DLCV 109-1
HW2
陳芃彣 r08942085

Problem 1
1-1
Backbone: pertained vgg16
Layer: 將兩次 upsample 的結果 concat，最後過三層 fully connected layers。

```
Vggfcn(
  (pretrain): 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): 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=1000, bias=True)
    )
  )
  (layer1): 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)
  )
  (conv1): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
  (up): Upsample(scale_factor=2.0, mode=bilinear)
  (conv2): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc): Sequential(
    (0): Linear(in_features=6144, out_features=2048, bias=True)
    (1): Linear(in_features=2048, out_features=256, bias=True)
    (2): Linear(in_features=256, out_features=50, bias=True)
  )
)
```

Backbone

```python
class Vggfcn(nn.Module):
    def __init__(self, pretrain=False):
        super(Vggfcn, self).__init__()
        self.pretrain = pretrain
        self.layer1 = pretrain.features
        self.conv1 = nn.Conv2d(512, 512, 1)
        self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        self.conv2 = nn.Conv2d(512, 256, 3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(256, 128, 3, stride=1, padding=1)
        self.fc = nn.Sequential(
            nn.Linear(6144, 2048),
            nn.Linear(2048, 256),
            nn.Linear(256, 50),
            )

    def forward(self, input):
        x = input
        x = self.layer1(x) #[32, 512, 1, 1]
        x = self.conv1(x)
        x = self.up(x) # 512, 2, 2
        x = self.conv2(x)
        x = self.up(x) # 256, 4, 4
        x2 = self.conv3(x)
        x3 = torch.cat([x, x2], dim=1)
        x3 = x3.view(x3.shape[0], -1)
        output = self.fc(x3)
        return output
```

Backbone

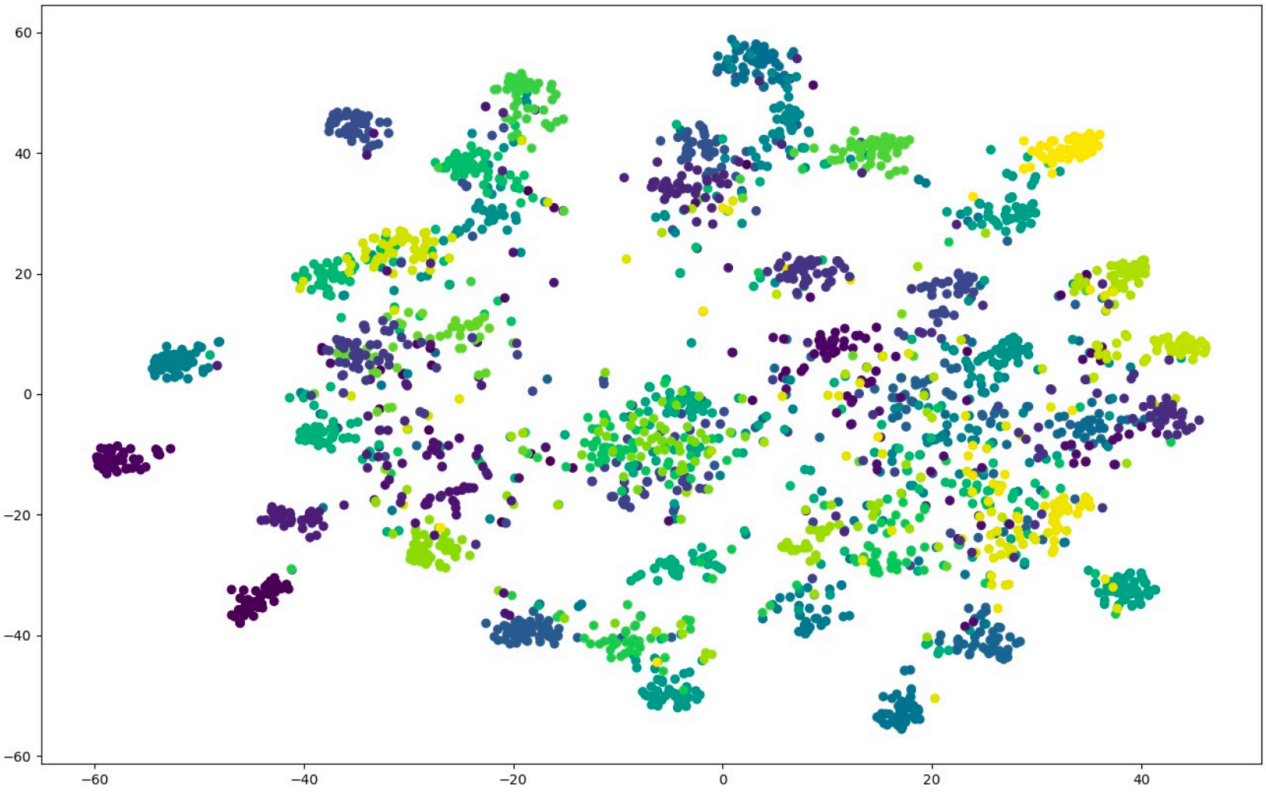
1-2

Accuracy on validation set:

```
Accuracy: 1821/2500 (73%)
```

1-3

Tsne result: 可以看到在 accuracy 73% 的分類結果下，資料點在高維分佈降維後的分群結果，有些有明顯的分群（例如周圍的部份），大致有 30 群，但如中間或中右的資料點，模型就沒有分群分得很好。



Problem2

2-1

Backbone: vgg16 (下圖中的 layer1)

Layer: 使用 convTranspose 五次，每次圖片放大兩倍，並且有經過 batch normalize 及 relu。

reference: https://github.com/pochih/FCN-pytorch/blob/master/python/fcn.py

```
Vggfcn32(
  (pretrain): 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): 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=1000, bias=True)
    )
  )
  (layer1): 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)
  )
  (relu): ReLU(inplace=True)
  (dev1): ConvTranspose2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
  (dev2): ConvTranspose2d(512, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
  (dev3): ConvTranspose2d(256, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
  (dev4): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
  (dev5): ConvTranspose2d(64, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
  (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn5): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv7): Conv2d(32, 7, kernel_size=(1, 1), stride=(1, 1))
)
```

Backbone

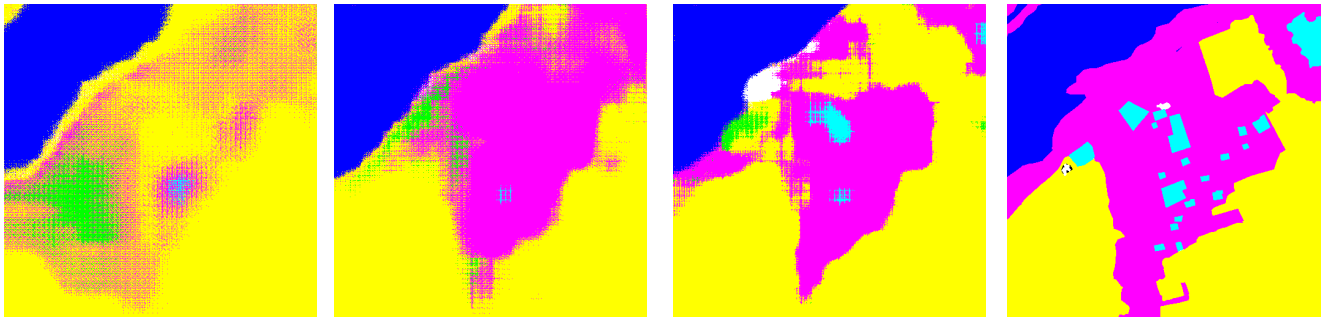
Backbone

```python
class Vggfcn32(nn.Module):
    def __init__(self, pretrain=False):
        super(Vggfcn32, self).__init__()
        self.pretrain = pretrain
        self.layer1 = pretrain.features
        self.relu = nn.ReLU(inplace=True)
        self.dev1 = nn.ConvTranspose2d(512, 512, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1)
        self.dev2 = nn.ConvTranspose2d(512, 256, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1)
        self.dev3 = nn.ConvTranspose2d(256, 128, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1)
        self.dev4 = nn.ConvTranspose2d(128, 64, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1)
        self.dev5 = nn.ConvTranspose2d(64, 32, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1)

        self.bn1 = nn.BatchNorm2d(512)
        self.bn2 = nn.BatchNorm2d(256)
        self.bn3 = nn.BatchNorm2d(128)
        self.bn4 = nn.BatchNorm2d(64)
        self.bn5 = nn.BatchNorm2d(32)
        self.conv7 = nn.Conv2d(32, 7, 1)

    def forward(self, input):
        x = input
        x = self.layer1(x)
        x = self.bn1(self.relu(self.dev1(x)))
        x = self.bn2(self.relu(self.dev2(x)))
        x = self.bn3(self.relu(self.dev3(x)))
        x = self.bn4(self.relu(self.dev4(x)))
        x = self.bn5(self.relu(self.dev5(x)))
        x = self.conv7(x)
        # pdb.set_trace()
        # x = self.convTranspose(x)
        # x = F.upsample_bilinear(x, input.size()[2:])
        return x
```

2-2

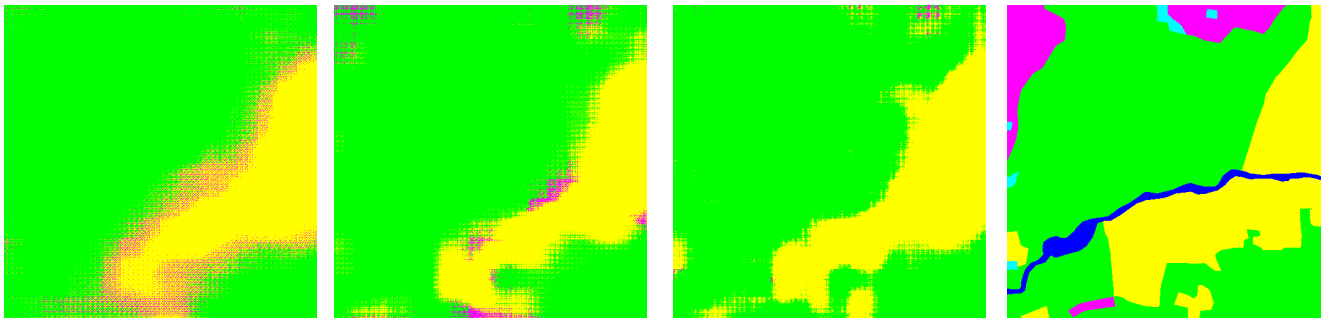
[0010_sat.jpg]

10th epoch 30th epoch 50th epoch gt



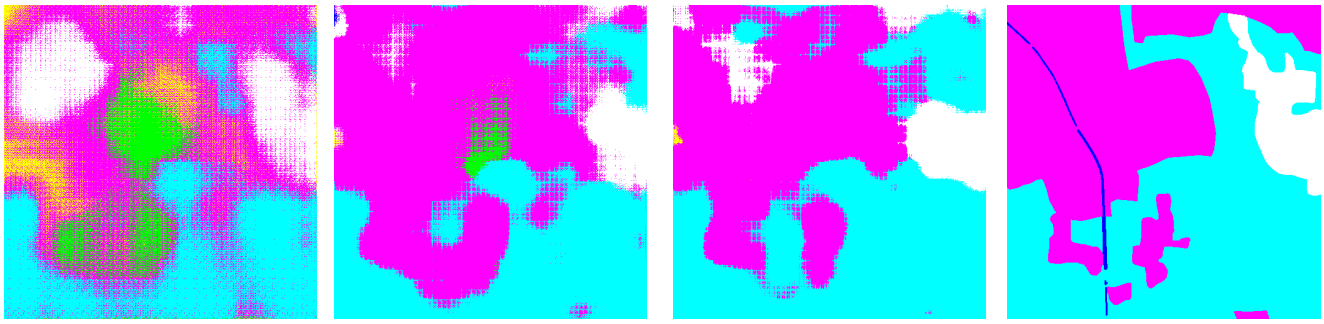
[0097_sat.jpg]

10th epoch 30th epoch 50th epoch gt



[107_sat.jpg]

10th epoch 30th epoch 50th epoch gt



2-3

Backbone: vgg16_bn
Layer: 模仿 fcn8s 的做法，把 vgg16_bn 前面 encode 的 features 與後來 upsample 的 features pixel wise 加起來，必經過幾次 convolution。本次 convTranspose 有先 initialize 成 bilinear upsample。

Reference: https://github.com/wkentaro/pytorch-fcn/blob/master/torchfcn/models/fcn32s.py

```
(block1): 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)
)
(block2): Sequential(
  (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)
)
(block3): Sequential(
  (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)
)
(block4): Sequential(
  (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)
)
(block5): Sequential(
  (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)
)
(relu): ReLU(inplace=True)
(dev1): ConvTranspose2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(dev1_1): ConvTranspose2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(dev2): ConvTranspose2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(dev3): ConvTranspose2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(dev4): ConvTranspose2d(64, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(bn1_1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(bn3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(bn4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(bn5): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(conv7): Conv2d(32, 7, kernel_size=(1, 1), stride=(1, 1))
(conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv2): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv3): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv4): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv5): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv6): Conv2d(32, 7, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
)
```

Backbone

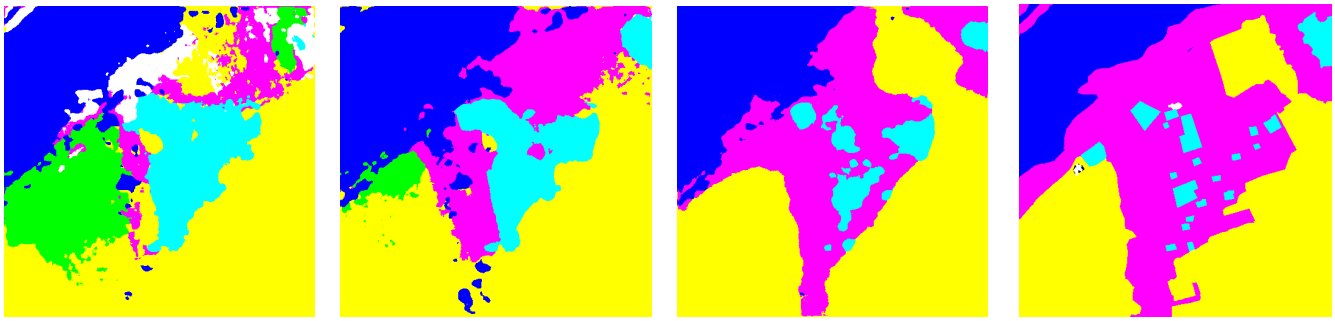
```python
class Vggfcn8(nn.Module):
    def __init__(self, pretrain=False):
        super(Vggfcn8, self).__init__()
        print("It's Vggfcn8_bn")
        self.pretrain = pretrain
        self.pretrain_all = pretrain.features

        self.block1 = pretrain.features[0:7] #[0:5]
        self.block2 = pretrain.features[7:14] #[5:10]
        self.block3 = pretrain.features[14:24] #[10:17]
        self.block4 = pretrain.features[24:34] #[17:24]
        self.block5 = pretrain.features[34:44] #[24:31]

        self.relu = nn.ReLU(inplace=True)
        self.dev1 = nn.ConvTranspose2d(512, 512, kernel_size=3,
                    stride=2, padding=1, dilation=1, output_padding=1)
        self.dev1_1 = nn.ConvTranspose2d(512, 512, kernel_size=3,
                    stride=2, padding=1, dilation=1, output_padding=1)
        self.dev2 = nn.ConvTranspose2d(256, 256, kernel_size=3,
                    stride=2, padding=1, dilation=1, output_padding=1)
        self.dev3 = nn.ConvTranspose2d(128, 128, kernel_size=3,
                    stride=2, padding=1, dilation=1, output_padding=1)
        self.dev4 = nn.ConvTranspose2d(64, 64, kernel_size=3,
                    stride=2, padding=1, dilation=1, output_padding=1)

        self.bn1 = nn.BatchNorm2d(512)
        self.bn1_1 = nn.BatchNorm2d(512)
        self.bn2 = nn.BatchNorm2d(256)
        self.bn3 = nn.BatchNorm2d(128)
        self.bn4 = nn.BatchNorm2d(64)
        self.bn5 = nn.BatchNorm2d(32)
        self.conv7 = nn.Conv2d(32, 7, 1)

        self.conv1 = nn.Conv2d(512, 512, 3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(512, 256, 3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(256, 128, 3, stride=1, padding=1)
        self.conv4 = nn.Conv2d(128, 64, 3, stride=1, padding=1)
        self.conv5 = nn.Conv2d(64, 32, 3, stride=1, padding=1)
        self.conv6 = nn.Conv2d(32, 7, 3, stride=1, padding=1)

        self._initialize_weights()
    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.ConvTranspose2d):
                assert m.kernel_size[0] == m.kernel_size[1]
                initial_weight = get_upsampling_weight(
                    m.in_channels, m.out_channels, m.kernel_size[0])
                m.weight.data.copy_(initial_weight)

    def forward(self, input):
        x = input

        x1 = self.block1(x) # 64, 256, 256
        x2 = self.block2(x1) # 128, 128, 128
        x3 = self.block3(x2) # 256, 64, 64
        x4 = self.block4(x3) # 512, 32, 32
        x5 = self.block5(x4) # 512, 16, 16

        x5 = self.conv1(x5)
        u1 = self.relu(self.bn1(self.dev1(x5))) # 512, 32, 32

        u2 = self.relu(self.bn1_1(self.dev1_1(u1+x4))) # 256, 64, 64
        u2 = self.conv2(u2) # 256, 32, 32

        u3 = self.relu(self.bn2(self.dev2(u2+x3))) # 128, 128, 128
        u3 = self.conv3(u3) # 256, 64, 64

        u4 = self.relu(self.bn3(self.dev3(u3+x2))) # 64, 256, 256
        u4 = self.conv4(u4) # 128, 128, 128

        u5 = self.relu(self.bn4(self.dev4(u4+x1))) # 32, 512, 512
        u5 = self.conv5(u5) # 64, 256, 256

        output = self.conv6(u5)
        # pdb.set_trace()
        return output
```

Backbone

2-4

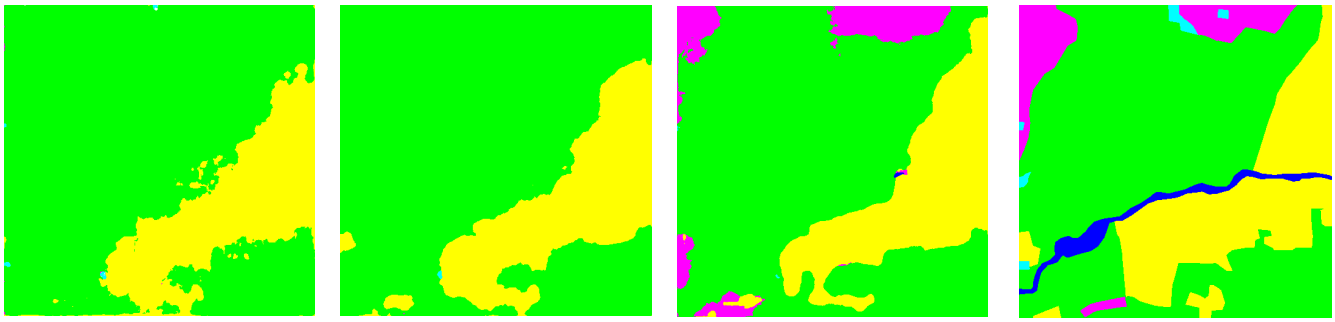
[0010_sat.jpg]

10th epoch | 25th epoch | 53th epoch | gt



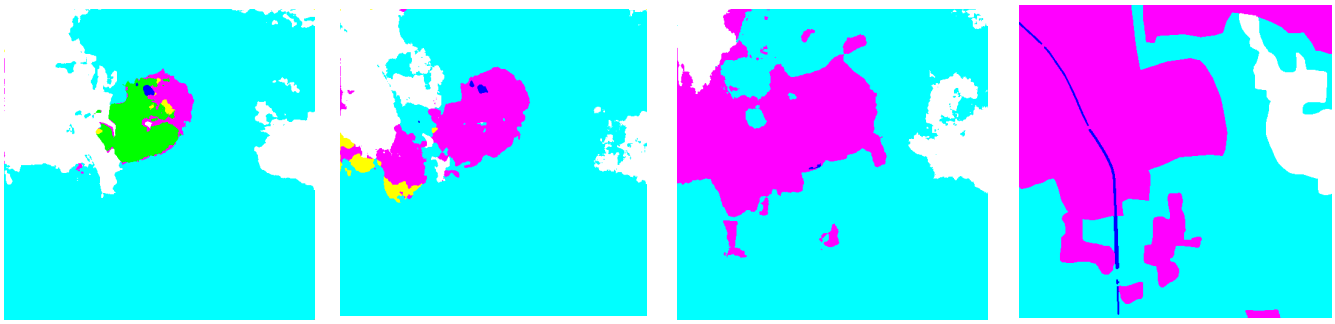
[0097_sat.jpg]

10th epoch | 25th epoch | 53th epoch | gt



[107_sat.jpg]

10th epoch | 25th epoch | 53th epoch | gt



2-5

兩個 segmentation model 差異除了 backbone 有沒有加 batch normalize 之外，另外就是後面 upsample 的做法：improvement model 我採用 fcn8s，與 fcn32s 不同的是它結合了 downsample 後的 feature 資訊，因此相較於 fcn32s，fcn8s 可以保留較多原始圖片的 context，因為 downsample 主要是取出 local context 內容資訊，upsample 主要取出 location 位置資訊，因此若

可以用一些 skip connection（concat or pixel wise sum），可以有效同時保留 context information 與 spatial information。

Baseline model 我取第 50 epoch 的 model，validation meanIOU 成績為 67%。
Improvement model 我取第 53 epoch 的 model，validation meanIOU 成績為 72%。

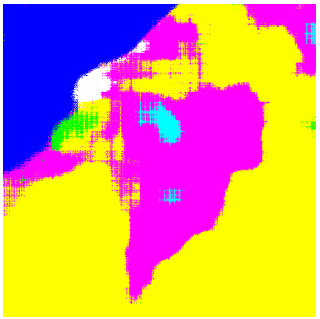
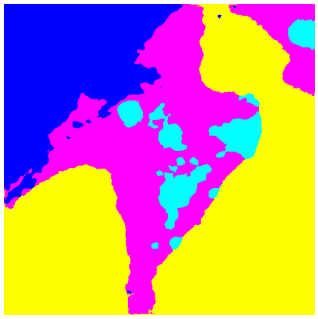
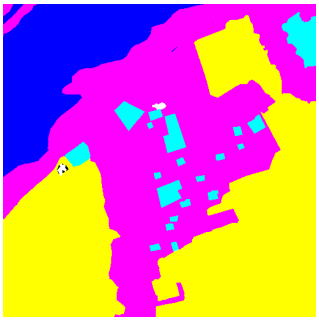
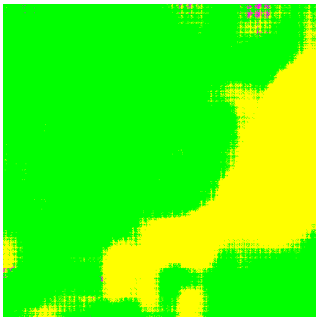
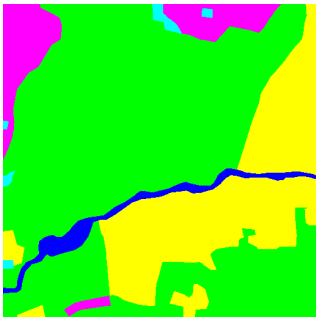
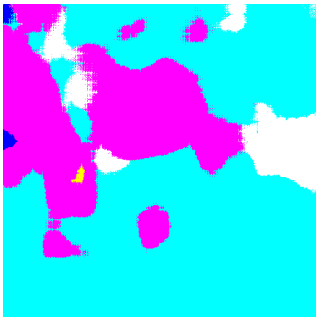
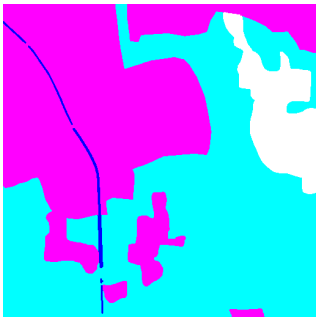
baseline

```
class #0 : 0.65544
class #1 : 0.87655
class #2 : 0.32324
class #3 : 0.79471
class #4 : 0.73741
class #5 : 0.65466

mean_iou: 0.673666
```

improve

```
class #0 : 0.77022
class #1 : 0.89533
class #2 : 0.40155
class #3 : 0.81994
class #4 : 0.77723
class #5 : 0.70049

mean_iou: 0.727459
```

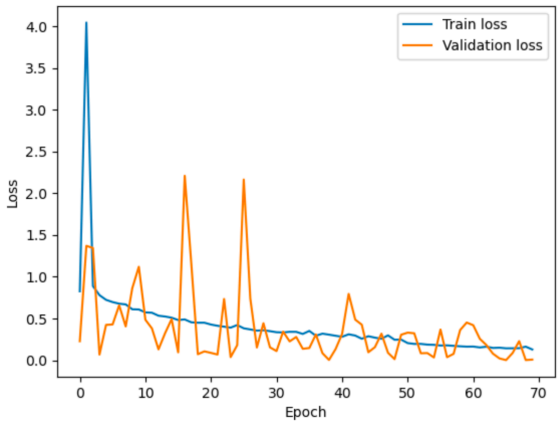
由下圖可以看到，兩者類別的位置資訊大致上都接近 ground truth，但 improvement model 的 segmentation 邊緣形狀比較接近 ground truth，也比較銳利，因此 improvement model 可以有效的保留 context 資訊。

	baseline	improve	gt
[0010_sat.jpg]			
[0097_sat.jpg]			
[107_sat.jpg]			

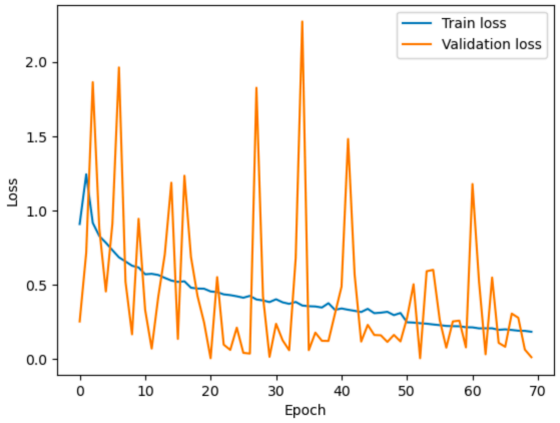
為了使 meanIOU 成績更好，我有使用 torch.optim 的 lr_schedular，每經過 50 個 epoch learning rate 就會乘上 0.5 下降，讓整體（train & validate）的 meanIOU 提升，另外 improvement model 中的每一層 convTransport 我都有 initialize 成 bilinear upsample，好處是可以在起始 epoch 就有不錯的成績，應可以更快收斂，另外，為了解決 overfitting，在做 improvement model 時我也有額外做 data augmentation，讓 training data 有 0.5 的機率水平翻轉，以及 0.5 的機率垂直翻轉，讓原本 improvement 的 meanIOU 從 70% 提升為 72%。

下方為有初始化及無初始化的 loss 圖，可以看到沒有初始化的 validation loss 超過 1.5 的次數很多，震盪相對多。

w initialization



w/o initialization



reference: **https://arxiv.org/pdf/1605.06211.pdf**
“no collaborators”