機器視覺作業報告

HW4-semantic segmentation

姓名:陳伸蓉
學號:0957119
日期:2023/6/11

方法

資料切割及資料前處理:

在這邊我將存有資料路徑的矩陣拆分成訓練資料(600)、驗證資料(150)及測試資料(250)。之所以會一開始就多拆了驗證資料是因為我認為在訓練的過程中驗證資料很重要，若沒有驗證資料就容易會有過擬和或者是模型性能評估不準等問題。

```
np.random.seed(9999)
idx = np.random.permutation(1000)+1
training_idx = idx[:750]
testing_idx  = idx[750:]

training_img=[]
training_label=[]

test_img=[]
test_label=[]

for i in training_idx:
    training_img.append(image_set[i])
    training_label.append(y_label[i])
for i in testing_idx:
    test_img.append(image_set[i])
    test_label.append(y_label[i])

path_to_training_img=np.array(training_img[:600])
path_to_training_label=np.array(training_label[:600])
path_to_valid_img=np.array(training_img[600:])
path_to_valid_label=np.array(training_label[600:])
print("train ",len(path_to_training_img))
print("val ",len(path_to_valid_label))

path_to_test_img=np.array(test_img)
path_to_test_label=np.array(test_label)
print("test ",len(path_to_test_label))
```

```
train  600
val  150
test  250
```

本來我的圖片大小是設置 256*256，但後來在訓練過程中發生了 GPU 記憶體的容量不足的問題。所以在這邊我把圖片的大小都調整成 128*128。

```
image_size=(128,128)
num_calsses=59
batch_size=16
```

製作 generator 來載入資料

```python
class img_deal(tf.keras.utils.Sequence):

    def __init__(self, batch_size,img_size,input_img_paths,targetlab_img_paths):
        self.batch_size=batch_size
        self.img_size = img_size
        self.input_img_paths = input_img_paths
        self.targetlab_img_paths = targetlab_img_paths

    def __len__(self):#總樣本數/batch_size=batch數
        return len(self.targetlab_img_paths) // self.batch_size

    def __getitem__(self, idx):
        low = idx * self.batch_size#開頭inedx
        high = min(low + self.batch_size, len(self.input_img_paths))#結尾index
        batch_input_img_paths = self.input_img_paths[low:high]
        batch_targetlab_img_paths = self.targetlab_img_paths[low:high]

        #x.shape=(16, 128, 128, 3)
        x=np.zeros((batch_size,)+self.img_size+(3,),dtype="float32")
        for j,path in enumerate(batch_input_img_paths):
            img = load_img(path, target_size=self.img_size, interpolation="bilinear")
            temp=img
            x[j]=img

        #y.shape=(16, 128, 128, 59)
        #有59種類別
        y=np.zeros((batch_size,)+self.img_size+(59,),dtype="float32")
        print("yshape " ,y.shape)
        for j,path in enumerate(batch_targetlab_img_paths):
            # 使用SciPy的loadmat函數載入.mat檔案
            mat_data = scipy.io.loadmat(path)
            # 提取[groundtruth]數組
            pixel_data = mat_data['groundtruth']
            # 將NumPy陣列轉換為PIL圖像物件
            img = Image.fromarray(pixel_data)
            img=img.resize(image_size,resample=Image.NEAREST)
            img_mask = np.array(img)
            img_mask = np.eye(59)[img_mask]#one hot encoding
            img_mask = img_mask.astype(np.float32)
            y[j]=img_mask
        return x,y
```

這是我第一次實作 generator，所以一開始在實作時不太知道該如何下手，後來請教過同學後才成功製作出來。在這邊比較值得注意的地方是當我在調整圖片大小時，因 photo 要考慮相鄰像素之間的距離和亮度權重，所以插值法用”bilinear";而換到標示資料(annotations)，時為了保證標籤的正確性，則要使用 NEAREST，來避免內插時發生錯誤。

模型架構:

一、VGG:

在這邊我嘗試著仿照 U-Net 網路架構並利用預訓練模型來當編碼器進行下採樣。而 U-Net 強調 skip layers 技巧，會讓上採樣部分也會接收到下採樣部分相同大小那層的輸出。一開始我參考的是老師示範的程式碼，所以先以 VGG 來嘗試。

```python
def upsample(filters, size):
    return [tf.keras.layers.Conv2DTranspose(filters, size, strides=2,padding='same'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.ReLU()]
```

這邊的上採樣進行了放大兩倍的操作，這是因為 VGG 在下採樣時每次會將長寬大小各縮小一半，而為了讓 skip connection 可以順利連結，我們需要確認在 concatenate 時 x 跟 skip 的大小相同，這樣才能夠將上下採樣拼接起來。

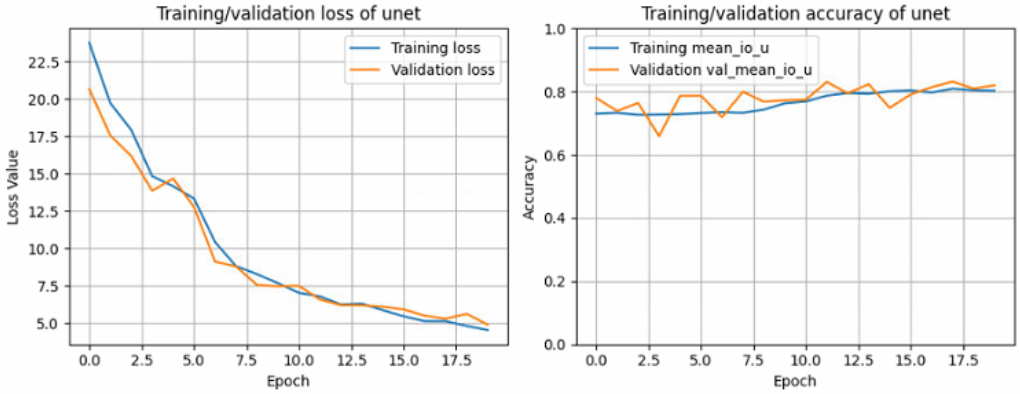
VGG(1):

```
def def_unet(classes,height,width,channels,base_model,layer_names,name):
    base_model = tf.keras.Model(inputs=base_model.input,outputs = [base_model.output]+
                                                    [base_model.get_layer(na).output for na in reversed(layer_names
    base_model.summary()
    inputs = tf.keras.layers.Input(shape=(height,width,channels))
    skips  = base_model(inputs)
    x      = skips[0]

    # U型的底部處理
    x      = tf.keras.layers.Conv2D(512,(3,3),padding='same',activation='relu')(x)
    for layer in upsample(512,(3,3)):
        x=layer(x)
    #upsampling network
    for ch,skip in zip([256,128,64,32],skips[1:-1]):
        x = tf.keras.layers.Concatenate()([x,skip])
        for layer in upsample(ch,(3,3)):
            x=layer(x)

    x = tf.keras.layers.Concatenate()([x,skips[-1]])
    x = tf.keras.layers.Conv2D(96,(1,1),padding='same')(x)
    #最後輸出
    x = tf.keras.layers.Conv2D(classes,(3,3),padding='same',activation='softmax')(x)

    unet = tf.keras.Model(inputs=inputs,outputs=x,name=name)
    unet.summary()
    base_model.trainable = False
    unet.compile(loss=tf.keras.losses.CategoricalCrossentropy(),optimizer='adam'#要用CategoricalCrossentropy!!!!!!!!!!
                 ,metrics=[tf.keras.metrics.MeanIoU(num_classes=59)])
    return unet

vgg16_layer_names = [
    'block1_conv2', # 1
    'block2_conv2', # 1/2
    'block3_conv3', # 1/4
    'block4_conv3', # 1/8
    'block5_conv3', # 1/16
]

model = def_unet(59,128,128,3,tf.keras.applications.VGG16(include_top = False),vgg16_layer_names,'unet-vgg16')
```

在這邊我透過 callback 將最好的結果儲存起來。

```
checkpoint  =[tf.keras.callbacks.ModelCheckpoint("hw4_VGG2.h5",monitor='val_loss',save_best_only=True)]
model_history=model.fit(train_gen, epochs= 20,
                        validation_data= val_gen,
                        callbacks=checkpoint)#[DisplayCallback()])
```

在這邊時 loss 最低的超過 4.5，而 val_loss 更是超過 0.8。



但因為走勢是往下的所以我有再多跑 20 個 epoch，但最後 loss 卻停在了 0.86 左右，而 val_loss 則是 2.0 多。

在這裡我利用 tf.keras.metrics.MeanIoU 來計算我的預測結果準確率。再把他們存起來。

首先先指定有 59 種類別。

```
iou_metric = tf.keras.metrics.MeanIoU(num_classes=num_calsses)
```

這邊有一個需要注意的地方是我的預測結果(my_pre)及真實結果(true_pre)要先轉成 one hot encoding 的形式才可以拿來計算 IOU，而在這邊我除了儲存 iou 值以外也存了該數值的 index。

```
iou_metric.update_state(true_pre, my_pre)
iou = iou_metric.result().numpy()
iou_metric.reset_states() #清除，以免累加
iou_val.append((iou,idx))
```

因在這裡我將該陣列由小到大排，所以可以直接取最大最小值，而透過 sorted_iou_val[index][1]的方式可以快速地得到他在原陣列的位置，來用於後續查詢及顯示圖片。

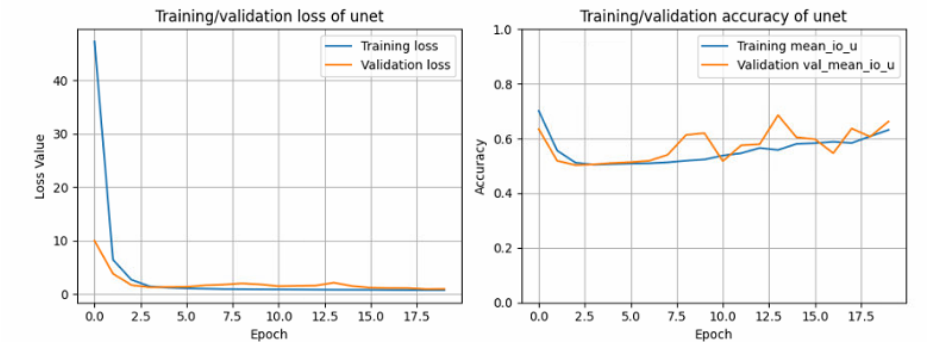
```
sorted_iou_val = sorted(iou_val, key=lambda x: x[0])
now_ious = [iou for iou, _ in iou_val]
average_iou = np.mean(now_ious)

print("max_iou:",sorted_iou_val[249][0])
print("average_iou:",average_iou)
print("min_iou:",sorted_iou_val[0][0])
```

VGG(2):

後來想到剛剛的結果不好有可能是因為底層模型的 trainable 為 false 的原因，所以這邊我將其改成 true，讓他可以再訓練，也多加了一層的捲積。

```
def def_unet(classes,height,width,channels,base_model,layer_names,name):
    base_model = tf.keras.Model(inputs=base_model.input,outputs = [base_model.output]+
                                                    [base_model.get_layer(na).output for na in reversed(layer_names)])
    base_model.summary()
    inputs = tf.keras.layers.Input(shape=(height,width,channels))
    skips  = base_model(inputs)
    x      = skips[0]

    # U型的底部處理
    x      = tf.keras.layers.Conv2D(512,(3,3),padding='same',activation='relu')(x)
    for layer in upsample(512,(3,3)):
        x=layer(x)
    #upsampling network
    for ch,skip in zip([256,128,64,32],skips[1:-1]):
        x = tf.keras.layers.Concatenate()([x,skip])
        for layer in upsample(ch,(3,3)):
            x=layer(x)

    x = tf.keras.layers.Concatenate()([x,skips[-1]])
    x = tf.keras.layers.Conv2D(128,(3,3),padding='same',activation='relu')(x)
    x = tf.keras.layers.Conv2D(96,(1,1),padding='same')(x)
    #最後輸出
    x = tf.keras.layers.Conv2D(classes,(3,3),padding='same',activation='softmax')(x)

    unet = tf.keras.Model(inputs=inputs,outputs=x,name=name)
    unet.summary()
    base_model.trainable = True
    unet.compile(loss=tf.keras.losses.CategoricalCrossentropy(),optimizer='adam'#要用CategoricalCrossentropy!!!!!!!!!!
                 ,metrics=[tf.keras.metrics.MeanIoU(num_classes=59)])
    return unet

vgg16_layer_names = [
    'block1_conv2', # 1
    'block2_conv2', # 1/2
    'block3_conv3', # 1/4
    'block4_conv3', # 1/8
    'block5_conv3', # 1/16
]

model = def_unet(59,128,128,3,tf.keras.applications.VGG16(include_top = False),vgg16_layer_names,'unet-vgg16')
```

下圖走勢雖然看起來比剛剛好，但當我多跑幾圈時，他的 loss 最低也只降到了 0.57，而 val_loss 也還是高達 0.73。



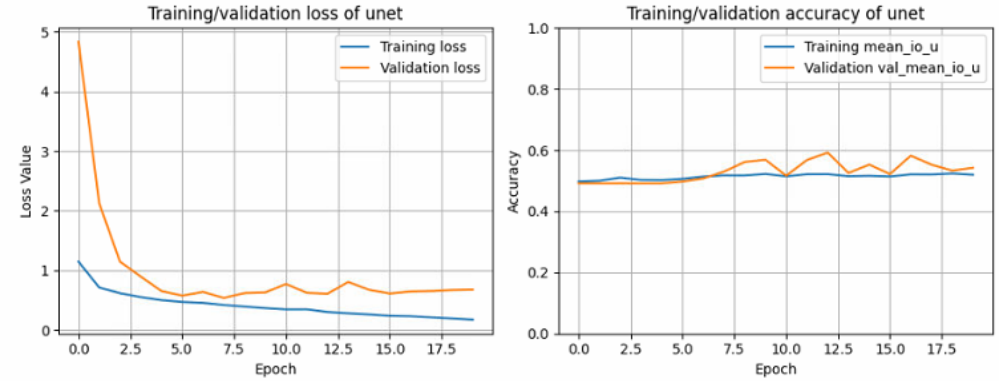
二、DenseNet121

DenseNet121(1):

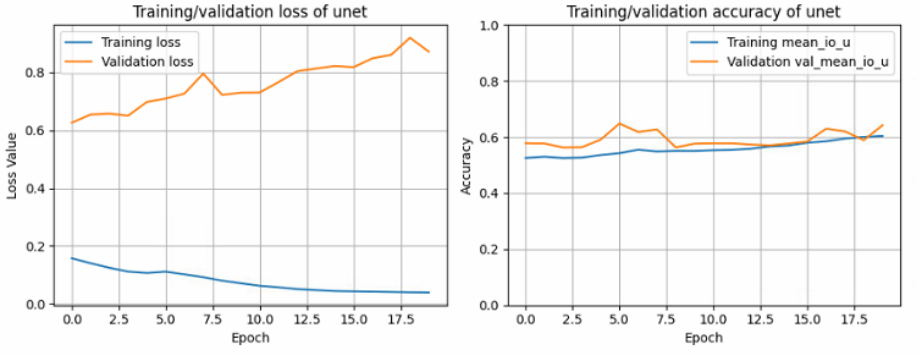
因為後來再針對使用 VGG 的架構中不管如何調整都無法有效增加準確率，所以嘗試著將預訓練模型換成 DenseNet121，而一開始若只把 VGG 改成 DenseNet 的話 val_mean_iou 會停在 0.4915 不會有變動，所以這邊有調整架構，並且將 trainable 設置為 true。

```
from tensorflow.keras.applications.densenet import DenseNet121
def def_unet(classes,height,width,channels,base_model,layer_names,name):
    base_model = tf.keras.Model(inputs=base_model.input,outputs = [base_model.output]+
                                                    [base_model.get_layer(na).output for na in reversed(layer_names)])
    #base_model.summary()
    inputs = tf.keras.layers.Input(shape=(height,width,channels))
    skips  = base_model(inputs)
    x      = skips[0]
    print(skips)

    # U型的底部處理
    #x      = tf.keras.layers.Conv2D(1024,(3,3),padding='same',activation='relu')(x)
    for layer in upsample(1024,(3,3)):
        x=layer(x)
    #upsampling network
    print(skips)
    for ch,skip in zip([512,256,64],skips[1:-1]):
        x = tf.keras.layers.Concatenate()([x,skip])
        for layer in upsample(ch,(3,3)):
            x=layer(x)

    x = tf.keras.layers.Concatenate()([x,skips[-1]])
    for layer in upsample(128,(3,3)):
        x=layer(x)
    x = tf.keras.layers.Conv2D(96,(1,1),padding='same')(x)
    #最後輸出
    x = tf.keras.layers.Conv2D(classes,(3,3),padding='same',activation='softmax')(x)

    unet = tf.keras.Model(inputs=inputs,outputs=x,name=name)
    unet.summary()
    base_model.trainable = True
    unet.compile(loss=tf.keras.losses.CategoricalCrossentropy(),optimizer='adam'
                 ,metrics=[tf.keras.metrics.MeanIoU(num_classes=59)])
    return unet

dense_layer_names = [
    'conv1/relu',
    'conv2_block6_concat',
    'conv3_block12_concat',
    'conv4_block24_concat',
]

model = def_unet(59,128,128,3,DenseNet121(include_top = False),dense_layer_names,'unet-dense')

```

從下圖可以發現 loss 跟使用 VGG 時相比低了許多，loss 最低有到 0.1712，而 val_loss 則是 0.62。所以接下來我又再訓練了 20epoch，想看看 loss 是否會一路下降。



沒想到卻發現再繼續訓練下去的時，val_loss 馬上升高，也就意味著過擬和的發生。

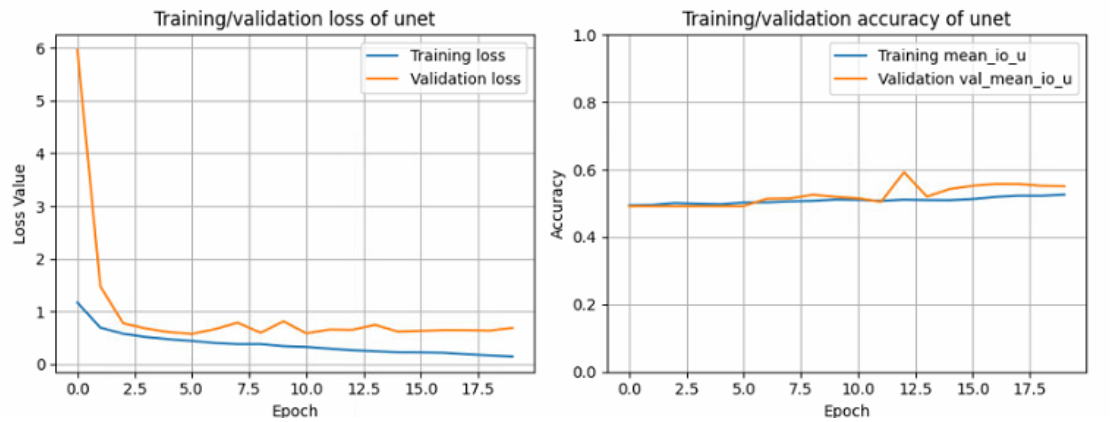


DenseNet121(2):

在這邊我想說先調整看看上採樣的通道數大小(通道數越大能越好的捕捉細節)，透過降低 skip 的通道大小的方式來看看能不能使得過擬合的情況好轉。

```python
from tensorflow.keras.applications.densenet import DenseNet121
def def_unet(classes,height,width,channels,base_model,layer_names,name):
    base_model = tf.keras.Model(inputs=base_model.input,outputs = [base_model.output]+
                                                    [base_model.get_layer(na).output for na in reversed(layer_names)])
    #base_model.summary()
    inputs = tf.keras.layers.Input(shape=(height,width,channels))
    skips  = base_model(inputs)
    x      = skips[0]

    # U型的底部處理
    #x      = tf.keras.layers.Conv2D(1024,(3,3),padding='same',activation='relu')(x)
    for layer in upsample(512,(3,3)):
        x=layer(x)
    #upsampling network
    print(skips)
    for ch,skip in zip([256,128,64],skips[1:-1]):
        x = tf.keras.layers.Concatenate()([x,skip])
        for layer in upsample(ch,(3,3)):
            x=layer(x)

    x = tf.keras.layers.Concatenate()([x,skips[-1]])
    for layer in upsample(128,(3,3)):
        x=layer(x)
    x = tf.keras.layers.Conv2D(96,(1,1),padding='same')(x)
    #最後輸出
    x = tf.keras.layers.Conv2D(classes,(3,3),padding='same',activation='softmax')(x)

    unet = tf.keras.Model(inputs=inputs,outputs=x,name=name)
    unet.summary()
    base_model.trainable = True
    unet.compile(loss=tf.keras.losses.CategoricalCrossentropy(),optimizer='adam'
                 ,metrics=[tf.keras.metrics.MeanIoU(num_classes=59)])
    return unet

dense_layer_names = [
    'conv1/relu',
    'conv2_block6_concat',
    'conv3_block12_concat',
    'conv4_block24_concat',
]

model = def_unet(59,128,128,3,DenseNet121(include_top = False),dense_layer_names,'unet-dense')
```

從下圖中我們可以發現他跟剛剛的結果其實是差不多的。



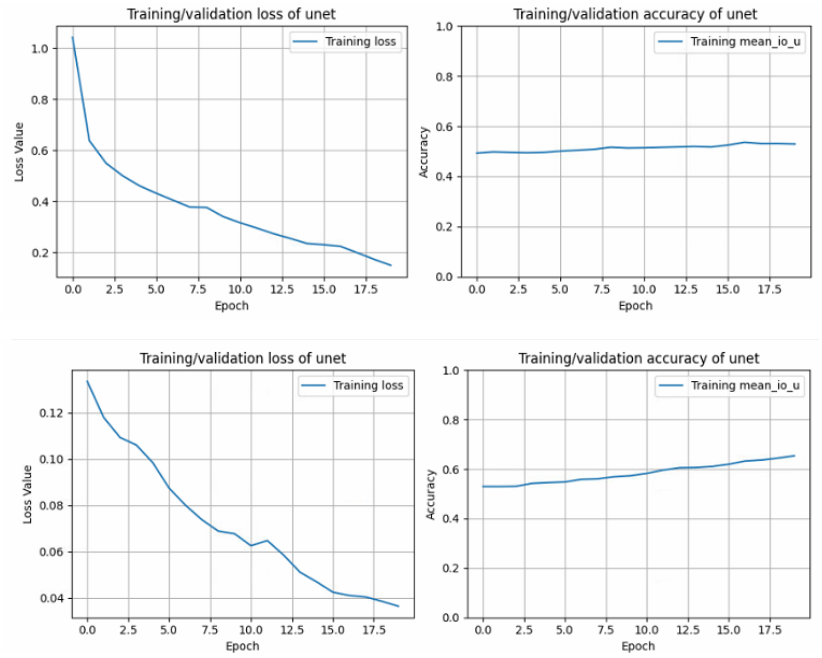
DenseNet121(3):

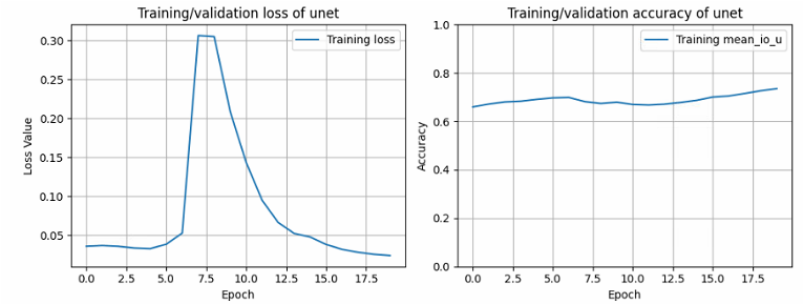
後來想到說，既然一開始老師就沒有設定驗證集，也說若需要驗證資料集再切割就好，所以接下來我試著只使用訓練資料(750)及測試資料(250)來訓練我的模型。

在這邊只有 callback 需要進行修改，將 monitor 的監測指標改成 loss。

```
checkpoint  =[tf.keras.callbacks.ModelCheckpoint("hw4_4_6_copy.h5",monitor='loss',save_best_only=True)]
model_history=model.fit(train_gen, epochs= 20,
                        # validation_data= val_gen,
                         callbacks=checkpoint)#[DisplayCallback()])
```

因為這邊的訴求變成 loss 越低越好，所以我跑了更多圈來期望能得到更低的 loss。由下圖可以看到 loss 是一路下降的，而在最後一張圖中 loss 最低來到了 0.0237。



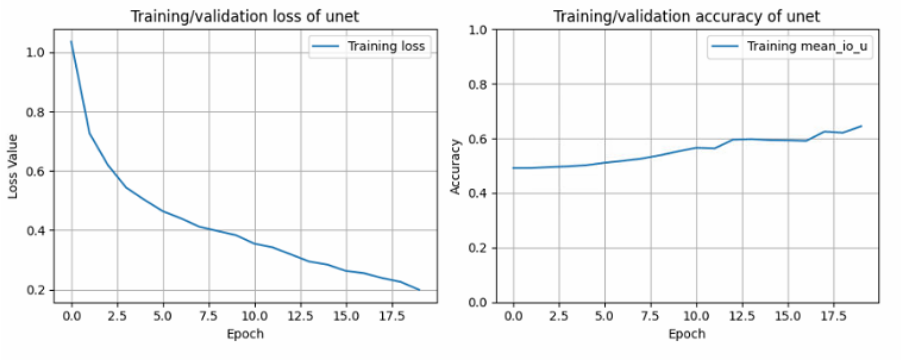


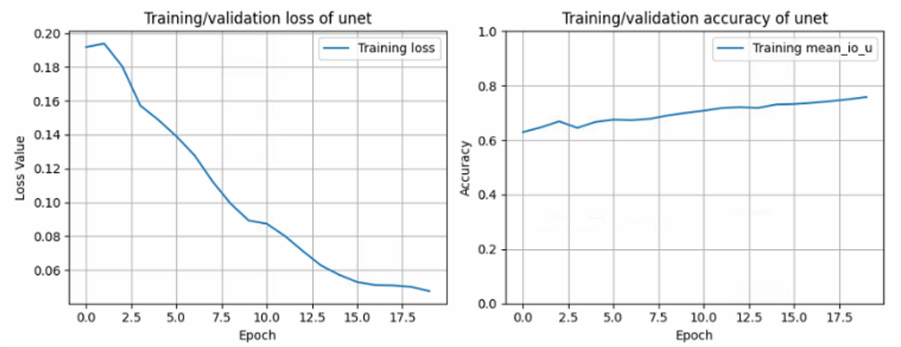
DenseNet121(4):

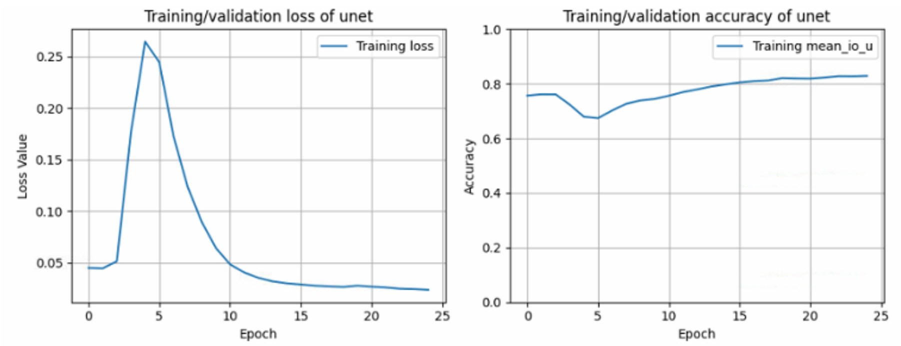
因為剛剛的 loss 沒有再下降了，所以這邊我想試試看如果我再試著增加通道數及模型的複雜程度看看能不能更好的抓取特徵。但通道數上升會使的計算量變大，所以也不能一味往上調整。

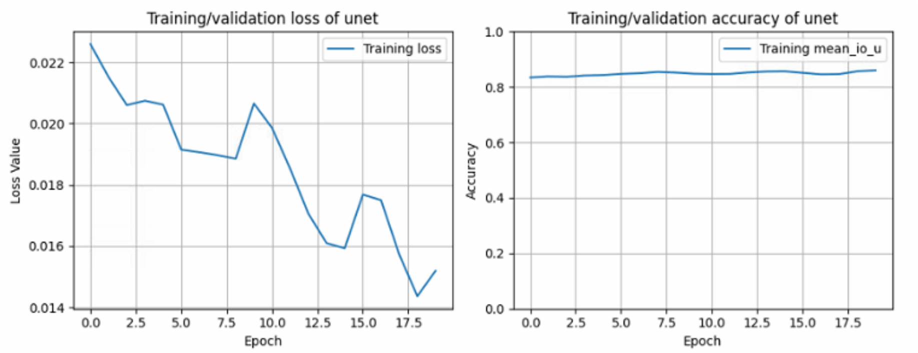
```
from tensorflow.keras.applications.densenet import DenseNet121
def def_unet(classes,height,width,channels,base_model,layer_names,name):
    base_model = tf.keras.Model(inputs=base_model.input,outputs = [base_model.output]+
                                                    [base_model.get_layer(na).output for na in reversed(layer_names)])
    #base_model.summary()
    inputs = tf.keras.layers.Input(shape=(height,width,channels))
    skips  = base_model(inputs)
    x      = skips[0]
    print(skips)

    # U型的底部處理
    x      = tf.keras.layers.Conv2D(1024,(3,3),padding='same',activation='relu')(x)
    for layer in upsample(1024,(3,3)):
        x=layer(x)
    #upsampling network
    #print(x)
    print(skips)
    for ch,skip in zip([512,256,128],skips[1:-1]):
        x = tf.keras.layers.Concatenate()([x,skip])
        for layer in upsample(ch,(3,3)):
            x=layer(x)
    print(x)
    x = tf.keras.layers.Concatenate()([x,skips[-1]])
    for layer in upsample(128,(3,3)):
        x=layer(x)
    x = tf.keras.layers.Conv2D(96,(3,3),padding='same',activation='relu')(x)
    x = tf.keras.layers.Conv2D(96,(1,1),padding='same')(x)#activation='relu')(x)
    #最後輸出
    x = tf.keras.layers.Conv2D(classes,(3,3),padding='same',activation='softmax')(x)

    unet = tf.keras.Model(inputs=inputs,outputs=x,name=name)
    unet.summary()
    base_model.trainable = True
    unet.compile(loss=tf.keras.losses.CategoricalCrossentropy(),optimizer='adam'
                 ,metrics=[tf.keras.metrics.MeanIoU(num_classes=59)])
    return unet

dense_layer_names = [
    'conv1/relu',
    'conv2_block6_concat',
    'conv3_block12_concat',
    'conv4_block24_concat',
]

model = def_unet(59,128,128,3,DenseNet121(include_top = False),dense_layer_names,'unet-dense')
```

在下圖中，loss 確實有下降，而在最低點時有到 0.0144。









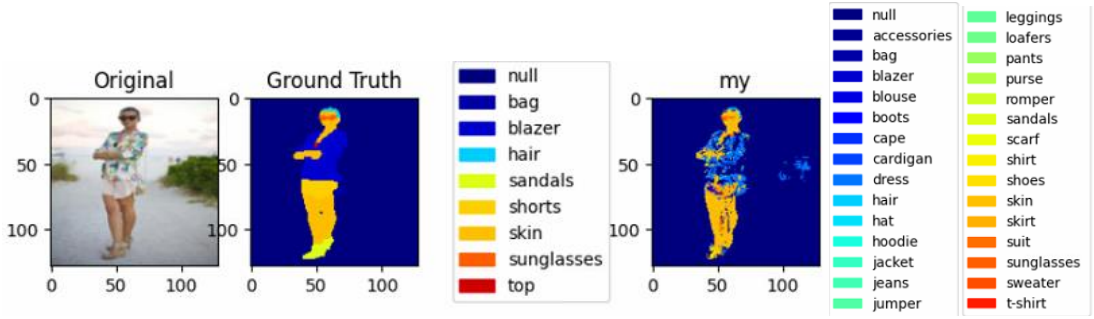
結果

average_iou 的部分我都是去找測試資料集中 iou 值最接近平均的圖片來顯示。

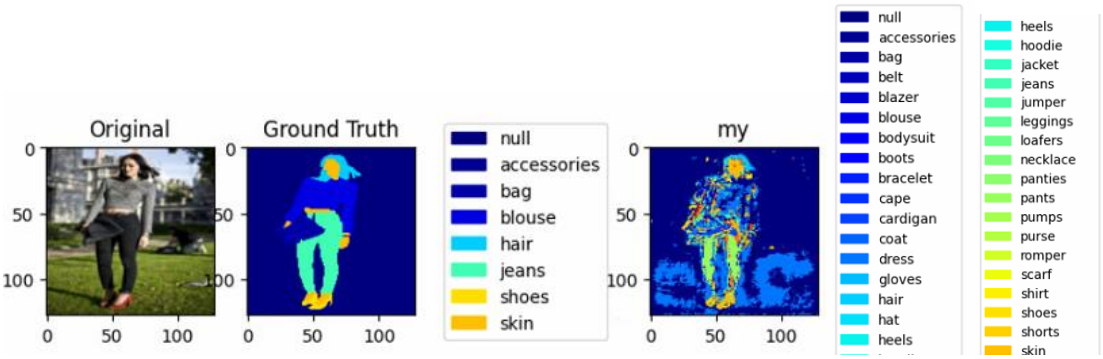
一、VGG

VGG(1):

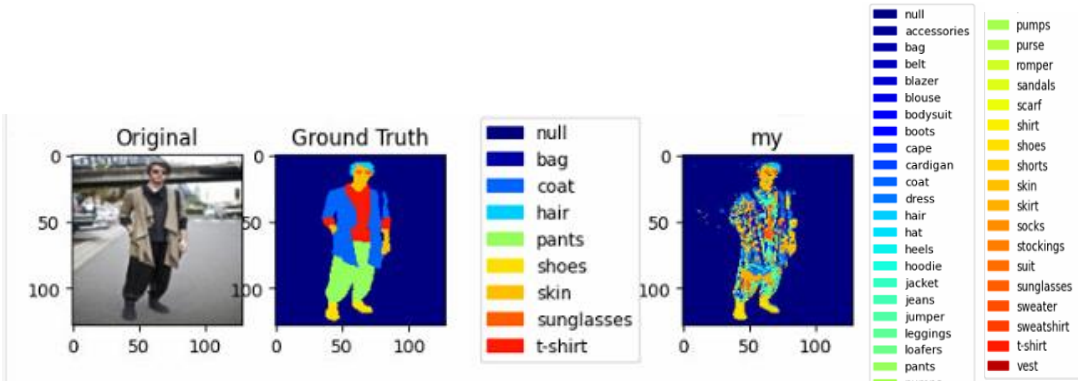
max_iou: 0.89373827



min_iou: 0.7272321

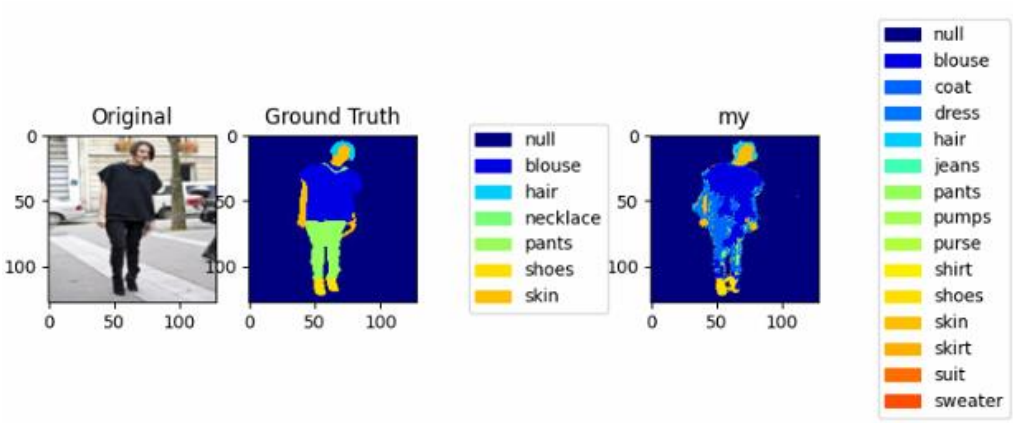


average_iou: 0.824109

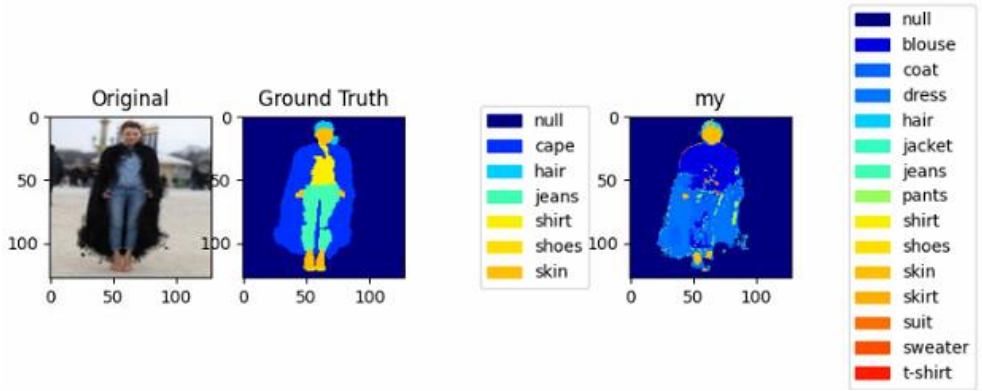


VGG(2):

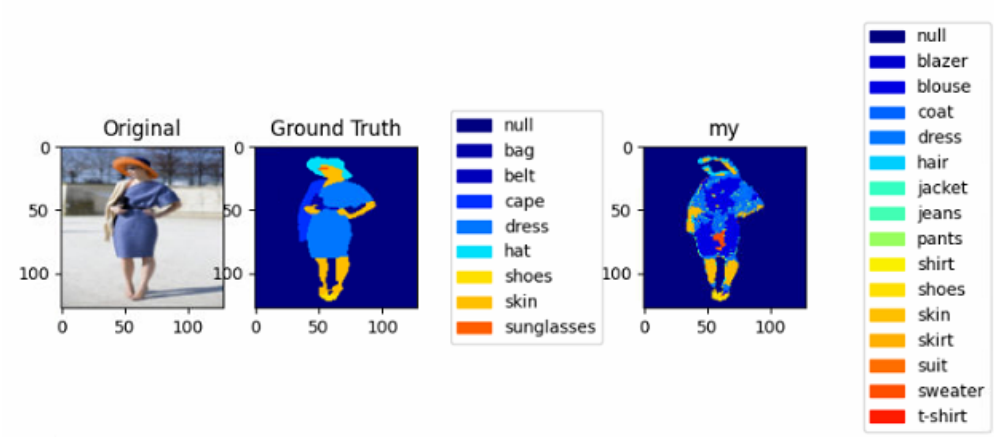
max_iou: 0.9015054



min_iou: 0.7618691



average_iou: 0.83337927

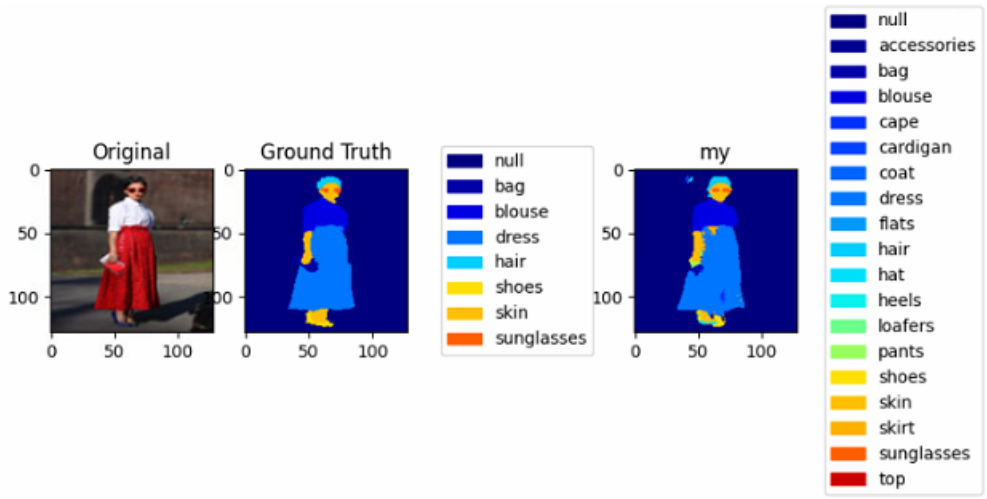


在 VGG(1)及 VGG(2)中我們可以發現平均及最大最值的 iou 都有升高，所以將 trainable 設置為 true 是有用的，但從圖片中也可以發現，雖然數值升高了，但出來的效果仍舊不是太好。

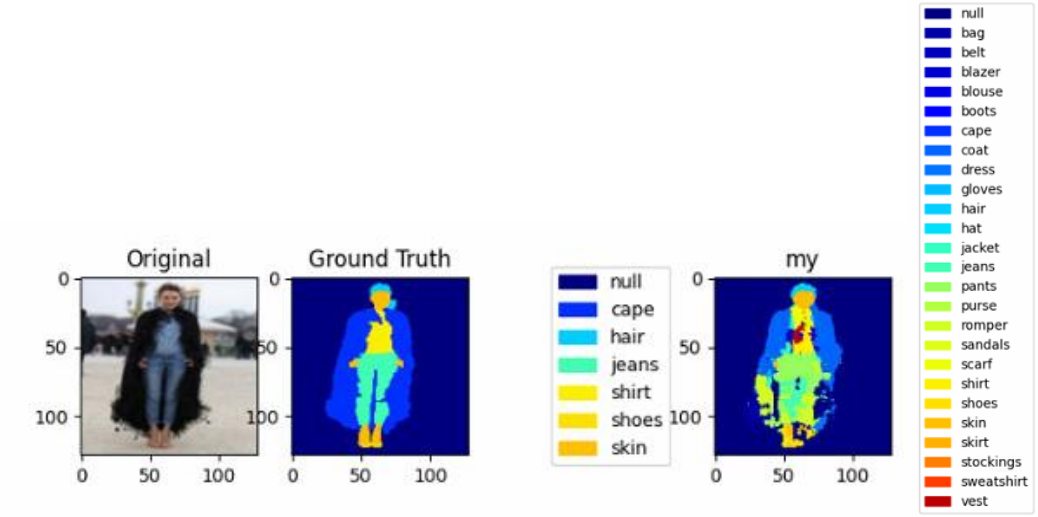
二、DenseNet121

DenseNet121(1):

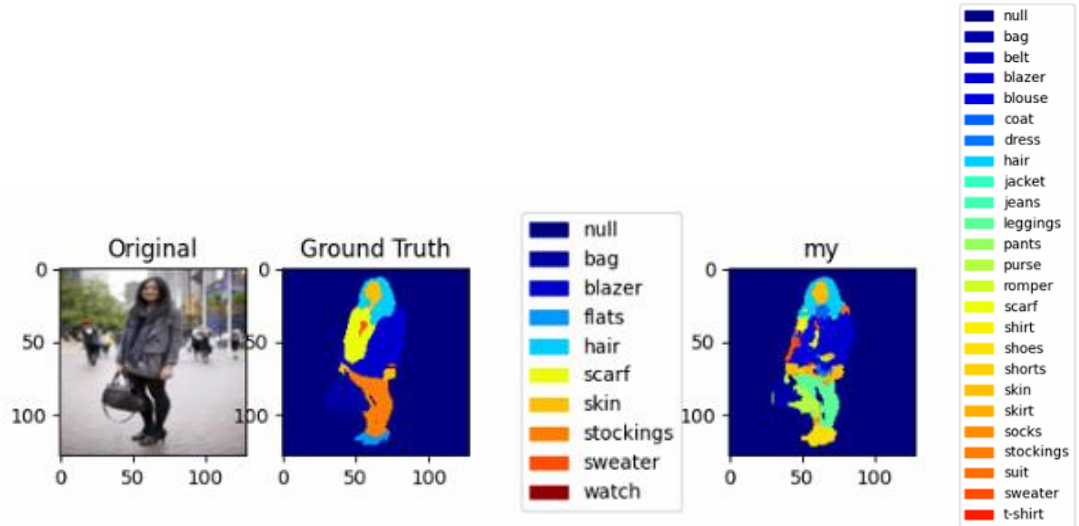
Max_iou: 0.9630014



min_iou: 0.78019714

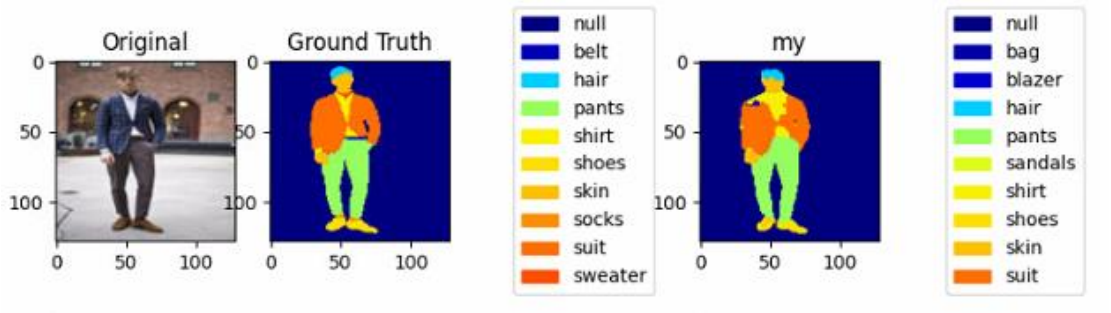


average_iou: 0.880

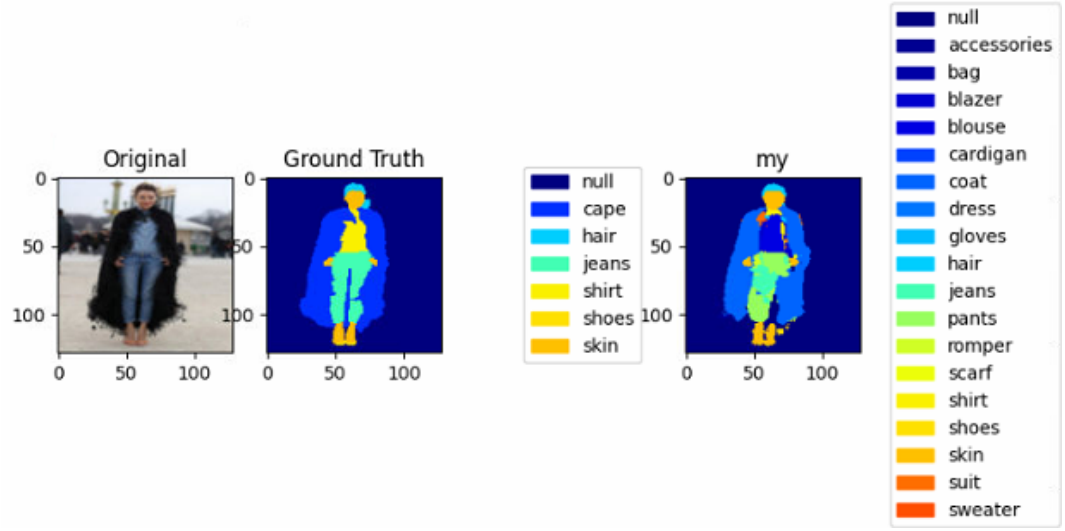


DenseNet121(2):

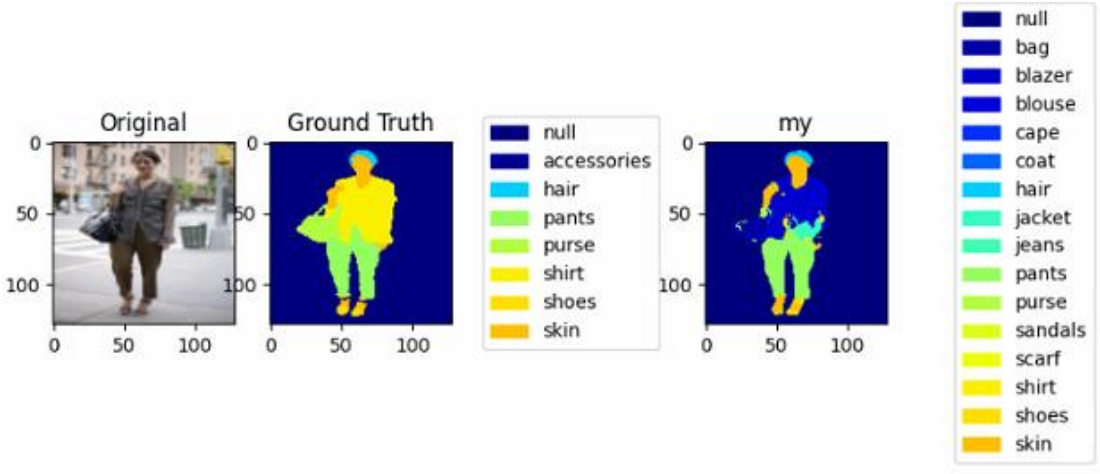
max_iou: 0.9668262



min_iou: 0.78701556

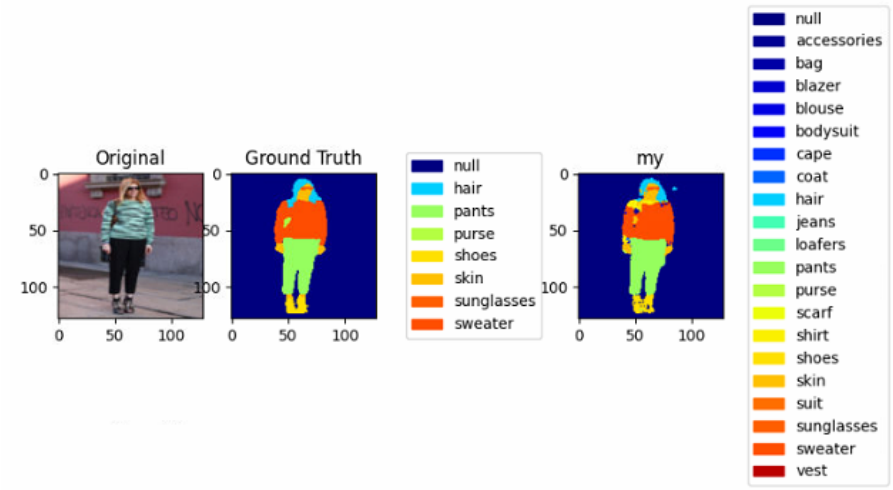


average_iou: 0.8755697

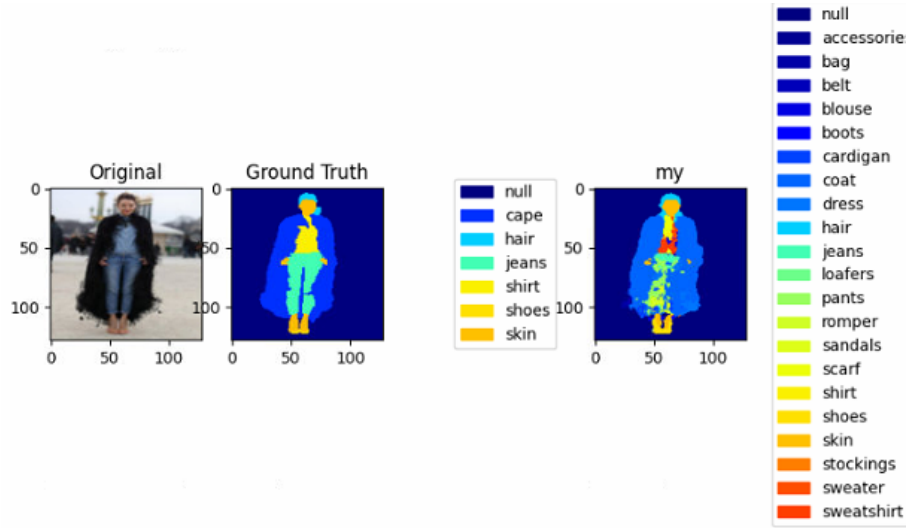


DenseNet121(3):

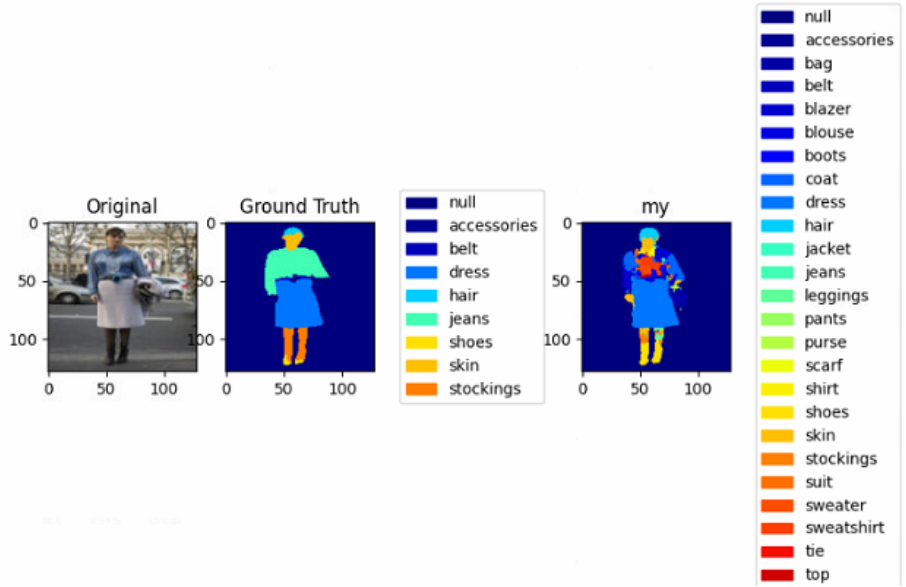
max_iou: 0.96491015



min_iou: 0.7741741

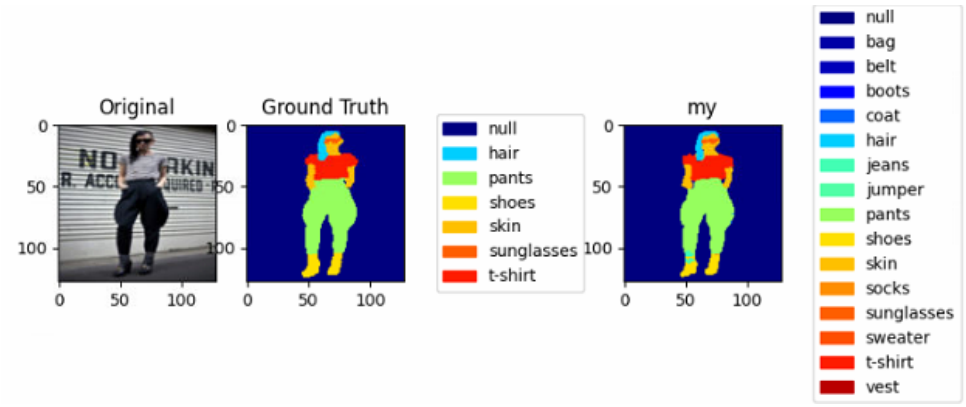


average_iou: 0.88187957

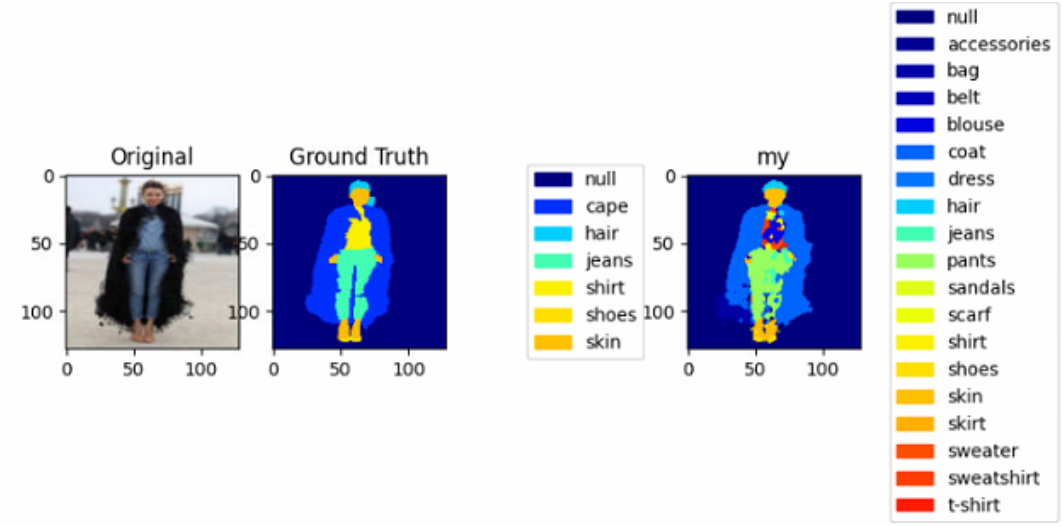


DenseNet121(4):

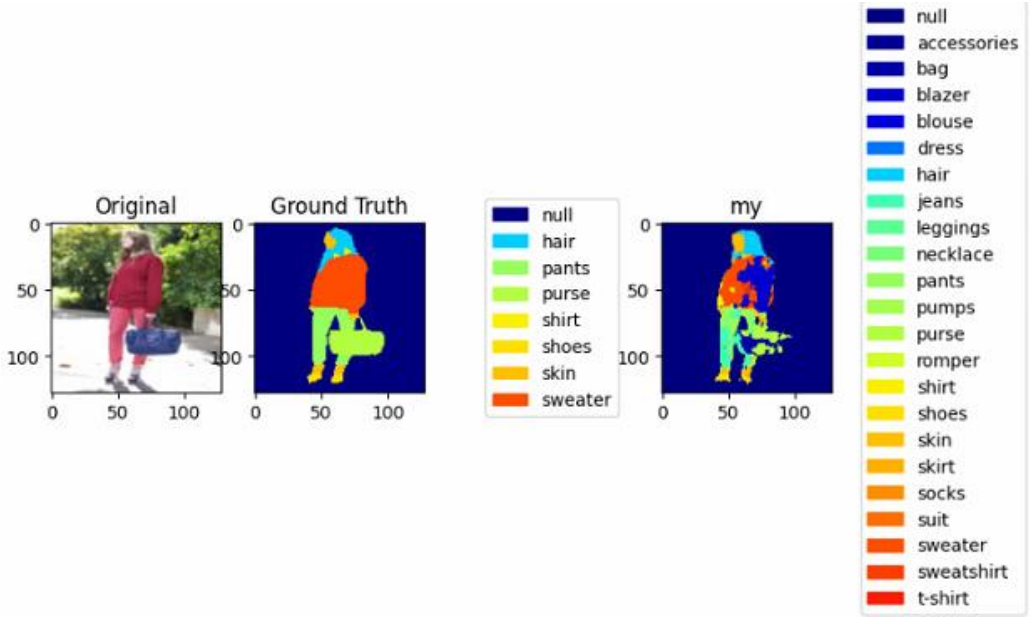
max_iou: 0.9832030



min_iou: 0.77288127



average_iou: 0.8855911



以 densenet 來看，我們可以發現當我們不使用驗證資料集時反而會得到更

好的 iou 數值，所以我認為以這次的作業而言，驗證資料的存在其實沒有那麼重要，反而是單純的使用訓練集就已足夠了。

結論

在這次的作業中，一開始就遇到了資料前處理的問題，因為之前從沒有自己實作過 generator，所以在實作時也研究了很久，最後也透過同學的教導成功寫出來，同時這也是我第一次遇到.MAT 檔的部分，一開始時忘記了 resize 的默認插值法是 bilinear，導致出來的訓練結果相當奇怪，後來才發現是自己忘記指定插值法。

而在後來的訓練過程中，VGG 的準確率一直上不去，出來的效果也不太好，在跟同學的討論及研究過程中發現 densenet 的效果不錯，也發現說如果我們沒有驗證集的效果會更好，而在有加驗證集的情況下都會很快發生過擬和，所以在後來的嘗試中我們也試過將驗證集拿掉，最後也得到了更好的測試結果。

而一開始在對測試資料集算 iou 的時候我使用的是面積的方式，後來才想到，因為在訓練時我使用的評估標準是 tf.keras.metrics.MeanIoU，所以在計算 iou 時我應該也要用 tf.keras.metrics.MeanIoU 才對。

而這次的作業基本上是由我、芷柔、恩妮、謹彤一起討論出來的結果，因為一開始其實大家都不太懂，所以也做了一些對於測試結果沒有很大改進的嘗試，或是因為小地方的錯誤而重跑了好幾次，但我也因此對於 unet 的架構更加了解，也更懂得該怎麼處理語意切割的標示圖片了。

參考文獻

https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence
https://www.tensorflow.org/api_docs/python/tf/keras/metrics/MeanIoU