B074020010 林宜璇 Final report

一、資料前處理

首先，先從雲端硬碟捷徑解壓縮檔案，把 images 資料夾裡的檔名存至 list 中。

```
data_root = "/content/drive/MyDrive/Colab Notebooks/Final Project/images"
train_file_path = os.listdir(f'{data_root}/training/')
val_file_path = os.listdir(f'{data_root}/validation/')

#get image name
train_img_name, val_img_name = [], []
for img in train_file_path:
    train_img_name.append(img.replace(".jpg", ""))

for img in val_file_path:
    val_img_name.append(img.replace(".jpg", ""))
```

以及把 annotations_instances 裡的檔名存至 list 中，並且和 images 比對有無缺少的檔案。可以發現在 train 資料中，annotations_instances 的訓練集有缺少的圖片，需要將它們去除，不可將多餘的圖片寫入讀取檔名的 txt 中。

```
#compare img names
data_root = "/content/drive/MyDrive/Colab Notebooks/Final Project/annotations_instance"
ann_train_file_path = os.listdir(f'{data_root}/training/')
ann_val_file_path = os.listdir(f'{data_root}/validation/')

#get image name
ann_train_img_name, ann_val_img_name = [], []
for img in ann_train_file_path:
    ann_train_img_name.append(img.replace(".png", ""))

for img in ann_val_file_path:
    ann_val_img_name.append(img.replace(".png", ""))

#find the differences (img in train dataset but not in annotation dataset)
remove_train = list(set(train_img_name) - set(ann_train_img_name))
remove_val = list(set(val_img_name) - set(ann_val_img_name))
print(remove_train)
print(remove_val)
```

```
['ADE_train_00006934', 'ADE_train_00017086', 'ADE_train_00006253', 'ADE_train_00014153', 'ADE_train_00013308', 'ADE_train_00001637',
[]
```

取得既有 train 的圖片也有 annotations_instances 的圖片檔名。

```
train_img_name = set(train_img_name) and set(ann_train_img_name)
```

因為資料集過大，因此只取 5000 筆圖片進行訓練寫入 txt 檔中。

```
data_root = "/content/drive/MyDrive/Colab Notebooks/Final Project/images"
train_img_name = [name+'\n' for name in train_img_name]
val_img_name = [name+'\n' for name in val_img_name]

#write into txt file
f = open(f'{data_root}/training.txt', "w")
for i in train_img_name[:5000]: #train 5000 images
    f.write(i)
f.close()
```

```
f = open(f'{data_root}/validation.txt', "w")
for i in val_img_name:
    f.write(i)
f.close()
```

Segmentation dataset（修改自 HW10 的 Segmentation_dataset.py）：
幫圖片製作 SEG_LABELS_LIST，資料集中總共有 100 類，只取 rgb channel 中的 "r"（其餘的 channel 都設為 0，如下簡圖）。

```
SEG_LABELS_LIST = [
        {"id": -1,  "name": "void",          "rgb_values": [0, 0, 0]},
        {"id": 0,   "name": "bed",           "rgb_values": [1, 0, 0]},
        {"id": 1,   "name": "windowpane",    "rgb_values": [2, 0, 0]},
        {"id": 2,   "name": "cabinet",       "rgb_values": [3, 0, 0]},
        {"id": 3,   "name": "person",        "rgb_values": [4, 0, 0]},
        {"id": 4,   "name": "door",          "rgb_values": [5, 0, 0]},
        {"id": 5,   "name": "table",         "rgb_values": [6, 0, 0]},
        {"id": 6,   "name": "curtain",       "rgb_values": [7, 0, 0]},
        {"id": 7,   "name": "chair",         "rgb_values": [8, 0, 0]},
        {"id": 8,   "name": "car",           "rgb_values": [9, 0, 0]},
        {"id": 9, "name": "painting",        "rgb_values": [10, 0, 0]},
```

這部分因為我們只需訓練出圖片的 red channel 需出現正確的類別，因此將圖片（變數 img）其他 channel 轉為 0。由於計算 cross entropy 時 id 必須是從 0 到 99（-1 代表的是 void），而 rgb_values 從 1 到 100，因此需特別留意 target_labels[mask] 及 target_labels 的運算。

```
def get_item_from_index(self, index):
        to_tensor = transforms.ToTensor()
        img_id = self.image_names[index].replace('.jpg', '')
        img = Image.open(os.path.join(self.root_dir_name, "images", self.img_type,
                                                              img_id + '.jpg')).convert('RGB')
        r, g, b = img.split()
        r = r.convert('RGB')
        center_crop = transforms.CenterCrop(240)
        img = center_crop(r)
        img = to_tensor(img)

        target = Image.open(os.path.join(self.root_dir_name, "annotations_instance", self.img_type,
                                                                  img_id + '.png'))
        target = center_crop(target)
        target = np.array(target, dtype=np.int64)

        target_labels = target[..., 0]
        for label in SEG_LABELS_LIST:
                mask = np.all(target == label['rgb_values'], axis=2)
                target_labels[mask] = label['id'] + 1

        target_labels = torch.from_numpy(target_labels.copy())
        target_labels -= 1
        return img, target_labels
```

Trainset 就是 training.txt 中所包含的圖檔名；testset 是 validation.txt 中所包含的圖檔名。
Validation set 是從 trainset 中隨機分割 20% 出來，而 test set 是用 validation.txt 中的圖檔。

```
data_dir = "/content/drive/MyDrive/Colab Notebooks/Final Project/images"
trainset, testset = load_data(data_dir)

test_abs = int(len(trainset) * 0.8)
train_subset, val_subset = random_split(
                trainset, [test_abs, len(trainset) - test_abs])

trainloader = torch.utils.data.DataLoader(
                train_subset,
                batch_size=32,
                shuffle=True,
                num_workers=8)
valloader = torch.utils.data.DataLoader(
                val_subset,
                batch_size=32,
                shuffle=True,
                num_workers=8)

testloader = torch.utils.data.DataLoader(
                testset,
                batch_size=32,
                shuffle=True,
                num_workers=8)
```

二、模型的建立

Model 1: SegmentationNN

這個模型使用 transfer learning，搭配 resnet 50 和 兩層 fcn 的結構，且為了避免梯度爆炸使用了 xavier_normal 並且使用 hyperparameter tuning 來調整參數 l1。

```
class SegmentationNN(nn.Module):
    def __init__(self, num_classes=100, l1=512):
        super(SegmentationNN, self).__init__()
        model = models.resnet50(pretrained=True)
        self.pretrained = torch.nn.Sequential(*(list(model.children())[:-1]))
        self.fcn = nn.Sequential(
            nn.Conv2d(2048, l1, kernel_size=1),
            nn.LeakyReLU(inplace=True),
            nn.Dropout(),
            nn.Conv2d(l1, num_classes, kernel_size=1)
        )
        torch.nn.init.xavier_normal_(self.fcn[0].weight, gain=1)
        torch.nn.init.xavier_normal_(self.fcn[3].weight, gain=1)


    def forward(self, x):
        out = self.pretrained(x)
        out = self.fcn(out)
        out = F.interpolate(out, size=x.size()[2:])

        return out
```

Model 2: SegmentationNN2

這個模型延續了 HW10 的模型架構——三層卷積網路加上 fully convolutional network，不過，這次還使用了 ray tune 來替 5 個參數做 hyperparameter learning。

```
class SegmentationNN2(nn.Module):
    def __init__(self, num_classes=100, l1=None, l2=None, l3=None, l4=None, l5=None):
        super(SegmentationNN2, self).__init__()
        self.conv = nn.Sequential(
                #layer 1
                nn.Conv2d(3, l1, kernel_size=32, stride=1),
                nn.BatchNorm2d(l1),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2),
                #layer 2
                nn.Conv2d(l1, l2, kernel_size=16, stride=1),
                nn.BatchNorm2d(l2),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2),
                #layer 3
                nn.Conv2d(l2, l3, kernel_size=7, stride=1),
                nn.BatchNorm2d(l3),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2, stride=2),
        )
```

```
        self.fcn = nn.Sequential(
            nn.Conv2d(l3, l4, kernel_size=2),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Conv2d(l4, l5, kernel_size=1),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Conv2d(l5, num_classes, kernel_size=1)
        )
```

```
def forward(self, x):
    out = self.conv(x)
    out = self.fcn(out)
    out = F.interpolate(out, size=x.size()[2:])

    return out
```

Model 3: SegmentationNN3

建立了 Unet 的模型，並使用 4 個參數做 hyperparameter tuning（這個模型用到 5 個參數的話，也就是 decoder 和 encoder 再新增一層會 cuda out of memory）。

```
class double_conv(nn.Module):
    def __init__(self, input, output, dropout=False):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(input, output, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(output, output, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
        )

    def forward(self, x):
        return self.conv(x)
```

```
class SegmentationNN3(nn.Module):
    def __init__(self, num_classes=100, l1=128, l2=256, l3=512, l4=1024):
        super(SegmentationNN3, self).__init__()

        self.decoder1 = double_conv(3, l1)
        self.decoder2 = double_conv(l1, l2)
        self.decoder3 = double_conv(l2, l3)
        self.decoder4 = double_conv(l3, l4)

        self.maxpool = nn.MaxPool2d(2)
        self.upsample = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)

        self.encoder1 = double_conv(l3 + l4, l3)
        self.encoder2 = double_conv(l2 + l3, l2)
        self.encoder3 = double_conv(l2 + l1, l1)

        self.last = nn.Conv2d(l1, num_classes, 1)
```

```python
def forward(self, x):
        out1 = self.decoder1(x)
        out11 = self.maxpool(out1)
        out2 = self.decoder2(out11)
        out21 = self.maxpool(out2)
        out3 = self.decoder3(out21)
        out31 = self.maxpool(out3)
        out4 = self.decoder4(out31)
        out5 = self.upsample(out4)
        out6 = torch.concat([out5, out3], dim=1)
        out7 = torch.concat([self.upsample(self.encoder1(out6)), out2], dim=1)
        out8 = torch.concat([self.upsample(self.encoder2(out7)), out1], dim=1)
        out9 = self.encoder3(out8)
        out10 = self.last(out9)

        return out10
```

以上建立了三種模型，我將比較這兩種模型訓練 30 個 epoch 結果後，選擇一個更適合的模型來做長時間的訓練。

三、Ray tune hyperparameters

根據 hyperparameter_tuning_tutorial.ipynb[1]的使用教學，在 net 中可放入要測試的參數。同時為了試著解決梯度爆炸的問題，我在訓練時加入了 torch.nn.utils.clip_grad_norm_ （但梯度還是經常會爆炸）。在訓練完不同的參數後，validation 會顯示針對該參數所呈現的模型 accuracy 及 loss。我透過這兩個指標來選擇適合的參數。

```python
def train_cifar(config, checkpoint_dir=None, data_dir=None):
        net = SegmentationNN(l1=config["l1"])
        #,l2=config["l2"],l3=config["l3"],l4=config["l4"],l5=config["l5"]
```

```python
# Validation loss
val_loss = 0.0
val_steps = 0
epoch_steps = 0
for i, data in enumerate(valloader, 0):
        with torch.no_grad():
                inputs, targets = data
                inputs, targets = inputs.to(device), targets.to(device)

                loss = 0
                outputs = net(inputs)
                for x in range(inputs.shape[0]):
                    a = outputs[x].reshape(100, -1).transpose(1, 0) #shape:(57600, 100)
                    b = targets[x].reshape(-1) #shape:(57600)
                    loss += criterion(a, b).cpu().numpy()
                epoch_steps += 1

_, preds = torch.max(outputs, 1)
targets_mask = targets >= 0
val_acc = np.mean((preds == targets)[targets_mask].data.cpu().numpy())
val_loss = loss / epoch_steps
print("val_loss:", val_loss, "val_acc:", val_acc)

with tune.checkpoint_dir(epoch) as checkpoint_dir:
        path = os.path.join(checkpoint_dir, "checkpoint")
        torch.save((net.state_dict(), optimizer.state_dict()), path)

tune.report(loss=val_loss, accuracy=val_acc)
```

在 main 的部分，config 中可以填入想要測試的參數範圍、learning rate、batch size...等。其中，我觀察到當 batch size 越大時，準確度會有所提高，但是當我選擇 batch size=64 時進行訓練時，colab 會顯示 cuda out of memory 的問題，考量到這個問題，我的 batch size 最大只能選擇 32。最後使用 tune.run()便可執行 hyperparameter tuning，選擇最適合的模型參數。

```python
from ray.tune.session import checkpoint_dir
from functools import partial
def main(num_samples=10, max_num_epochs=10, gpus_per_trial=2):
    data_dir = os.path.abspath("/content/drive/MyDrive/Colab Notebooks/Final Project/images/")
    checkpoint_dir = os.path.abspath("/content/drive/MyDrive/Colab Notebooks/Final Project/checkpoints")
    load_data(data_dir)
    config = {
        "l1": tune.sample_from(lambda _: np.random.randint(500, 700)), #2 ** np.random.randint(6, 10)
        "l2": tune.sample_from(lambda _: np.random.randint(4000, 5000)),
        #"l3": tune.sample_from(lambda _: np.random.randint(256, 512)),
        #"l4": tune.sample_from(lambda _: np.random.randint(1024, 1200)),
        #"l5": tune.sample_from(lambda _: np.random.randint(1024, 2048)),
        "lr": tune.loguniform(1e-4, 1e-2),
        "batch_size": tune.choice([16, 32])
    }
    scheduler = ASHAScheduler(
        metric="accuracy",
        mode="min",
        max_t=max_num_epochs,
        grace_period=1,
        reduction_factor=2)
    reporter = CLIReporter(
        #parameter_columns=["l1", "lr", "batch_size"], #"l2",
        metric_columns=["loss", "accuracy", "training_iteration"])
    result = tune.run(
        partial(train_cifar, data_dir=data_dir, checkpoint_dir=checkpoint_dir),
        resources_per_trial={"cpu": 2, "gpu": gpus_per_trial},
        config=config,
        num_samples=num_samples,
        scheduler=scheduler,
        progress_reporter=reporter)

if __name__ == "__main__":
    # You can change the number of GPUs per trial here:
    main(num_samples=10, max_num_epochs=3, gpus_per_trial=1)
```

四、模型 ray tuning 結果

Model 1: SegmentationNN

Trial name	status	loc	batch_size	l1	lr	loss	accuracy	training
train_cifar_c9013_00000	TERMINATED	172.28.0.2:6594	32	64	0.000233706	nan	0.335287	
train_cifar_c9013_00001	TERMINATED	172.28.0.2:6945	16	64	0.00109399	nan	0.00247564	
train_cifar_c9013_00002	TERMINATED	172.28.0.2:7283	32	64	0.00116815	1.142	0	
train_cifar_c9013_00003	TERMINATED	172.28.0.2:7546	32	128	0.0100408	nan	0.0175451	
train_cifar_c9013_00004	TERMINATED	172.28.0.2:7798	32	256	0.000696113	nan	0.48059	

可以發現當 batch_size 為 32 時、l1 為 256、learning rate 為 0.000696113 為 model 1 最佳的參數。

Model 2: SegmentationNN2

loc	batch_size	l1	l2	l3	l4	l5	lr	loss	accuracy	training_iteration
172.28.0.2:2735	16	32	16	256	512	1024	0.089893	nan	0	2
172.28.0.2:3216	32	8	128	256	512	2048	0.0562411	nan	0.11987	1
172.28.0.2:3526	16	16	64	128	512	8192	0.0102152	nan	0.180734	2
172.28.0.2:3993	32	4	16	256	1024	2048	0.000287734	nan	0.108112	1
172.28.0.2:4237	32	32	64	512	1024	8192	0.0236085	nan	0.0758537	1
172.28.0.2:4492	16	32	32	256	256	1024	0.000713937	nan	0.0765088	1
172.28.0.2:4735	32	4	64	512	2048	1024	0.000204222	nan	0.267071	1
172.28.0.2:4979	16	32	32	128	256	8192	0.00305118	0.489402	0.154222	1
172.28.0.2:5219	32	8	32	512	256	1024	0.0410977	nan	0.114848	2
172.28.0.2:5665	16	8	128	512	256	4096	0.00048489	nan	0.129322	2

可以發現當 batch_size 為 32 時、l1=4、l2=64、l3=512、l4=2048、l5=1024 且 learning rate 為 0.000204222 時為 model 2 最佳的參數。

Model 3: SegmentationNN3

這部分忘記截到圖了，但是最後的結果顯示當 batch_size 為 32 時、l1=40、l2=206、l3=415、l4=1176 且 learning rate 為 0.000138391 時為 model 3 最佳的參數。

五、模型 training 結果

為了方便檢測測試集資料的結果，我對測試集進行 mean iou 的評估。跑 mean iou 相當費時，因此在訓練過程中就不評估這項指標了，只在測試集時跑（P.S 測試集是 validation 資料夾的圖片）。

```python
def miou(output, targets, class_num=100):
    total_iou, length = 0, 0
    for i in range(1, class_num+1):
        output_ind = (output == i) #[False, True, False,  False]
        target_ind = (targets == i) #[True, True, False, False]

        if (sum(target_ind)): #若target有該class則計算
            inter = sum(output_ind & target_ind)
            union = sum(output_ind | target_ind)

            iou = inter  / union
            total_iou += iou
            length += 1

    return total_iou/length
```

Model 1: SegmentationNN

在經過兩次 15 個 epoch (30 epoch)的結果：

```
epoch 10 :
train_loss: nan train_acc: 0.5523377969416547
val_loss: nan val_acc: 0.5906017718937335

epoch 11 :
train_loss: nan train_acc: 0.6246231037193151
val_loss: nan val_acc: 0.4927932058770758

epoch 12 :
train_loss: nan train_acc: 0.5327294683700765
val_loss: nan val_acc: 0.6062918992486873

epoch 13 :
train_loss: nan train_acc: 0.6829203174680226
val_loss: nan val_acc: 0.5660667081324833

epoch 14 :
train_loss: nan train_acc: 0.4897436060932608
val_loss: nan val_acc: 0.6271133934635228

epoch 15 :
train_loss: nan train_acc: 0.5056478071433507
val_loss: nan val_acc: 0.3620415751225283
test_loss: nan test_acc: 0.30357718337093137 mean_iou: tensor(0.0166, device='cuda:0')
```

Model 2: SegmentationNN2

在經過兩次 15 個 epoch (30 epoch)的結果：

```
epoch 10 :
train_loss: nan train_acc: 0.1655572598142358
val_loss: nan val_acc: 0.3033223645748075

epoch 11 :
train_loss: nan train_acc: 0.17388555342965847
val_loss: nan val_acc: 0.08280220406156502

epoch 12 :
train_loss: nan train_acc: 0.14267489752193863
val_loss: nan val_acc: 0.16348292643470524

epoch 13 :
train_loss: nan train_acc: 0.16823050233821088
val_loss: nan val_acc: 0.23790575653352747

epoch 14 :
train_loss: nan train_acc: 0.2645241075331924
val_loss: nan val_acc: 0.33844984457604427

epoch 15 :
train_loss: nan train_acc: 0.19310986928194065
val_loss: nan val_acc: 0.1860479498649735
test_loss: nan test_acc: 0.2448988340192044 mean_iou: tensor(0.0097, device='cuda:0')
```

Model 3: SegmentationNN3

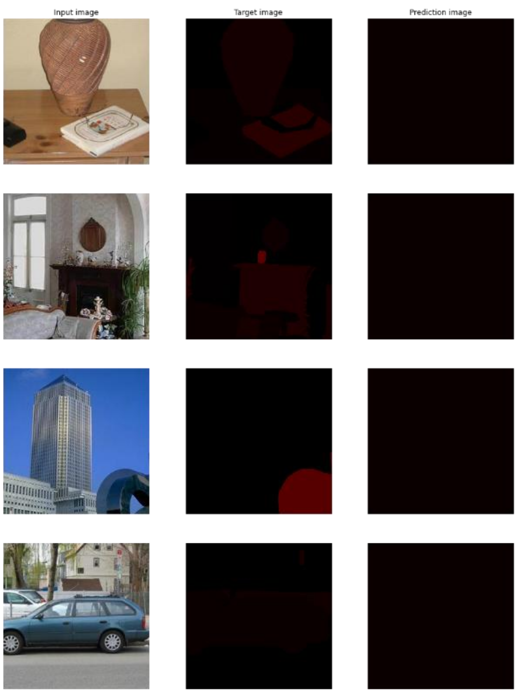
在經過兩次 15 個 epoch (30 epoch)的結果：

```
epoch 10 :
train_loss: nan train_acc: 0.15725790502788492
val_loss: nan val_acc: 0.09989852294508964

epoch 11 :
train_loss: nan train_acc: 0.23569810081988213
val_loss: nan val_acc: 0.14845258110716733

epoch 12 :
train_loss: nan train_acc: 0.10382695442749369
val_loss: nan val_acc: 0.07869474890751486

epoch 13 :
train_loss: nan train_acc: 0.22872621645399857
val_loss: nan val_acc: 0.07965435500536905

epoch 14 :
train_loss: nan train_acc: 0.13553603319009908
val_loss: nan val_acc: 0.03790322580645161

epoch 15 :
train_loss: nan train_acc: 0.16416495688930569
val_loss: nan val_acc: 0.21644538035737226
```

```
test_loss: nan test_acc: 0.10428961111724555 mean_iou: tensor(0.0046, device='cuda:0')
```

六、模型最終的選擇

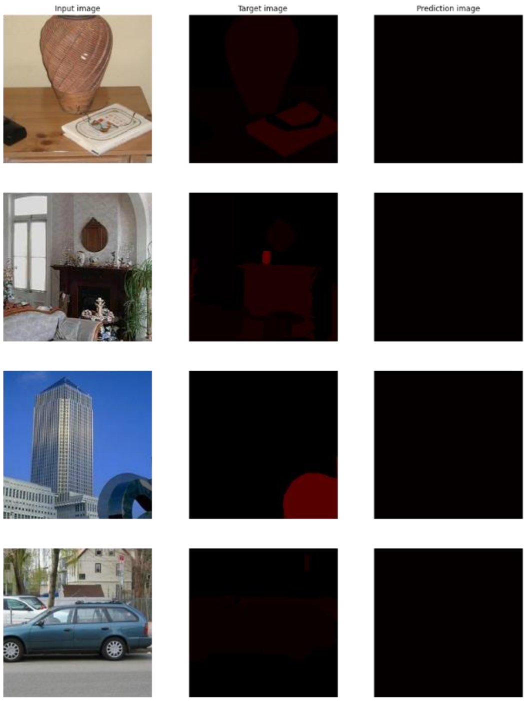
由上可發現，model 1 有最佳的模型表現。下圖為經過 45 epoch 的測試集輸出結果，在下方範例中，其大部分面積的"rgb_values"為 [9, 0, 0]，對應到車子的類別，因此我將繼續訓練該模型，讓他可以表現更好。



最後，模型經過 100 epoch 的訓練結果如下：

```
test_loss: nan test_acc: 0.32745271719778585 mean_iou: tensor(0.0121, device='cuda:0')
```

其大部分面積的"rgb_values"為 [4, 0, 0]，類別從原本的車子變成對應到人的類別。相較於 model 2 預測出的結果會出現許多不同的類別，model 1 傾向預測出較少雜訊。如果運算資源允許的話，或許我的 image segmentation 模型要再建的更深一些，或是加上 self attention。



七、參考資料

[1] https://colab.research.google.com/github/pytorch/tutorials/blob/gh-pages/_downloads/c24b93738bc036c1b66d0387555bf69a/hyperparameter_tuning_tutorial.ipynb#scrollTo=ADI0qQaeHChs

[2] https://github.com/milesial/Pytorch-UNet

[3] https://github.com/aladdinpersson/Machine-Learning-Collection/blob/master/ML/Pytorch/image_segmentation/semantic_segmentation_unet/model.py

[4] https://zhuanlan.zhihu.com/p/89588946