Homework #2

Deep Learning for Computer Vision

Problem 1: Image Classification (10%)

1. (2%)

```
model(
  (backbone): VGG(
    (features): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU(inplace=True)
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (9): ReLU(inplace=True)
      (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (12): ReLU(inplace=True)
      (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (16): ReLU(inplace=True)
      (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (19): ReLU(inplace=True)
      (20): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

此題主要的修改是將 VGG16_BN 中 classifier 的最後一層 Linear 的 out_features 從 1000 個 class 調整為 50。

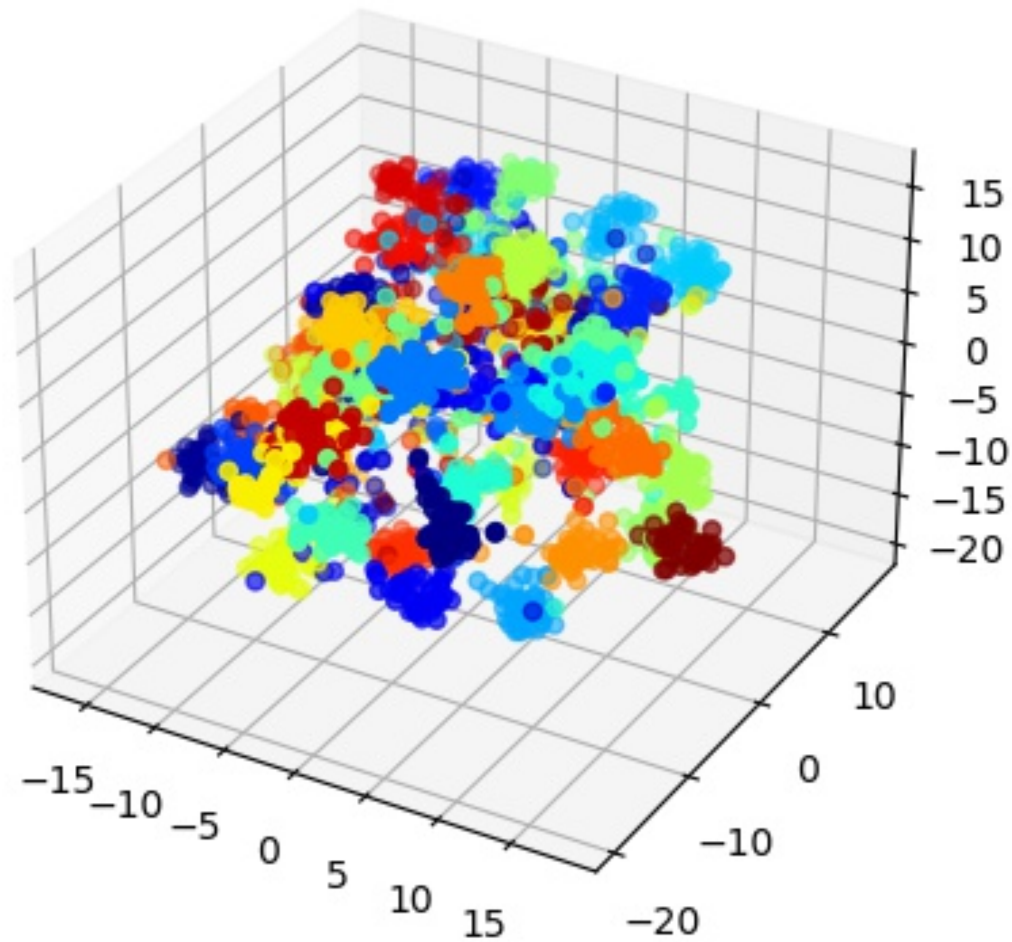
```
      (22): ReLU(inplace=True)
      (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (24): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (25): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (26): ReLU(inplace=True)
      (27): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (28): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (29): ReLU(inplace=True)
      (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (31): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (32): ReLU(inplace=True)
      (33): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (35): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (36): ReLU(inplace=True)
      (37): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (38): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (39): ReLU(inplace=True)
      (40): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (41): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (42): ReLU(inplace=True)
      (43): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
    (classifier): Sequential(
      (0): Linear(in_features=25088, out_features=4096, bias=True)
      (1): ReLU(inplace=True)
      (2): Dropout(p=0.5, inplace=False)
      (3): Linear(in_features=4096, out_features=4096, bias=True)
      (4): ReLU(inplace=True)
      (5): Dropout(p=0.5, inplace=False)
      (6): Linear(in_features=4096, out_features=50, bias=True)
    )
  )
)
```

此題主要的修改是將 VGG16_BN 中 classifier 的最後一層 Linear 的 out_features 從 1000 個 class 調整為 50。

2. (2%)

Accuracy: 82.04%

3. (6%)



此處將 t-SNE 的結果投影到三維的空間上，方便觀察。

可以看到 features 被分成一團一團的，以鮮明的顏色做為類別的區隔，不過還是有少數顏色不同的零星部分四散，可能就是沒有被 model 良好區分的部分。

Problem 2: Semantic Segmentation (30%)

1. (5%)

```
model(
  (backbone): VGG(
    (features): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU(inplace=True)
      (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): ReLU(inplace=True)
      (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (6): ReLU(inplace=True)
      (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (8): ReLU(inplace=True)
      (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (11): ReLU(inplace=True)
      (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (13): ReLU(inplace=True)
      (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (15): ReLU(inplace=True)
      (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (18): ReLU(inplace=True)
      (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (20): ReLU(inplace=True)
      (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (22): ReLU(inplace=True)
      (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (25): ReLU(inplace=True)
      (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (27): ReLU(inplace=True)
      (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (29): ReLU(inplace=True)
      (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    )
    (avgpool): Identity()
```

```
      (classifier): Identity()
    )
    (Upsample16x16): Sequential(
      (0): Conv_Block(
        (block): Sequential(
          (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU(inplace=True)
        )
      )
      (1): Conv_Block(
        (block): Sequential(
          (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU(inplace=True)
        )
      )
      (2): Conv_Block(
        (block): Sequential(
          (0): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU(inplace=True)
        )
      )
      (3): Trans_Block(
        (block): Sequential(
          (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1,
1), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): LeakyReLU(negative_slope=1, inplace=True)
        )
      )
      (4): Trans_Block(
        (block): Sequential(
          (0): ConvTranspose2d(128, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1,
1), bias=False)
```

```
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): LeakyReLU(negative_slope=1, inplace=True)
      )
    )
    (5): Trans_Block(
      (block): Sequential(
        (0): ConvTranspose2d(128, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1,
1), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): LeakyReLU(negative_slope=1, inplace=True)
      )
    )
    (6): Trans_Block(
      (block): Sequential(
        (0): ConvTranspose2d(128, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1,
1), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): LeakyReLU(negative_slope=1, inplace=True)
      )
    )
    (7): Trans_Block(
      (block): Sequential(
        (0): ConvTranspose2d(128, 7, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(7, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): LeakyReLU(negative_slope=1, inplace=True)
      )
    )
  )
)
```

實現 VGG16_FCN32s：

此題主要利用了 Conv_Block(Conv2d, BatchNorm2d, ReLU), Trans_Block(ConvTranspose2d, BatchNorm2d, LeakyReLU)。

先將原 VGG16 的 avgpool, classifier 替換成 Identity（output 即 input）的結構。

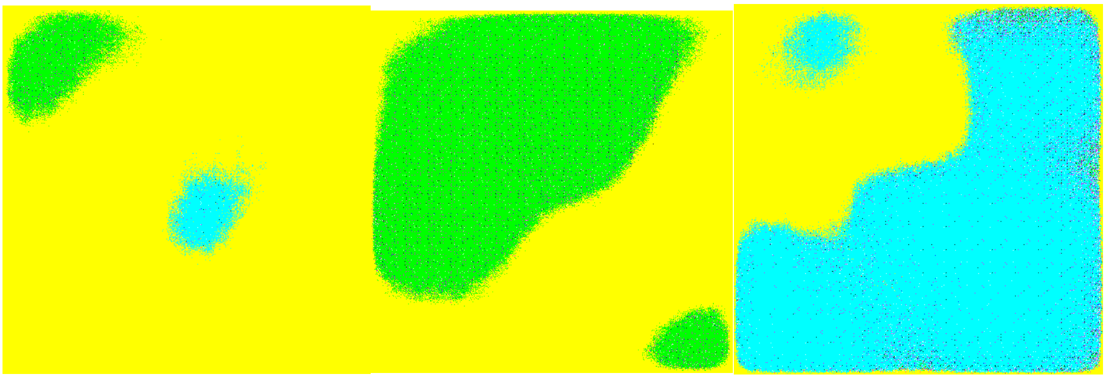
再將 classifier 的 output 經過三個 Conv_Block 和五個 Trans_Block，其中每次的 ConvTranspose2d 會將影像的邊長放大 2 倍，一共 32 倍。

為助理解，架構流程如下圖所示：

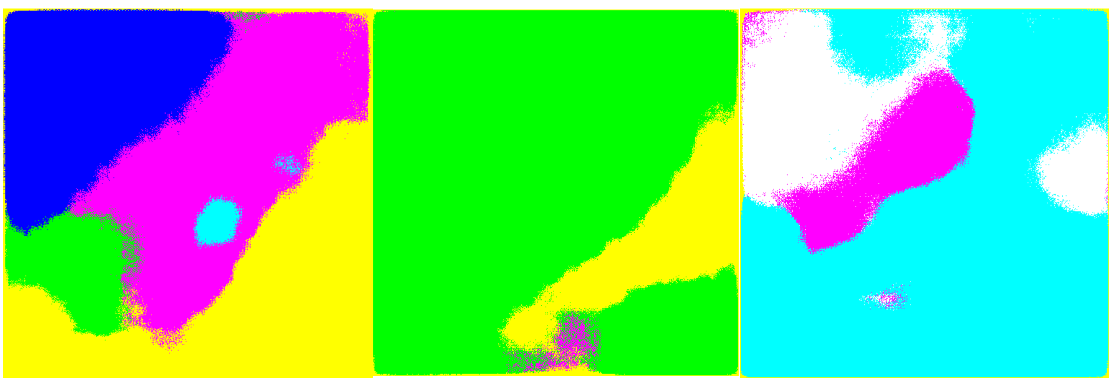
```
# FCN 32s
self.backbone = vgg16(pretrained = True)
self.backbone.avgpool = Identity()
self.backbone.classifier = Identity()
self.Upsample16x16 = nn.Sequential(
                                    Conv_Block(512, 512, 3, 1, 1),
                                    Conv_Block(512, 512, 3, 1, 1),
                                    Conv_Block(512, 256, 3, 1, 1),
                                    Trans_Block(256, 128, 4, 2, 1),
                                    Trans_Block(128, 128, 4, 2, 1),
                                    Trans_Block(128, 128, 4, 2, 1),
                                    Trans_Block(128, 128, 4, 2, 1),
                                    Trans_Block(128, 7, 4, 2, 1),
                                )
features, _ = self.backbone(input)
features = features.reshape(-1, 512, 16, 16)
image = self.Upsample16x16(features)
```

2. (5%)

Early stage:



Middle stage:



Final stage:

3. (5%) Improved network architecture:

```
model(
  (backbone): VGG(
    (features): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU(inplace=True)
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (9): ReLU(inplace=True)
      (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (12): ReLU(inplace=True)
      (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (16): ReLU(inplace=True)
      (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (19): ReLU(inplace=True)
      (20): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (22): ReLU(inplace=True)
      (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (24): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (25): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (26): ReLU(inplace=True)
      (27): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (28): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
        (29): ReLU(inplace=True)
        (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (31): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (32): ReLU(inplace=True)
        (33): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
        (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (35): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (36): ReLU(inplace=True)
        (37): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (38): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (39): ReLU(inplace=True)
        (40): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (41): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (42): ReLU(inplace=True)
        (43): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      )
      (avgpool): Identity()
      (classifier): Identity()
    )
    (Upsample16x16): Upsample(scale_factor=2.0, mode=nearest)
    (Upsample32x32): Sequential(
      (0): Conv_Block(
        (block): Sequential(
          (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU(inplace=True)
        )
      )
      (1): Upsample(scale_factor=2.0, mode=nearest)
    )
    (magnitude): Conv_Block(
      (block): Sequential(
        (0): Conv2d(256, 512, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
      )
    )
```

```
  (Upsample64x64): Sequential(
    (0): Conv_Block(
      (block): Sequential(
        (0): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
      )
    )
    (1): Conv_Block(
      (block): Sequential(
        (0): Conv2d(256, 128, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2),
bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
      )
    )
    (2): Conv_Block(
      (block): Sequential(
        (0): Conv2d(128, 7, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), bias=False)
        (1): BatchNorm2d(7, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
      )
    )
    (3): Upsample(scale_factor=8.0, mode=nearest)
  )
)
```

實現 VGG16_BN_FCN8s：

此處採用的方法不完全同於原 paper，但使用相同的概念，主要利用了 Conv_Block(Conv2d, BatchNorm2d, ReLU), Upsample 以取代 ConvTranspose2d。

先將原 VGG16_BN 的 avgpool, classifier 替換成 Identity（output 即 input）的結構。

視 VGG16_BN 中 features 的後三個 MaxPool2d 的 output 為三種 scales = (64, 64), (32, 32), (16, 16) 的 feature。

首先將 size = (16, 16) 的 feature 經過 Upsample 成 (32, 32)，再和原 size = (32, 32) 的 feature 作 elementwise 相加，得到融合後的 size = (32, 32) 的 feature。

之後再將此融合後的 feature 經過 Conv_Block, Upsample 成 (64, 64) 的 feature，和經過 Conv_Block 的原 size = (64, 64) 的 feature 作 elementwise 相加，得到融合後的 size = (64, 64) 的 feature。

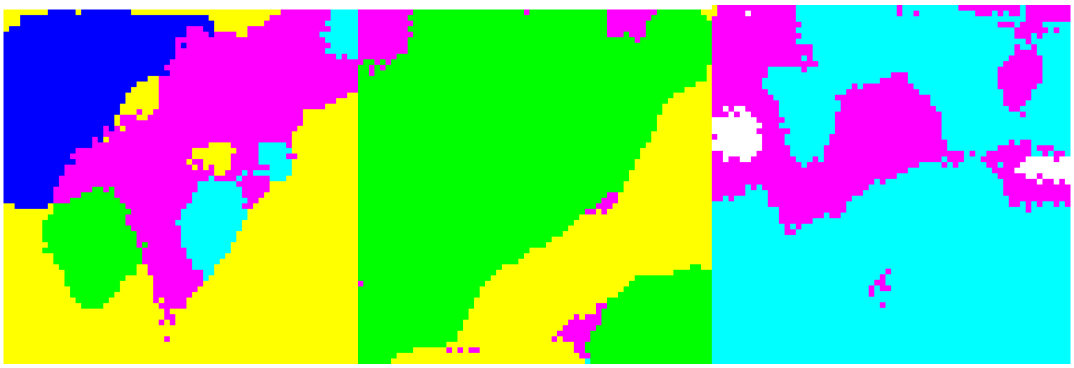
最後將此融合後的 feature 經過三次的 Conv_Block，最後 Upsample 成 size = (512, 512) 的大小。

為助理解，架構流程如下圖所示：

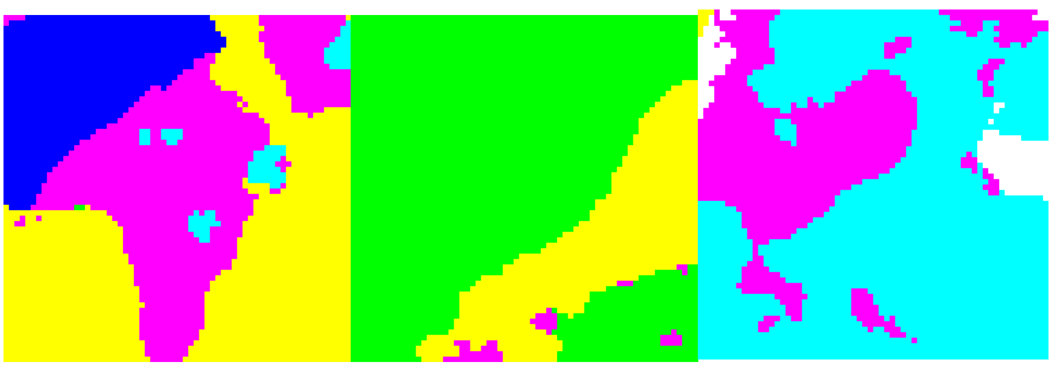
```
# FCN 8s
self.Upsample16x16 = nn.Upsample(scale_factor = 2)
self.Upsample32x32 = nn.Sequential(
                                    Conv_Block(512, 512, 3, 1, 1),
                                    nn.Upsample(scale_factor = 2)
                                )
self.magnitude = Conv_Block(256, 512, 5, 1, 2)
self.Upsample64x64 = nn.Sequential(
                                    Conv_Block(512, 256, 3, 1, 1),
                                    Conv_Block(256, 128, 5, 1, 2),
                                    Conv_Block(128, 7, 7, 1, 3),
                                    nn.Upsample(scale_factor = 8)
                                )
_, features_64_32_16 = self.backbone(input)
temp32x32 = self.Upsample16x16(features_64_32_16[2]) + features_64_32_16[1]
temp64x64 = self.Upsample32x32(temp32x32) + self.magnitude(features_64_32_16[0])
image = self.Upsample64x64(temp64x64)
```

4. (5%)

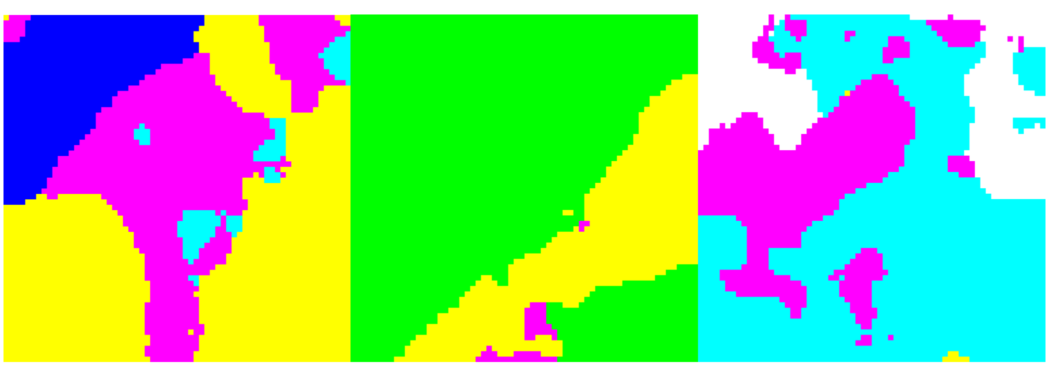
Early stage:



Middle stage:



Final stage:



5. (10%)

Baseline:
class #0 : 0.72787
class #1 : 0.86715
class #2 : 0.22550
class #3 : 0.78474
class #4 : 0.72001
class #5 : 0.65088
mean_iou: 0.662692

Improve:
class #0 : 0.74051
class #1 : 0.87762
class #2 : 0.32882
class #3 : 0.79041
class #4 : 0.72958
class #5 : 0.64965
mean_iou: 0.686099

從 model 選用的 feature 上進行分析：

Baseline model 僅有將 scale = (16, 16) 的 feature 來進行 Convolution 運算，可以理解為這樣的 feature 主要是原影像上 scale 較大的資訊（因為 receptive field 較大），所以在某些 scale 較小的物體、區塊上，baseline model 沒有辦法預測的非常準確。

而 Improve model 則因為有將 scale = (64, 64), (32, 32), (16, 16) 的三種大小的 feature 納入運算並進行疊加融合，所以除了大面積的物體、區塊可以被預測到，較小區塊的資訊也有被保留住。

從預測結果圖上來看，可以看到 Improve model 的確有比 Baseline model 較細節上的呈現。

從 model 的 Upsample 方法上進行比較：

Baseline model 是採用 TransposeConv2d 來進行上採樣，雖然會有比較多的參數可以進行調整、學習，但這種方法當中 padding 的部分可能會讓產生的圖片四周的邊緣處和原圖有較大的落差，這點不管是 early, middle, final stage 中，都可以從預測影像中觀察得到。

Improve model 則是採用 Convolution + Upsample 的方法，以取代上述的作法，可以看到在影像四周的邊緣處這種方法的結果比較自然，而且 model 對於自己的切割預測也比較確定，結果會是一塊一塊的，不會像 Baseline model 產生的圖會有粉狀的預測產生。

從 Training 上進行比較：

兩個 model 的參數量（21941582, 22404814）、大小（85739KB, 87586KB）差不多，使用相同的 batch_size = 8, optimizer = SGD, learning rate = 0.001。

Baseline model mIoU 提升的速度較慢，一開始的 mIoU 只有 0.3。而 Improve model 則較快（較少的 iteration 次數就可以達到 Baseline model 的 mIoU 分數），第1個 epoch 的 mIoU 就有 0.5 以上，應該是 Upsample 方法選用上的區別所致，因為有將各個 scale 大小的 feature 做結合、融合，所以在很前面的 epoch 就可以保有大略和細節的預測。

URL:

https://arxiv.org/pdf/1411.4038v2.pdf

https://pytorch.org/docs/stable/_modules/torchvision/models/vgg.html#vgg16

https://pytorch.org/docs/stable/_modules/torchvision/models/vgg.html#vgg16_bn

https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html