

DeepLearning Assignment 3 Report

≡ Tags

Semi-Supervised Learning

記憶體排列


深度學習模型的圖片記憶方式主要有兩種，根據這兩種排列方式的不同數據的轉換方法也要跟著改變。

1. pytorch: 使用 Channel-first 的方式排列資料 [Channel, Height, Width]
2. Channel-last: 把通道數排在最後的排列方式 [Height, Width, Channel]

Augmentation Process

Illustration of transforms — Torchvision 0.16 documentation

Package Reference

 https://pytorch.org/vision/stable/auto_examples/transforms/plot_transforms_illustrations.html#sphx-glr-auto-examples-transforms-plot-transforms-illustrations-py

資料預處理時嘗試的 augmentation method 主要以 PyTorch 網站提供的預設方法為模板，其中包含以下幾種方式：

1. Training Dataset

訓練集為了增加資料的豐富度所以進行"多變"的圖片轉換

```
transforms_train = v2.Compose([
    v2.ToImage(), # Convert to tensor, only needed if you had a PIL image
    v2.ColorJitter(), # 變更圖片的顏色亮度
    v2.RandomResizedCrop(size=(224, 224), antialias=True), # 對圖片進行隨機切割按照給定的大小重塑圖片
    v2.RandomHorizontalFlip(p=0.5), # 水平翻轉
    v2.RandomVerticalFlip(p=0.5), # 垂直翻轉
    v2.ToDtype(torch.float32, scale=True), # 資料型態轉換
    v2.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]), # 標準化圖片(mean, std 使用的值為慣例使用的標準化數值)
])
```

`v2.AutoAugment()`: 預先定義好的資料增強方式(policy)，不用自行定義每一個增強的規定。

2. Test Dataset

訓練集需要保持資料的真實性，因此只對圖片大小轉換和進行正規化增加計算速度

```
transforms_test = v2.Compose([
    v2.ToImage(), # Convert to tensor, only needed if you had a PIL image
    v2.Resize((224, 224), antialias=False), # Or Resize(antialias=True)
    v2.ToDtype(torch.float32, scale=True), # Normalize expects float input
    v2.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

以上的方法被各自定義為 `transforms_train`, `transforms_test`，分別為我們引入的資料集檔案進入訓練階段前進行資料的轉換(DataLoader)。

當 DataLoader 也定義完成，代表訓練模型所需的資料已經準備好了，下個步驟開始定義模型。

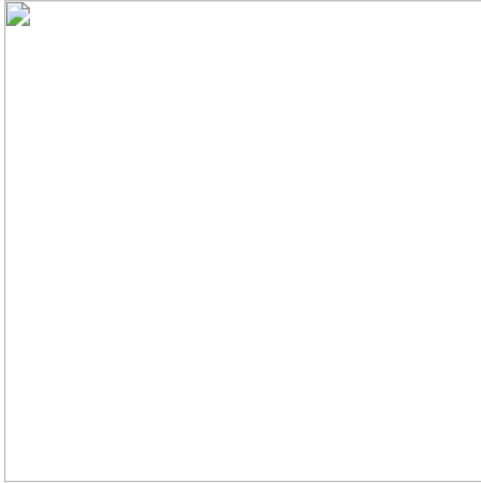
Define a CNN Model

一開始用的方法是定義好多個 `self.Conv2D` 的函數實現多個 Conv Layers，但是每個層都只使用一次使得程式碼太過冗長，改以 `Sequential` 的方式建立模型。

```
self.conv = nn.Conv2D(...) → nn.Sequential(...)
```

卷機神經網路架構





Sequential Method

根據上課教的神經網路架構來定義這次的模型，

輸入層 (Input) → 卷機層(Convolutional Layer) → 全連接層(FC Layer)

與前次作業不同的地方在於中間多出一個卷機層，其中卷機層的定義又如下所述：

【Input】 →
【Convolutional Layer 1 (input_channel, output_channel, kernel size, padding)】 →
【ReLU()】 (or other Activation functions) →
【Convolutional Layer 2】 →
【Pooling】 →
【...】

```
self.conv_layers = nn.Sequential(  
    # Conv 1  
    nn.Conv2d(in_channels=3, out_channels=12, kernel_size=3, stride=1, padding=1),  
    nn.BatchNorm2d(12),  
    nn.ELU(),  
  
    # Conv 2  
    nn.Conv2d(in_channels=12, out_channels=21, kernel_size=3, stride=1, padding=1),  
    nn.BatchNorm2d(21),  
    nn.ELU(),  
    nn.MaxPool2d(2, stride=2), # 112  
  
    # Conv 3  
    nn.Conv2d(in_channels=21, out_channels=30, kernel_size=3, stride=1, padding=1),  
    nn.BatchNorm2d(30),  
    nn.ELU(),  
    nn.MaxPool2d(2, stride=2),  
  
    # Conv 4  
    nn.Conv2d(in_channels=30, out_channels=39, kernel_size=3, stride=1, padding=1),  
    nn.BatchNorm2d(39),  
    nn.ELU(),  
    nn.MaxPool2d(2, stride=2),  
  
    # Conv 5  
    nn.Conv2d(in_channels=39, out_channels=48, kernel_size=3, stride=1, padding=1),  
    nn.BatchNorm2d(48),  
    nn.ELU(),  
  
    # Conv 6  
    nn.Conv2d(in_channels=48, out_channels=57, kernel_size=3, stride=1, padding=1),  
    nn.BatchNorm2d(57),  
    nn.ELU(),
```

```

)

self.fc_layers = nn.Sequential(
    nn.Linear(57 * 28 * 28, 2440), # fc1
    nn.BatchNorm1d(2440),
    nn.ELU(),
    nn.Linear(2440, 1440), #fc2
    nn.BatchNorm1d(1440),
    nn.ELU(),
    nn.Linear(1440, 144), # fc3
    nn.BatchNorm1d(144),
    nn.ELU(),
    nn.Dropout(0.5), # dropout
    nn.Linear(144, 5) # fc4
)

```

模型的訓練方法決定後須調整內部的參數以達到最佳的訓練結果(Higher Acc, Lower loss)。

經過十多次的提交正確率還是難以突破 0.76 的坎。

實作的過程中嘗試的幾個不同的方向，

1. **增加 Convolution Layer 的數量**從三層增加為四層，確實可以有效的提升Accuracy在每個 epoch 提高的幅度，但是overfitting 的問題也更快的出現，跟 Valid Data 擬合的速度非常快，感覺 Accuracy 沒辦法升得很高（critical point）。
2. **增加 Convolution Layer 的通道(channel)數量**，使用越多的通道數越能學習複雜的特徵。
3. **Pooling Layer**: 由於上面增加 Convolution Layer 導致擷取的特徵數增加，訓練速度非常慢，透過 Maxpooling 的方式可以逐一對圖片大小縮減，減少計算量提升訓練速度。
4. **增加 Fully Connected Layer 的層數和神經元數量**，相較於 Convolution Layer 的影響力，Fully connection layer 對於整體模型正確率的表現較**不明顯**。
5. **修改 Activation function**：從基本的 `ReLU()` 換為 Pytorch 建議的 `ELU()` 觀察是否對模型訓練有更好的表現。
6. **調整 Learning Rate**：觀察員先定義的 lr 為 0.01 訓練的圖形上發現動盪的幅度非常大，懷疑原因為 lr 的數值過大導致動盪，因此對 lr 進行修該為更小的 0.001，在學習上有更好的表現。
7. **更換 Optimizer 演算法**：比較 SGD、Adam、AdamM 表現的不同。
8. **Dropout**在層跟層之間正常會添加一個 drop 避免模型 overfitting，在自行訓練的模型加入兩層的 dropout。 [How ReLU and Dropout Layers Work in CNNs | Baeldung on Computer Science](#)。



使用較小的 $lr = 0.001$

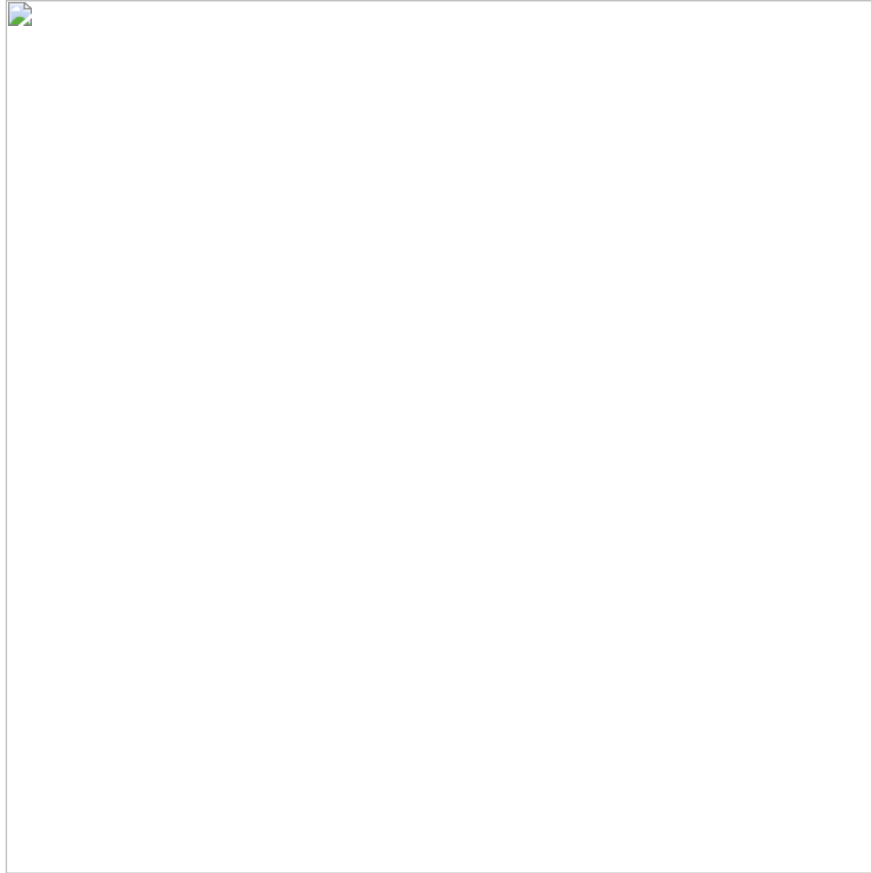


使用較小的 $lr = 0.001$





Adam(lr=0.001), 200 epochs, 以圖形看起來還可以繼續訓練未達到擬合



使用訓練好的模型將Unlabeled圖片標記

```
global unlabeled_set_list, train_set, pseudo_loader

t = torch.tensor(threshold, dtype=torch.float32).to(device)
remove_index, index = [], 0

model.eval()

softmax = nn.Softmax(dim=-1)

new_labeled_image, new_labeled_label = 0, 0

sub_set_list = []

# Iterate over the dataset by batches.
if(len(unlabeled_set_list)>0):
    for img in tqdm(unlabeled_set_list):
        #####
        #      TODO:                                     #
        #      1. Forward the data, Using torch.no_grad() accelerates the forward process      #
        #      2. Obtain the probability distributions by applying softmax on logits              #
        #      3. Filter the data with threshold                                                #
        #      4. Combine the labeled training data with the pseudo-labeled data              #
        #      to construct a new training set. then removed                                  #
        #      5. the unlabeled data from unlabeled_set_list                                  #
        #      hint: ConcatDataset                                                                #
        #####
        with torch.no_grad():
            outputs = model(img.unsqueeze(0).to(device)) # 獲得預測
            probabilities = softmax(outputs) # 獲得img對照各個class的機率
            # Check if any class probability exceeds the threshold for this image
            prob_max = torch.max(probabilities)
            #print(img.device.type, prob_max.device.type, t.device.type)
            if prob_max > t:
                new_labeled_image = img
                new_labeled_label = torch.argmax(probabilities, 1).item()
```

```

        sub_set = ([new_labeled_image, new_labeled_label])
        sub_set_list.append(sub_set)
        #train_set = ConcatDataset([train_set, sub_set])

        remove_index.append(index)
        #print(sub_set)
        index += 1

#####
#                               End of your code                               #
#####
remove_index.reverse()
for i in remove_index:
    del unlabeled_set_list[i]
train_set = ConcatDataset([train_set, sub_set_list])

```

透過逐一辨識每一張照片，給予其對應的標記，把這些資料儲存在 list 之中，最後依照 train_set 的格式把 list 與 train_set 兩者結合。



定義 get_pseudo_labels 函式遇到如何把標記好的圖片以正確格式Concat 到 train_set 中，因為經過 PyTorch ConcatDataset 操作過後的 dataset 資料型態會被轉換，並且無法再次進行 concat 因此經過多次的嘗試發現，python 原生的 list 可以直接 concat 到 train_set 中，迴避資料型態錯誤以及無法重複連接的問題。

接下來這個經過結合的資料集通過 DataLoader 的轉換即可作為新的模型的訓練資料。

```

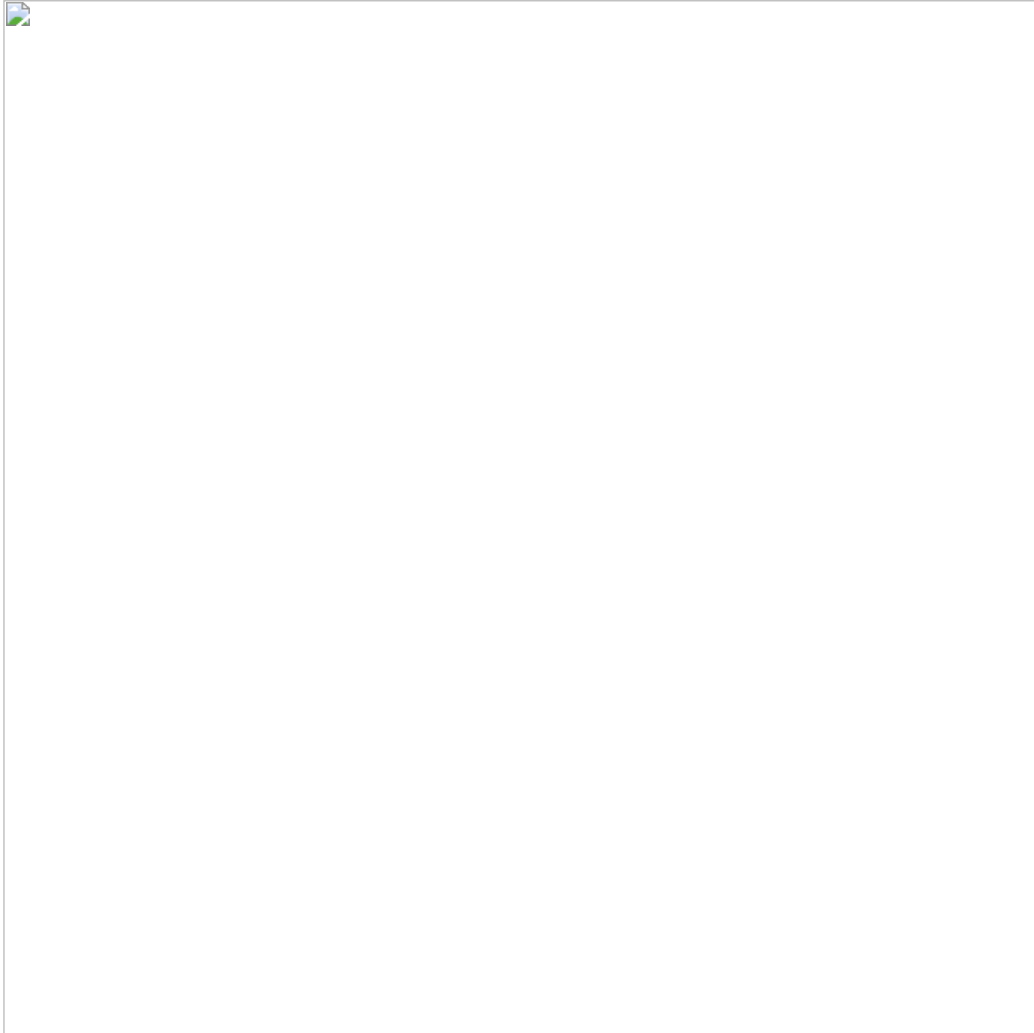
get_pseudo_labels(model, 0.6)
train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True, pin_memory=False, drop_last=True)

```

輸出的模型正確率應該要比單使用已標記資料的表現更好

使用偽標記資料增加資料的數量對於模型有正向的結果，原先僅使用為數不多的labeled 資料進行訓練模型的準確率約落在0.7左右，偽標記資料的加入可使模型提升表現至0.8以上。

使用 threshold = 0.65 篩選圖片並增加訓練模型的資料，



結果為 train = 0.87923, valid = 0.77072 相較於單純使用已標記的 train data 有更高的準確度。

10/31 紀錄

後續將 optimizer 換為 Adam，並且增加 Convolutional Layer 的通道數，並增加 Pooling 次數成功突破 0.8 的正確率，訓練的 epoch 約為 200 個。