測的時候，因為 source data/target data 的 feature 相差太多而無法正確預測 target data。

  2. Model 不能夠 overfit source data

  通過使用第三題的方法，我確定 feature extractor 的確能夠將 source data/target data 投影在相同處後，我開始將重點放在如何讓 model 不要 overfit source data 的部分。

  為甚麼要避免 model overfitting 呢? 其實這樣的講法可能不盡然準確，但主要的想法是:target data 是一些跟 source data 相似但又不完全相同的 data，那假如 model generalize 的能力很好，也許在 model 的觀點中，target data 就只是一種很像 source data 的 data 罷了，所以我認為避免 model overfit source data，可能有助於幫助 target data 的預測。

  為了達成，我使用了很多的 data augmentation:

  除了 RandomFlip 跟 RandomRotation 之外，我先加入 ColorJitter，這一個部分我認為也是最具效果的，使用了這個方法後，我的 performance 幾乎立刻上升了 10%，我認為可能的原因是:我們在做 training/testing 的時候，我們會先將圖片變成 grayscale，這個時候 data 的色彩已經不太重要了，在灰白的圖片當中，通過調整對比亮度等等，可以讓兩張不同圖片看起來非常相似，套用在我們這次的 task，很可能就可以讓 source data 看起來更像 target data。

  後來我又使用了 RandomPerspective 透射變換，加入這個 data Augmentation 之後，model 的 training accuracy 變的上升非常緩慢，這是可以想像的，因為到目前為止已經使用了 RandomFlip RandomRotation ColorJitter RandomPerspective 4 種 data augmentation 的技術，相當於在 training 的過程資料數量比原本多了非常多倍，所以自然訓練過程會變得比較困難，然而使用完這個方法後，accuracy 又大約上升了 2%。

3.Adaptive Parameter:

$$\lambda_p = \frac{2}{1+\exp(-\gamma \cdot p)} - 1,$$

調整 adaptive parameter 也讓我的 model accuracy 有所上升，我照著論文題到的方式，讓 lambda 的數值從 0 逐漸變成 1，我自己猜測這樣子的好處是，一開始專注在 label predictor+feature extractor 的訓練，讓 feature extractor 盡快學習如何擷取有用得 feature 以達到正確的預測，當 training 到達後面時 lambda 越來越大，也就是 loss 中考慮 domain classifier 的比例會越來越高，這時候 feature extractor 就會越來越重視如何產生 feature 以騙過 domain classifier。

- 其他改善方向:

  這一部分是我因為時間不夠來不及嘗試，但我認為可以讓 accuracy 更往上跑的一些想法。

  1.使用更熱門的 CNN 架構:

  前面提到了使用完 data augmentation 之後，accuracy 上升變的緩慢，我認為這個原因有一部分是來自於原本 model 的架構太小了，對於這麼大量的資料，這個架構已經不夠 powerful 去 fit，也就是說，如果使用更強的 model 架構，有機會讓整個 performance 變的更強，我有將 feature extractor 改成 VGG16 的架構，performance 上升大約 5%，但若將 feature extractor 改成更深的架構呢?

  2.Ensemble:

  幾乎每個 kaggle 競賽者都會使用，讓每個 model 對 label 預測的結果去做投票，通常都有助於最終的結果，且這個方法也是避免 overfitting 的好方法，讓具有高 variance 低 bias 特性的 model 去做 emsemble，可能有機會得到低 variance 低 bias 的結果。

  3.Dropout:

  在 label predictor 的地方加入 dropout，通常都可以有效避免 overfit，我也思考過若將 dropout 加在 domain classifier 會如何，但我猜應該不會太好，因為讓 domain classifier 去避免 overfitting 可能比較沒有意義，但是若可以找到方法進一步去增強 domain classifier 的話，也許可以讓 feature extractor 更強(為了騙過 domain classifier)，進而使得預測結果更好。

2. 請視覺化真實圖片以及手繪圖片通過沒有使用 domain adversarial training 的 feature extractor 的 domain 分布圖。(2%)

- 參數設定:

Batch size:32
Optimizer:Adam
Learning rate:0.001
Num_epoch = 200

- Model 架構(no domain classifier):

	Type	Filter Size	Input size
Feature Extractor	Conv2D	3x3x1x64	32x32x1
	BatchNorm2d		32x32x64
	ReLU		32x32x64
	MaxPool2d		32x32x64
	Conv2D	3x3x64x128	16x16x64
	BatchNorm2d		16x16x128
	ReLU		16x16x128
	MaxPool2d		16x16x128
	Conv2D	3x3x128x256	8x8x128
	BatchNorm2d		8x8x256
	ReLU		8x8x256
	MaxPool2d		8x8x256
	Conv2D	3x3x256x256	4x4x256
	BatchNorm2d		4x4x256
	ReLU		4x4x256
	MaxPool2d		4x4x256
	Conv2D	3x3x256x512	2x2x256
	BatchNorm2d		2x2x512
	ReLU		2x2x512
	MaxPool2d		2x2x512
Label Predictor	Linear(512,512)		512
	ReLU		512
	Linear(512,512)		512
	ReLU		512
	Linear(512,10)		10

- Domain distribution:

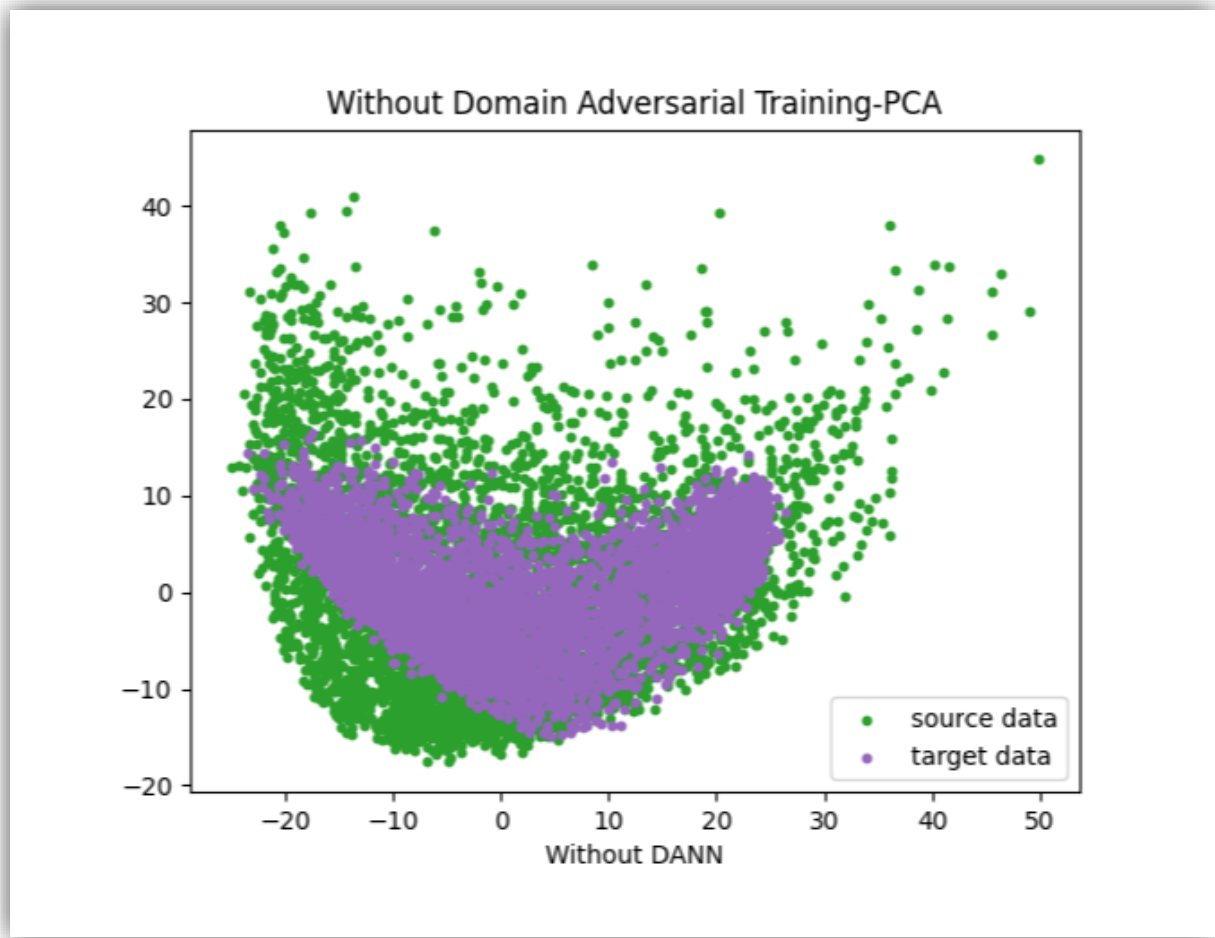
實作方法:

利用訓練好的 feature extractor，得到 source data 和 target data 的 feature，接著利用 PCA 和 TSNE 對 data 作降維。

因為考慮到使用 PCA fit(source data+target data 混合在一起)和 PCA 個別單獨 fit_transform source data/target data 的結果可能有所差異，因此我兩種都有做。

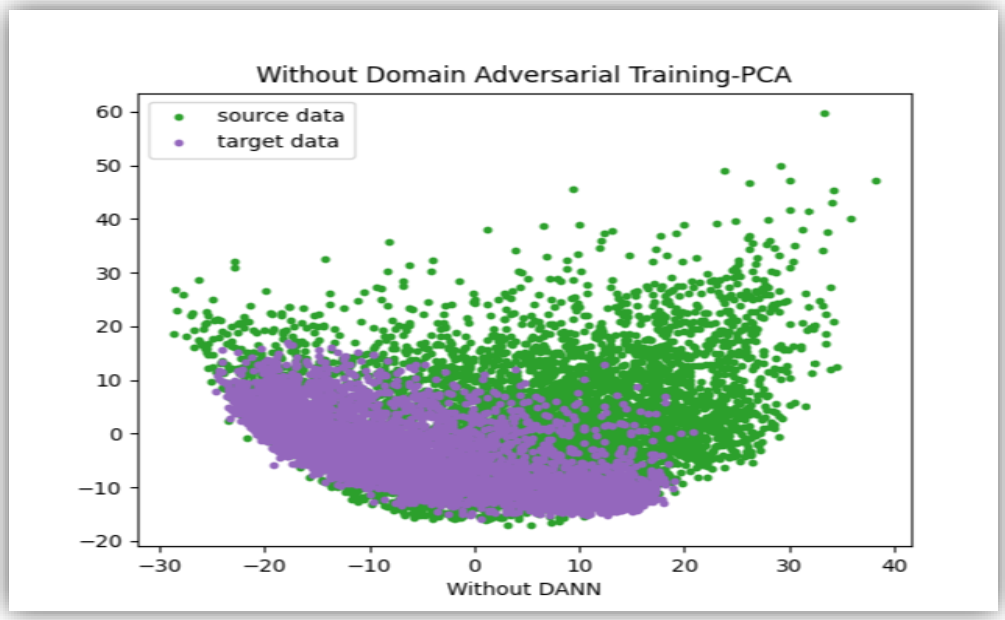
1.(未使用 domain adversarial)使用 PCA fit transform 個別對 source data、target data 降維。

```
pca = PCA(n_components=2)
source_projected = pca.fit_transform(source_feature)
pca = PCA(n_components=2)
target_projected = pca.fit_transform(target_feature)
```

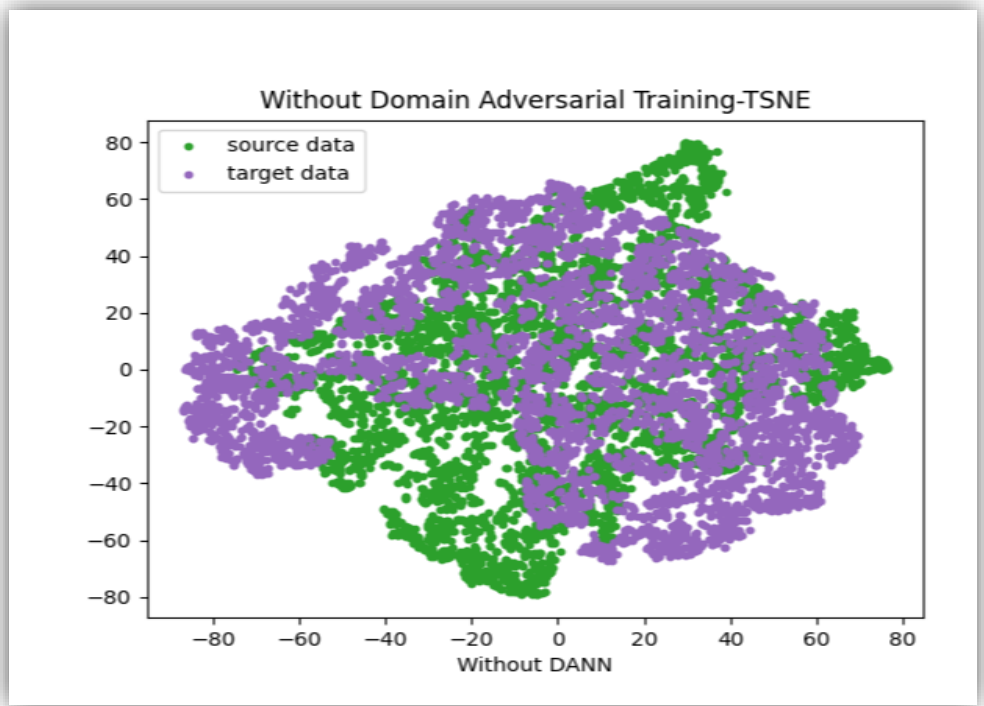


2.(未使用 domain adversarial)PCA 先對混合的 data 進行 fit，再個別進行 transform

```
pca = PCA(n_components=2)
pca.fit(all_feature)
source_projected = pca.transform(source_feature)
target_projected = pca.transform(target_feature)
```



3. (未使用 domain adversarial)使用 TSNE 進行降維



觀察:

綠色為 source data

紫色為 target data

雖然 source data 跟 target data 混合後的平均數變異數應該跟混合前有些差距，不過可以看到 PCA 的兩種結果分布都很相似，從 PCA 中得到的結果是，兩種 distribution 分布有些差異，落在相同位置處的點僅有少部分。

改將注意力放在 TSNE 降維的結果，可以清楚的看到，兩種 domain data 的 distribution 是更加分散的，這可能就是因為沒有使用 domain adversarial training，所以使得兩種 domain 無法被投射在相同處。

TSNE 中還可以看到一個比較有趣的現象，就是 source data 被分成了很多分散的小區塊，這可能是因為不同 label 所萃取出的 feature 有所不同造成的。

4. 請視覺化真實圖片以及手繪圖片通過有使用 domain adversarial training 的 feature extractor 的 domain 分布圖。(2%)

- 參數設定:

Batch size:32
Optimizer:Adam
Learning rate:0.001
Num_epoch = 200
Loss = predictor loss - lambda*domain classifier loss
Lambda 照以下式子做調整 其中 gamma 為 10，p 依據當前 epoch 由 0 逐漸變為 1

$$\lambda_p = \frac{2}{1+\exp(-\gamma \cdot p)} - 1,$$

- Model 架構:

	Type	Filter Size	Input size
Feature Extractor	Conv2D	3x3x1x64	32x32x1
	BatchNorm2d		32x32x64
	ReLU		32x32x64
	MaxPool2d		32x32x64
	Conv2D	3x3x64x128	16x16x64
	BatchNorm2d		16x16x128
	ReLU		16x16x128
	MaxPool2d		16x16x128
	Conv2D	3x3x128x256	8x8x128
	BatchNorm2d		8x8x256
	ReLU		8x8x256
	MaxPool2d		8x8x256
	Conv2D	3x3x256x256	4x4x256
	BatchNorm2d		4x4x256
	ReLU		4x4x256
	MaxPool2d		4x4x256
	Conv2D	3x3x256x512	2x2x256
	BatchNorm2d		2x2x512
	ReLU		2x2x512
	MaxPool2d		2x2x512
Label Predictor	Linear(512,512)	512	
	ReLU	512	
	Linear(512,512)	512	
	ReLU	512	
	Linear(512,10)	10	
	Linear(512,512)	512	

Domain Classifier	BatchNorm2d	512
	ReLU	512
	Linear(512,512)	512
	BatchNorm2d	512
	ReLU	512
	Linear(512,512)	512
	BatchNorm2d	512
	ReLU	512
	Linear(512,512)	512
	BatchNorm2d	512
	ReLU	512
	Linear(512,1)	512

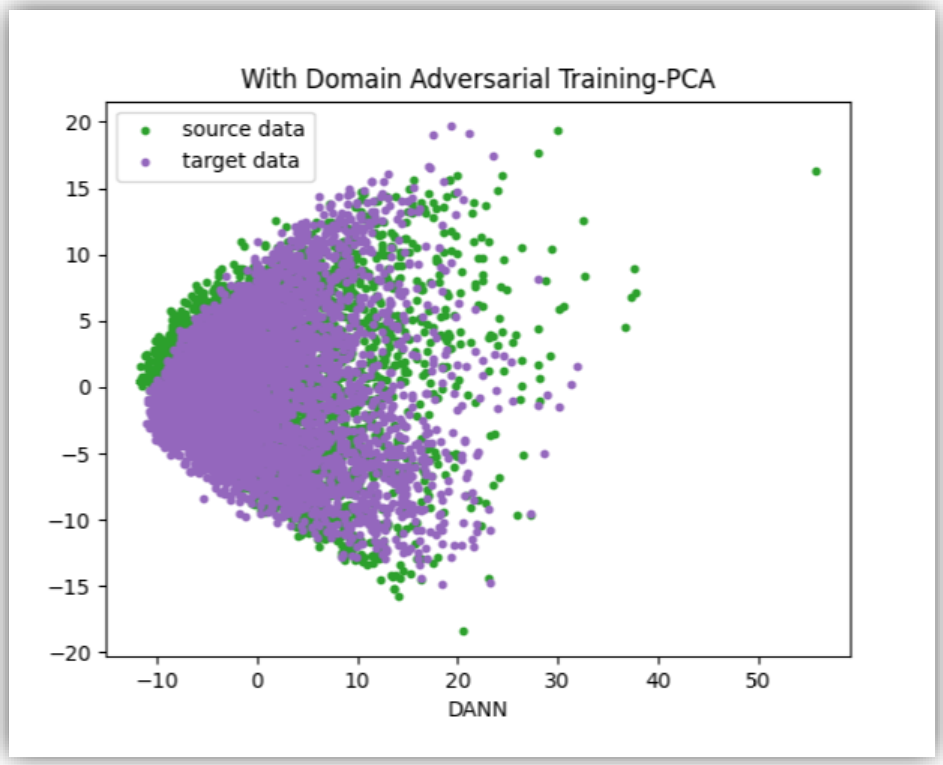
- Domain distribution:

實作方法:

基本上與上題相同，使用 feature extractor 取出 feature 後用 PCA/TSNE 做降維。

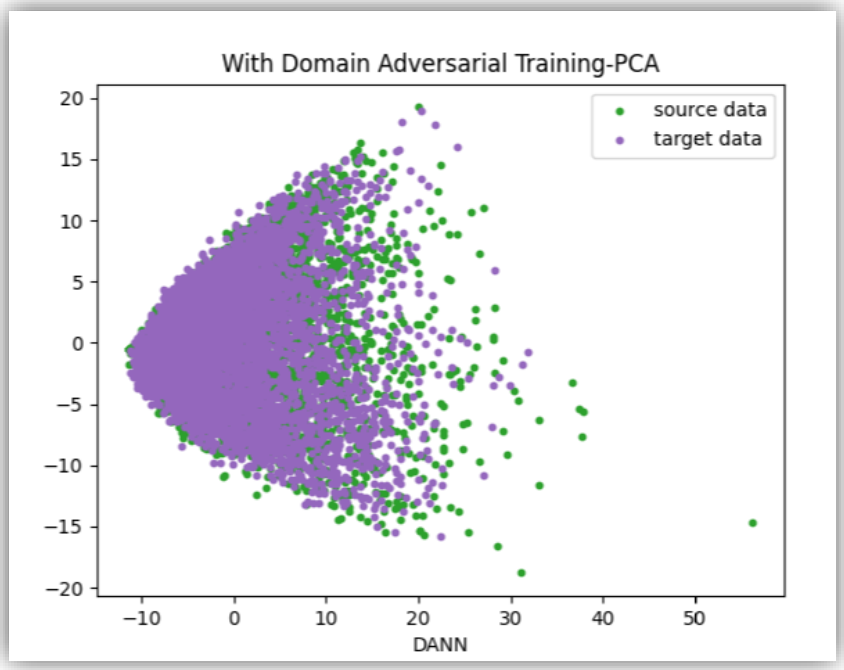
1. 使用 PCA fit transform 個別對 source data、target data 降維。

```
pca = PCA(n_components=2)
source_projected = pca.fit_transform(source_feature)
pca = PCA(n_components=2)
target_projected = pca.fit_transform(target_feature)
```

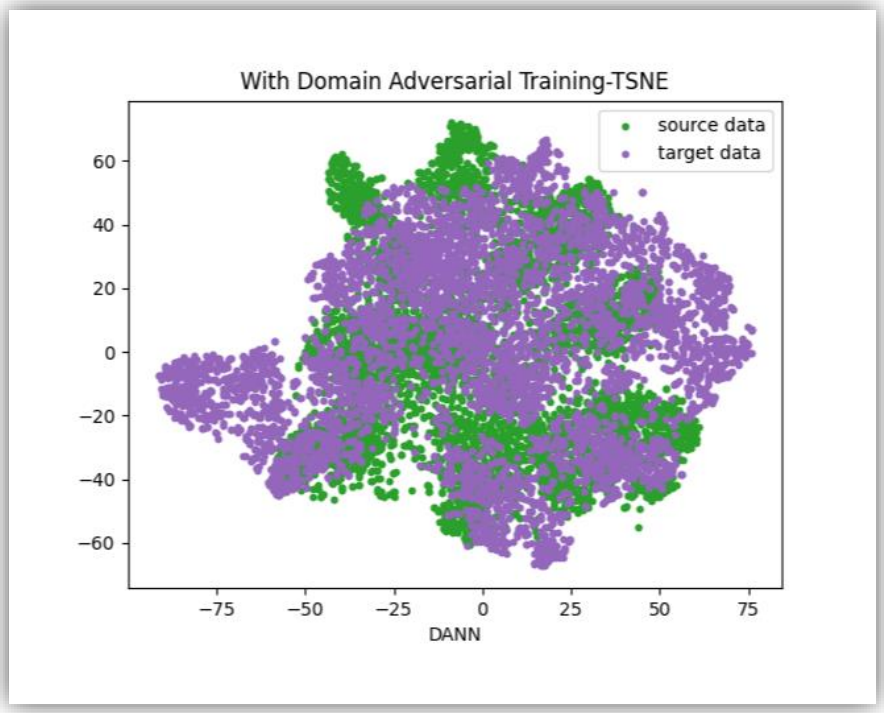


2. PCA 先對混合的 data 進行 fit，再個別進行 transform

```
pca = PCA(n_components=2)
pca.fit(all_feature)
source_projected = pca.transform(source_feature)
target_projected = pca.transform(target_feature)
```



3. 使用 TSNE 進行降維



觀察:

綠色為 source data

紫色為 target data

跟前一題得到的結果相比，可以發現降維後的結果的確重合的部分變大許多，兩種 PCA 的結果算是相似，只有少部分的點跟重合的區域差距較大。

TSNE 的部分，在上一題中結果可以看到，source data 被投影在很多不同小區塊中(很可能是因為 label 不同造成的)，但是 source data 跟 target data 卻幾乎沒有重合的區域，而在這題的結果當中，source data 跟 target data 重合的部分明顯變多了。

- 一些猜測與想法:

從這題的結果可以清楚看到 domain classifier 的確幫助了 feature extractor 去將 source data 的 feature 跟 target data 的 feature 投影在相近的地方，然而看到這個投影的結果時我開始在想一個問題，會不會 domain classifier 其實也對預測結果有負面影響?

為何這樣說呢，因為目前這些投影的點，他們彼此幾乎都黏在一起，雖然 source data/target data 投影在相近的地方，但是這些投影點卻沒有很明顯的分群。

如果能夠像 clustering 的概念一樣，把這些 feature extractor 萃取出來的 feature，依據 label 去分群的話，是否能夠讓預測結果更進一步提升呢?

然而 domain classifier 的目標就是讓 feature 的分布越相近越好，相近到分不出 domain 是最好的，但這樣的過程中，有沒有可能也造成不同 label 分布被投影到很相近的地方，讓 predictor 不好預測？

當然，就結果看起來，training accuracy 總是可以到達 90 幾%，這個問題應該是不太重要 XD。