

---

# Domain Adaptation

---

材料三 b06507002 林柏勳

電機四 b05901001 陳世豪

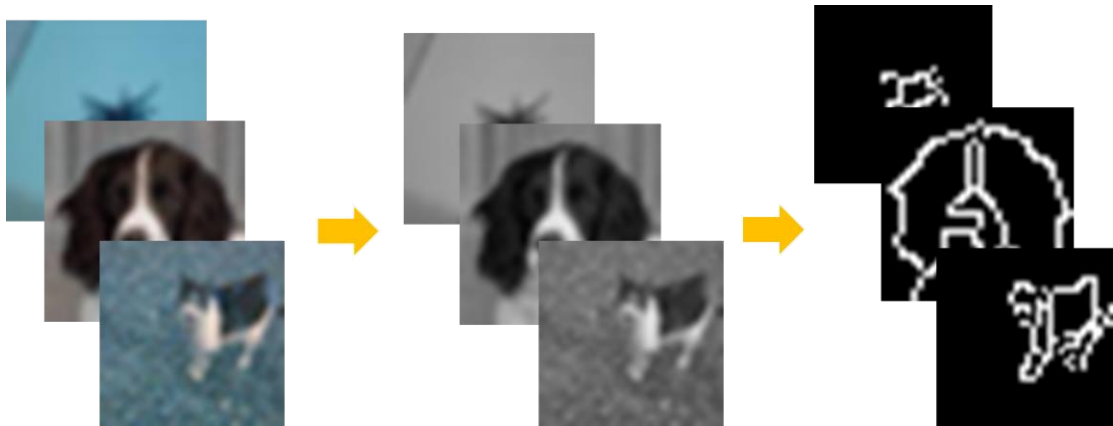
會碩二 r07722005 徐佳揚

## 1. Introduction & Motivation

領域自適應(Domain Adaptation)是一項機器學習的問題，它應用了遷移學習的技術。其處理的問題如下：我們具有兩個不同但相關的資料領域：源域(source domain)、目標域(target domain)。其中，我們僅有源域上有標記的資料，但是在目標域上卻只有未標記的資料。在本次的課題中，源域為  $32*32*3$  的實景照片，而目標域為  $28*28*1$  的手繪圖片。我們希望能訓練出一個模型，在源域跟目標域的預測都達到高準確率。

## 2. Data Preprocessing/Feature Engineering

由於本次的源域與目標域相差甚大，因此我們希望藉由預處理將兩個領域的特徵變得相近。我們將源域的影像轉為黑白之後，採用 cv2 的 canny edge detection。Canny edge detection 會針對相鄰的元素計算灰階亮度的梯度，當梯度大於一定的閾值時，該點的輸出值就會為正，以得到偵測影像邊緣的效果。由於一般人手繪時均只描繪圖像外框，因此 edge detection 將可以有效拉近源域與目標域的距離。



## 3. Model Description (At least two different models)

本課題之中，我們先後採用了兩種方法。首先是 Domain adversarial neural network(DANN)，以該方法達到近八成的準確率。為了進一步處理問題，我們擷取多個 DANN 的輸出結果，接著使用半監督式學習(semi-supervised learning)方法訓練，進一步提高正確率。以下，我們分別簡介這兩種方法。

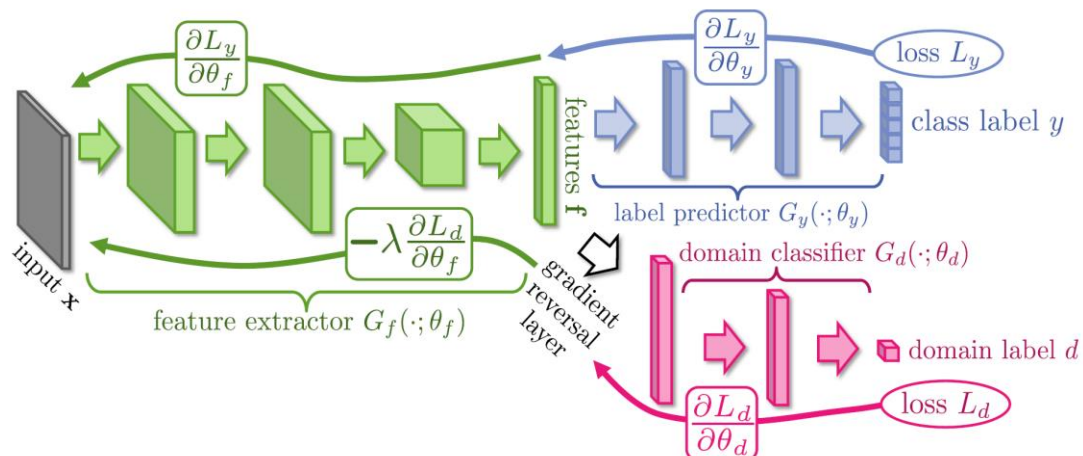
### 3.1 DANN

在一般的卷積式類神經網路(convolutional neural network, CNN)之中，前半段的卷積層可以視為抽取特徵，而後半段的全連接層才進行標記分類。若欲分類兩個以上領域的圖案，且只有其中一者有標記時，feature extractor 將只能針對一

個領域區分出明確的數個特徵，另一個領域則容易無法分成數類。

我們希望抽取 feature 時，可以消除不同領域的基礎差異，而讓每一個類別都擁有來自兩個不同領域的資料。為此，DANN 除了一般的 label predictor 以外，還新增了 domain classifier，專門判斷這筆資料來自源域還是目標域。Domain predictor 工作的同時，將該層的梯度以一個負的權重加回 feature extractor，以讓 feature extractor 逐漸消除兩個領域的差異影響。

我們以 DANN 的方法在這次的課題達成了八成左右的準確率。



我們使用的 feature extractor、label predictor 與 domain classifier 之模型如下。

#### feature extractor

layer	output
Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	[-1, 64, 32, 32]
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	[-1, 64, 32, 32]
ReLU(inplace=True)	[-1, 64, 32, 32]
MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)	[-1, 64, 16, 16]
Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	[-1, 128, 16, 16]
BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	[-1, 128, 16, 16]
ReLU(inplace=True)	[-1, 128, 16, 16]
MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)	[-1, 128, 8, 8]
Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	[-1, 256, 8, 8]
BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	[-1, 256, 8, 8]
ReLU(inplace=True)	[-1, 256, 8, 8]
MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)	[-1, 256, 4, 4]
Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	[-1, 256, 4, 4]
BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	[-1, 256, 4, 4]
ReLU(inplace=True)	[-1, 256, 4, 4]
MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)	[-1, 256, 2, 2]
Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	[-1, 512, 2, 2]

BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	[-1, 512, 2, 2]
ReLU(inplace=True)	[-1, 512, 2, 2]
MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)	[-1, 512, 1, 1]

### label predictor

layer	output
Linear(in_features=512, out_features=512, bias=True)	[-1, 512]
ReLU(inplace=True)	[-1, 512]
Linear(in_features=512, out_features=512, bias=True)	[-1, 512]
ReLU(inplace=True)	[-1, 512]
Linear(in_features=512, out_features=10, bias=True)	[-1, 10]

### domain classifier

layer	output
Linear(in_features=512, out_features=512, bias=True)	[-1, 512]
LeakyReLU(negative_slope=0.2)	[-1, 512]
BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	[-1, 512]
Linear(in_features=512, out_features=512, bias=True)	[-1, 512]
LeakyReLU(negative_slope=0.2)	[-1, 512]
BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	[-1, 512]
Linear(in_features=512, out_features=512, bias=True)	[-1, 512]
LeakyReLU(negative_slope=0.2)	[-1, 512]
BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	[-1, 512]
Linear(in_features=512, out_features=512, bias=True)	[-1, 512]
LeakyReLU(negative_slope=0.2)	[-1, 512]
BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	[-1, 512]
Linear(in_features=512, out_features=1, bias=True)	[-1, 1]
Sigmoid()	[-1, 1]

## 3.2 半監督式學習

為了進一步提升準確率，我們以 DANN 產生數個模型後擷取其預測結果，進行半監督式學習，具體的步驟如下：

- (1) 將各個模型所預測的分類完全相同的資料，視為分類信心較高的資料。
- (2) 以這些資料作為訓練集(training set)，其他資料為測試集(testing set)，設置一個 CNN 模型進行訓練。

(3) 將新訓練出來的模型加入(1)之中，重複步驟(1)

在半監督式學習的過程中，我們使用兩種模型。第一個為類似 Resnet18 的模型，第二個為類似 Resnet18 之中的第一個 block 和 linear 部分的模型。這一部分的輸出為兩種模型的 ensemble。藉此，我們進一步達到大約 83% 的準確率。

類似 Resnet18 的模型：

Layer	Output
Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))	-1, 32, 28, 28
LeakyReLU(negative_slope=0.05)	-1, 32, 28, 28
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	-1, 32, 28, 28
Conv2d(32, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))	-1, 32, 28, 28
LeakyReLU(negative_slope=0.05)	-1, 32, 28, 28
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	-1, 32, 28, 28
Conv2d(32, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))	-1, 32, 28, 28
LeakyReLU(negative_slope=0.05)	-1, 32, 28, 28
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	-1, 32, 28, 28
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)	-1, 32, 14, 14
Dropout(p=0.1, inplace=False)	-1, 32, 14, 14
Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	-1, 32, 14, 14
LeakyReLU(negative_slope=0.05)	-1, 32, 14, 14
BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	-1, 32, 14, 14
Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	-1, 32, 14, 14
LeakyReLU(negative_slope=0.05)	-1, 32, 14, 14
BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	-1, 32, 14, 14
Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	-1, 32, 14, 14
LeakyReLU(negative_slope=0.05)	-1, 32, 14, 14
BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	-1, 32, 14, 14
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)	-1, 128, 7, 7
Dropout(p=0.1, inplace=False)	-1, 128, 7, 7
Conv2d(128, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	-1, 128, 7, 7
LeakyReLU(negative_slope=0.05)	-1, 128, 7, 7
BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	-1, 128, 7, 7
Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	-1, 128, 7, 7
LeakyReLU(negative_slope=0.05)	-1, 128, 7, 7
BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	-1, 128, 7, 7
Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	-1, 128, 7, 7
LeakyReLU(negative_slope=0.05)	-1, 128, 7, 7

BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	-1, 128, 7, 7
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)	-1, 512, 3, 3
Linear(in_features=18432, out_features=512, bias=True)	-1, 512
ReLU()	-1, 512
BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	-1, 512
Linear(in_features=512, out_features=10, bias=True)	-1, 10

類似 Resnet18 之中的第一個 block 和 linear 部分的模型：

Layer	Output
Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))	-1, 32, 28, 28
LeakyReLU(negative_slope=0.05)	-1, 32, 28, 28
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	-1, 32, 28, 28
Conv2d(32, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))	-1, 32, 28, 28
LeakyReLU(negative_slope=0.05)	-1, 32, 28, 28
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	-1, 32, 28, 28
Conv2d(32, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))	-1, 32, 28, 28
LeakyReLU(negative_slope=0.05)	-1, 32, 28, 28
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	-1, 32, 28, 28
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)	-1, 32, 14, 14
Dropout(p=0.1, inplace=False)	-1, 32, 14, 14
Linear(in_features=6272, out_features=512, bias=True)	-1, 512
ReLU()	-1, 512
BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)	-1, 512
Linear(in_features=512, out_features=10, bias=True)	-1, 10

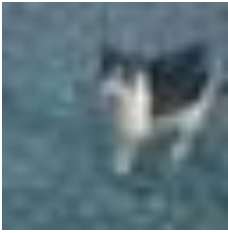
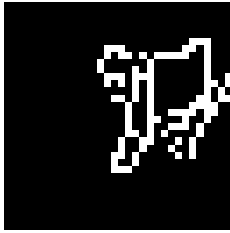

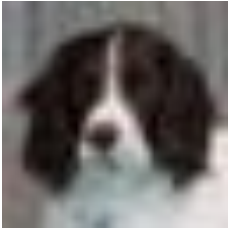
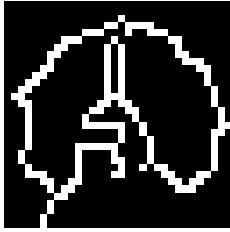
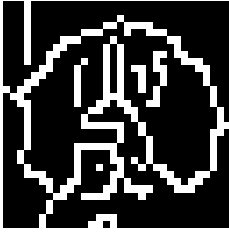
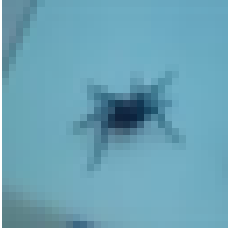
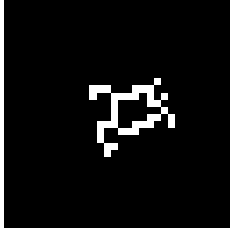
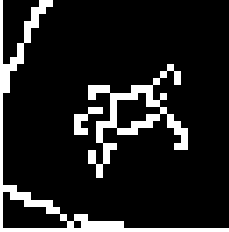
## 4. Experiment and Discussion

在訓練的過程之中，我們嘗試了多種方法。canny edge detection 的閾值能影響圖片內的梯度要超過多少時才被取為有效的邊緣，而各模型的厚度與深度也都影響到其表現。接著，我們在此介紹試驗過程中曾嘗試調整的數種內容。

### 4.1 Preprocessing: canny edge detection

預處理時，除了將影像轉換為黑白的之外，canny edge detection 也可以將照片的邊緣取出，使訓練集資料更加接近手繪圖片的特徵。我們參考了 cv2 的 canny 文檔：cv2.Canny(image, threshold1, threshold2)。Canny 算子會計算一個點在圖片的 x 與 y 方向上的亮度梯度並取兩者的方均根，而當梯度的方均根值處於兩個閾值(threshold)之間時，輸出影像的該點才會有數值。Canny 的閾值將影響我們究竟將多銳利的亮度差異取為邊緣。

下表演示了三張在兩種不同的閾值之下所得到的輸出影像。

原圖片	閾值(170,300)	閾值(30,150)
		
		
		

當閾值較低時，圖片之中較次要的顏色變化也會被取為邊緣值，如同上圖中，連放置蜘蛛的框框和狗的毛色變化也會被取為邊界。反過來說，當閾值太低時，有些主要的線條也沒辦法被取為邊界。事實上，每一個類型的圖片，其色彩的對比都不大相同，尤其貓、狗、馬、蜘蛛等動物線條較複雜，其與背景的對比也未必很明顯，或許很難找到一個閾值既能滿足簡單、對比大的圖片，又能滿足細節多、對比小的圖片。

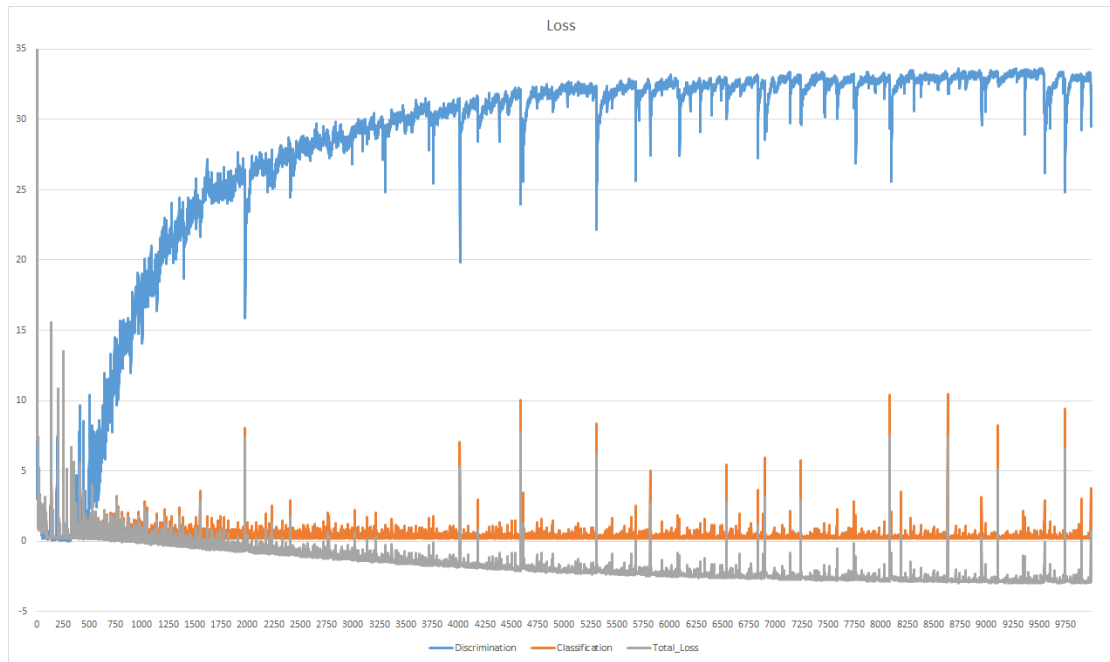
試驗過程之中，我們逐漸發現 canny 的閾值對 DANN 訓練過後的準確率影響極大。即便只改變 canny 閾值而不改變模型架構，準確率也會從四成到八成之間劇烈變化。我們花了許多時間調整、訓練並重複上傳測試，最後找到表現相對較佳的 canny 閾值。

#### 4.2 DANN

訓練 DANN 時，我們調整的內容主要有兩方面，一是訓練次數(epoch)，二是 feature extractor、label predictor 與 domain classifier 三者的模型架構。

訓練 DANN 的過程之中，其損失函數(Loss)隨訓練次數降低得相當慢。起初我們只訓練數百個 epoch，並嘗試調整各方面的模型架構，但準確率始終只有四成到五成。嘗試花費數個小時訓練上千個 epoch 之後，我們發現雖然 loss 降低緩慢，但準確率的確仍慢慢提高。我們最後發現大約要訓練 5000~10000 個 epoch，DANN 的準確率才趨於收斂。至此以後，我們每一次訓練 DANN 大約均需耗時六到八個小時。

下圖的訓練曲線，可以看出至少需要 3000 個以上的 epoch 才會使 domain classifier 的 loss 上升許多，也就是此時兩個領域的特徵差異才被消除到足夠的地步。



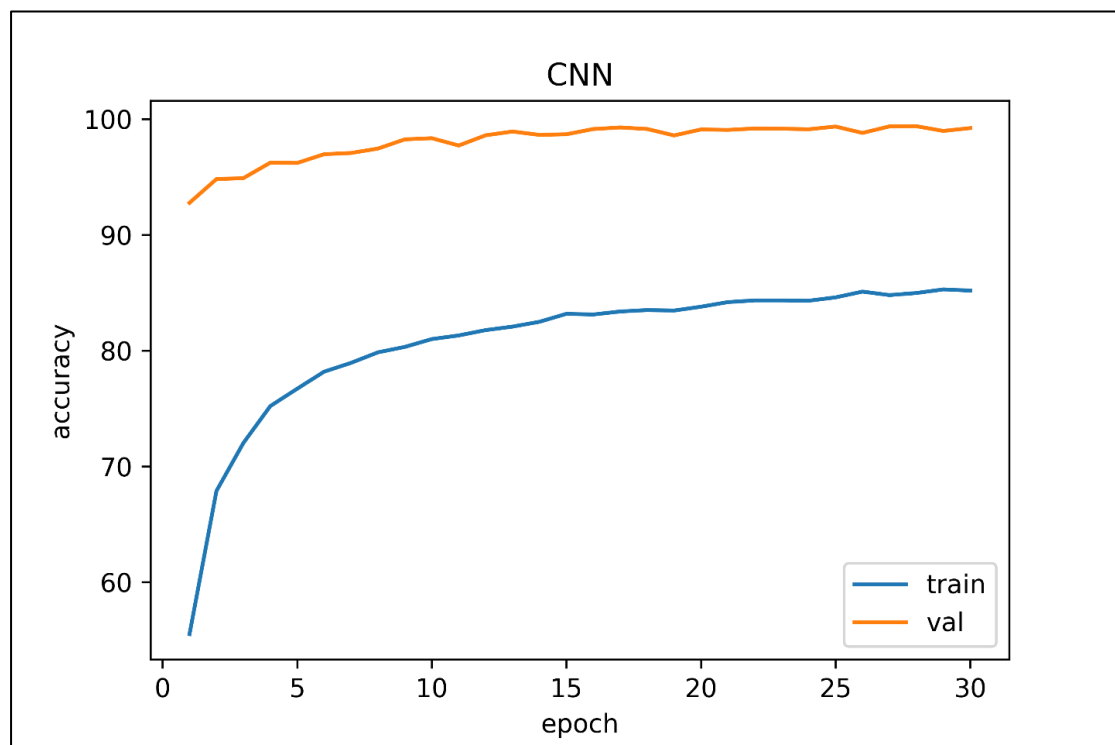
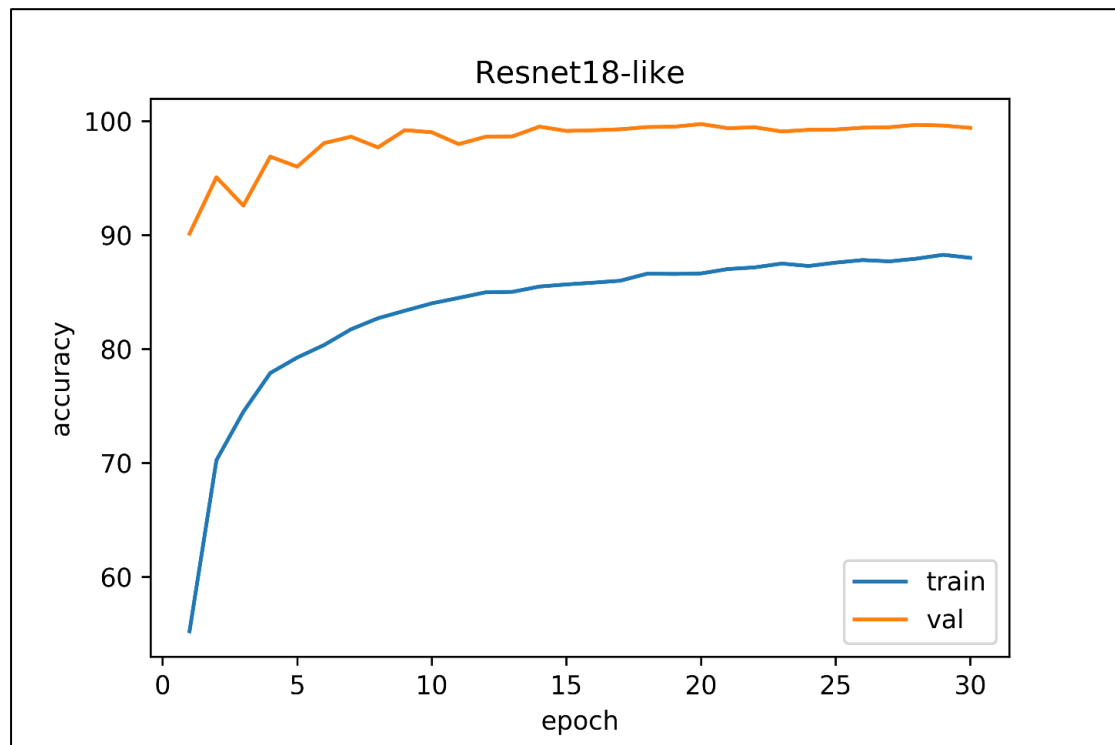
同時，我們亦嘗試調整 DANN 的各部分架構。Feature extractor 的架構為數層 Convolution layer 組成。由於圖片尺寸小，feature 尚未太複雜，因此我們嘗試了數層至十幾層 convolution layer 的結構，最後發現大約使用五層左右就能達到不錯的效果。Label predictor 與 domain classifier 均為 Full connection layer。Label predictor 亦由於圖片特徵簡單，試驗過後發現大約兩三層 512 維度即可達到不錯的效果。Domain classifier 經過試驗則發現需要五層以上才會有較好的表現。我們認為 Domain classifier 的架構需要比 Label predictor 大上許多，是因為 DANN 的 feature extractor 以及 domain classifier 會互相對抗。Domain classifier 進行分類的同時，其回傳到 feature extractor 的梯度將會被乘上負值的權重，使 feature extractor 逐步削減兩個領域的差異。我們期待對抗持續一定的訓練次數 (epoch) 之後，feature extractor 最終順利消除兩個領域的差異，此時 domain classifier 才會幾乎完全失準。為了達成此目的，feature extractor 與 domain classifier 將需要具有相近的強度。倘若 domain classifier 能力太弱，將會很快就無法分辨此時的圖片來自哪一個領域，而此時 feature extractor 仍然未順利將兩個領域的差異消除殆盡；但若 domain classifier 太強，兩者的對抗將非常難趨於收斂。

#### 4.3 半監督式學習

調整過 canny 閾值與 DANN 各架構之後，我們得到的準確率的上限大約在八成左右。為了進一步提升準確率，我們採用半監督式學習方法，將各個 DANN 物測結果之中完全相同的測試資料(約七萬張)定為假標記(pseudo label)作為下一次訓練的標記，並設立卷積式類神經網路(CNN)來針對目標域訓練，並將訓練出來的模型之交集再做為下次訓練的標記，如此反覆訓練。我們並結合了資料增強(data augmentation)將圖片隨機裁切、翻轉與正規化等。

我們採用了兩種不同的模型架構，一者類似 Resnet-18 的第一個 block 加上 full connection layer，另一者則為與 Resnet-18 類似的模型，並將此兩項模型數次得出來的結果作投票以得到最終結果。將 DANN 的結果依此方法訓練後，我們最終得到約 83% 的準確率。

下圖分別表示了兩種模型的訓練曲線，其中 train 曲線表示經過目標集經過資料增強與 dropout 的曲線，val 曲線則表示目標集的資料。





## 5. Conclusion

本次我們欲利用彩色圖片作為源域資料、手繪圖片為目標域資料，建構一個能順利將目標域分別為十類的類神經網路。首先，我們將源域圖片轉為黑白後使用 cv2 的 canny edge detection 以拉近兩個領域的特徵。接著我們以 DANN 的方式建構一組由 feature extractor、label predictor 以及 domain classifier 的類神經網路，在做分類的同時，將 domain classifier 的梯度反向以逐漸消弭兩個領域對於 feature extractor 的差異。該方法能達到八成左右的準確率。生成數個模型後，我們將其輸出完全相同的資料定為目標域的假標記，以這些標記進行半監督式學習，並將新輸出的標記與前述模型一同並列，如此反覆進行訓練。藉由此法，我們達到了 83% 左右的準確率。

藉由本文提及的 canny edge detection 以及 DANN 與半監督式學習方法，我們能有效處理領域自適應的問題，在由彩色圖片轉移到手繪圖片的分類任務上得到約 83% 的準確率。

## 6. References

- [1]Ganin, Yaroslav, et al. "Domain-adversarial training of neural networks." *Domain Adaptation in Computer Vision Applications*. Springer, Cham, 2017. 189-209.
- [2]Ganin, Yaroslav, and Victor Lempitsky. "Unsupervised domain adaptation by backpropagation." *arXiv preprint arXiv:1409.7495* (2014).