B093040024
郭晏涵

深度視覺 Final Project

notebook 執行過程

環境建置及平台：

使用 Google Colaboratory 作為執行平台，執行環境內包含兩個 CPU。

```
[20] !cat /proc/cpuinfo | grep model\ name

model name      : Intel(R) Xeon(R) CPU @ 2.20GHz
model name      : Intel(R) Xeon(R) CPU @ 2.20GHz
```

框架選擇：

使用 Pytorch-Lightning 作為深度學習框架。

資料前處理：

資料前處理主要在 SegmentationData()中完成，首先透過輸入的文字檔讀取 RGB 影像，並將輸入影像調整成 240*240 的尺寸。Annotations instance 的部分需另外藉由 SEG_LABELS_LIST 將資料集的 lable 對應到影像的 R 值，G 值和 B 值則設為 0。

```
SEG_LABELS_LIST = [
        {"id": 0 , "name": "viod",          "rgb_values": [0,  0,  0]},
        {"id": 1 , "name": "bed",           "rgb_values": [1,  0,  0]},
        {"id": 2 , "name": "windowpane",    "rgb_values": [2,  0,  0]},
        {"id": 3 , "name": "cabinet",       "rgb_values": [3,  0,  0]},
        {"id": 4 , "name": "person",        "rgb_values": [4,  0,  0]},
        {"id": 5 , "name": "door",          "rgb_values": [5,  0,  0]},
        {"id": 6 , "name": "table",         "rgb_values": [6,  0,  0]},
        {"id": 7 , "name": "curtain",       "rgb_values": [7,  0,  0]},
        {"id": 8 , "name": "chair",         "rgb_values": [8,  0,  0]},
        {"id": 9 , "name": "car",           "rgb_values": [9,  0,  0]},
        {"id": 10 , "name": "painting",     "rgb_values": [10,  0,  0]},
        {"id": 11 , "name": "sofa",         "rgb_values": [11,  0,  0]},
        {"id": 12 , "name": "shelf",        "rgb_values": [12,  0,  0]},
        {"id": 13 , "name": "mirror",       "rgb_values": [13,  0,  0]},
        {"id": 14 , "name": "armchair",     "rgb_values": [14,  0,  0]},
        {"id": 15 , "name": "seat",         "rgb_values": [15,  0,  0]},
```

超參數設定：

Learning rate 設定為 0.01、batch size 設為 16、損失函數則使用 CrossEntropyLoss()，其中 reduction 設為 mean，返回 16 筆資料平均的 loss。

```
hparams = {
        "height":240, "width":240, "loss":torch.nn.CrossEntropyLoss(ignore_index=0, reduction='mean'),
        "lr":1e-2, "train_dataset":train_data, "batch_size":16
}
```

模型建構：

__init__()的部分包括一個 AlexNet 的 feature extractor(作為神經網路的架構)、一個 output channel 為 101 的 convolution layer (用於將輸入分類為 1 ~ 100 的已知 class 和 0 的 viod class)，和一個 upsampling layer (放大並增加圖像的訊息)。

```
class SegmentationNN(pl.LightningModule):

    def __init__(self, num_classes=101, hparams=None):
        super().__init__()
        self.hp = hparams

        self.features = models.alexnet(pretrained=True).features

        for param in self.features.parameters():
            param.requires_grad = False

        self.conv_to101 = nn.Conv2d(256, num_classes, 1)
        self.upsample = nn.Upsample(size=(self.hp["height"], self.hp["width"]), mode='bilinear', align_corners=True)
```

forward()的部分則是利用 Sequential 函數，依據初始化所宣告的 object 來實現神經網路，經過 AlexNet 後將輸出轉為 101 個 class，最後使用 upsampling 將影像放大。

```
    def forward(self, x):
        """
        Use for inference only.
        Inputs:
        - x: PyTorch input Variable
        """

        final = nn.Sequential(self.features, self.conv_to101, self.upsample)
        x = final(x)

        return x
```

接著在模型中定義三個使用 pl.Trainer 進行訓練時必須包含的 functions：

training_step()為每個 batch 的處理函數。功能是依序將 batch 的輸入(x) 傳入 forward()計算得到預測結果 (y_hat)，再將結果與正確的 lable (y) 傳入損失函數中計算並回傳 loss。

```
    def training_step(self, batch, batch_idx):
        """
        The complete training loop.
        """
        x, y, _, _, _ = batch
        y_hat = self.forward(x)
        loss = self.hp["loss"](y_hat, y)
        return loss
```

train_dataloader()是將傳入的 train_dataset 當中的 20210 筆訓練資料切割成 size 為 16 的 mini batch，因此一個 epoch 總共要訓練 1264 個 steps。Shuffle 在這裡設定為 True，用於每次訓練時隨機變換 batch 的順序。

```
    def train_dataloader(self):
        """
        Wrap the dataset defined.
        This is the dataloader that the Trainer fit() method uses
        """
        return DataLoader(self.hp["train_dataset"], batch_size=self.hp["batch_size"], shuffle=True)

```

configure_optimizers()則是對目前訓練所的參數進行優化，降低 loss 的大小，這裡使用 Adam 作為 optimizer。

```
    def configure_optimizers(self):
        """
        Define optimizers and LR schedulers.
        """
        return torch.optim.Adam(self.parameters(), lr=self.hp["lr"])
```

訓練方式/過程：

將建構完成的模型放入 Pytorch-Lightning 的 Trainer 行訓練，總共訓練 20 個 epoch。由於一個 epoch 的訓練時間長達四至六小時，且 Cloab 連線時常處於不穩定的狀態，因此採用接力訓練的方式，訓練完一個 epoch 便將 weight 儲存，並從雲端硬碟載入 weight 放入 model 中進行下一個 epoch 的訓練。

```
model = SegmentationNN(hparams=hparams)
model.load_state_dict(torch.load('weight21.pt'))
```

```
<All keys matched successfully>
```

```
trainer = pl.Trainer(max_epochs=1, gpus=0, fast_dev_run=False, log_every_n_steps=1)

trainer.fit(model)
```

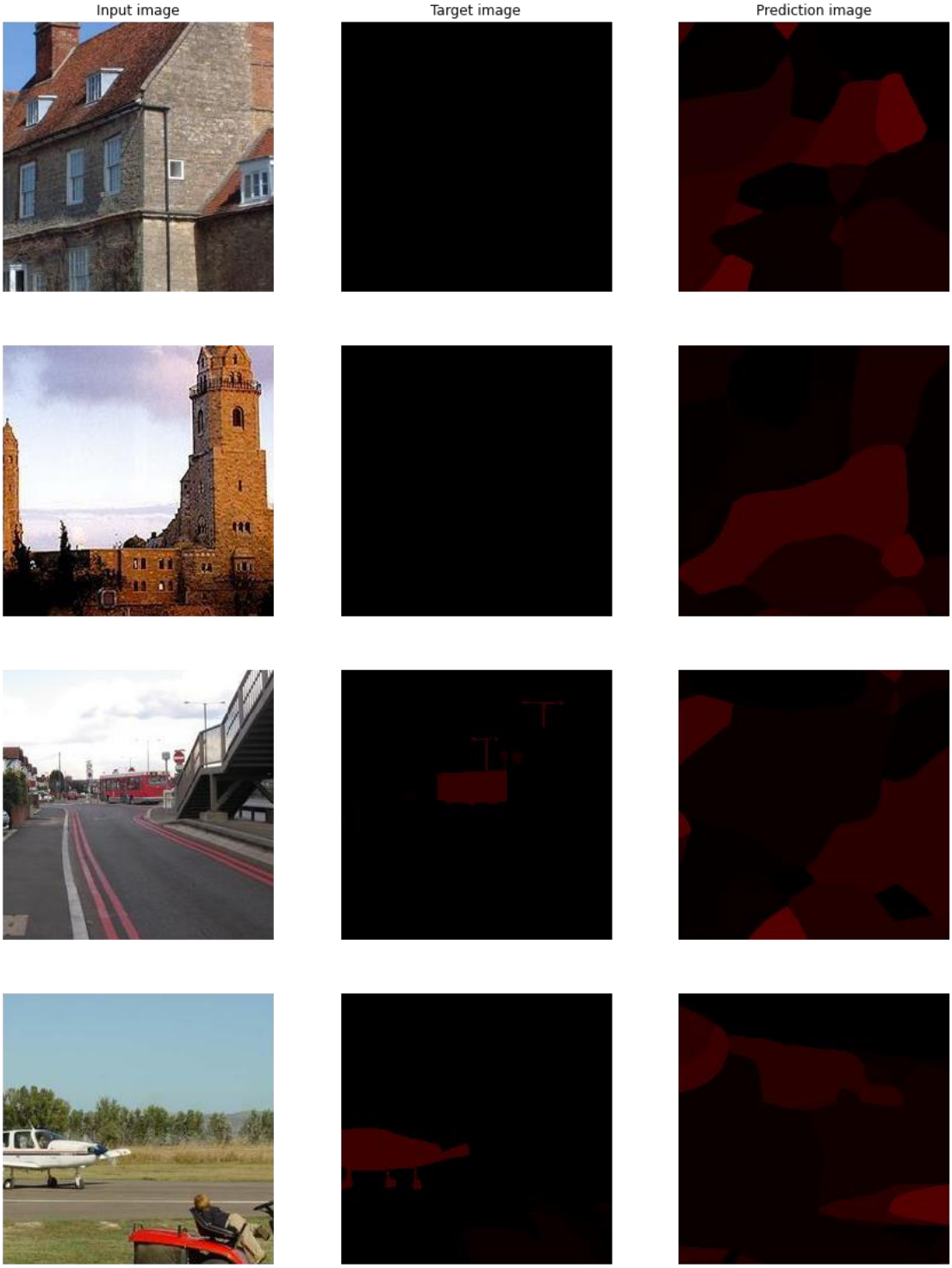
```
GPU available: False, used: False
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs

  | Name       | Type       | Params
-----------------------------------------
0 | features   | Sequential | 2.5 M
1 | conv_to101 | Conv2d     | 26.0 K
2 | upsample   | Upsample   | 0
-----------------------------------------
26.0 K    Trainable params
2.5 M     Non-trainable params
2.5 M     Total params
9.983     Total estimated model params size (MB)
Epoch 0: 100% 1264/1264 [4:41:05<00:00, 13.34s/it, loss=2.95, v_num=23]
```

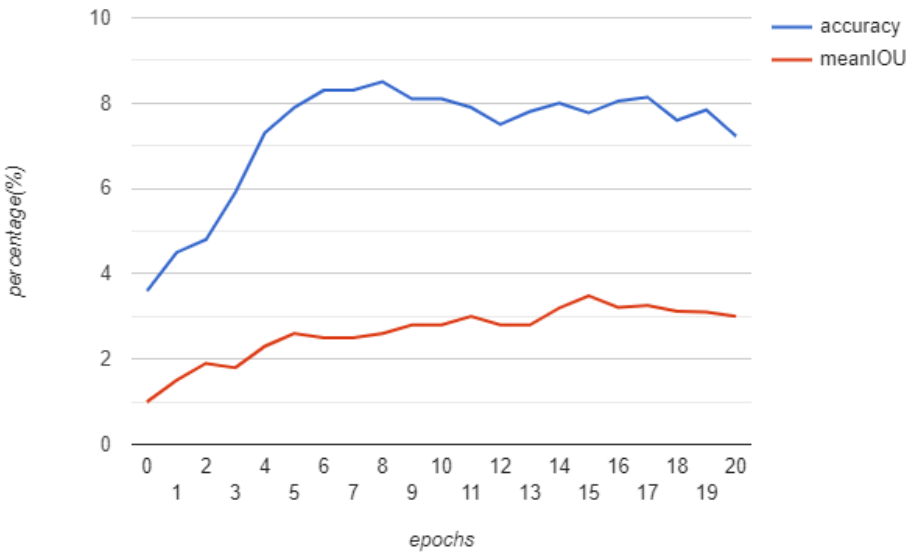
```
torch.save(model.state_dict(), "weight22.pt")
```

訓練結果圖：

以 validation 中四張影像的訓練結果為例。



驗證平均 IOU 與準確度：



我的努力和我為訓練做的改變：

我在訓練上遇到的第一個困難點是，一個 epoch 的訓練時間過長，而免費版 Colab 的 GPU 又有限時，其連續可使用的時間測試下來不到六小時，不足以負擔如此龐大的訓練量。在無法購買專業版環境，又想保留資料的完整性，不減少資料集的情況下，我採取的作法是將 batch size 稍微增大，這樣做有三個優點：

1. 提高的 ram 的利用率，並改善計算的平行化效率。
2. 跑完一次 epoch 所需的 step 減少，加快相同數據量的處理速度。
3. 在一定範圍內，batch size 越大，相鄰的 batch 差異就越小，訓練時梯度較平滑。

另一個改善方法是降低模型的大小，選用參數較少的 AlexNet 作為主要架構，以減少計算量。

第二個問題點則是訓練的準確度過低，我花費許多時間修改模型，嘗試增加非線性的 ReLu 層，或加上 Dropout 層，避免訓練時不會過度依賴某一些神經元，也曾經思考是否是因為 AlexNet 的參數過少導致 underfitting，因此改用層數較多的 ResNet 進行訓練。但以上方式的成效都不如預期，不但讓訓練時間加倍，meanIOU 也沒有顯著提升，因此最終決定放棄修改的版本。因此課程結束後我會重新複習老師上課所教的內容，瞭解我在期末報告所缺少的重要知識。