Deep Learning for Computer Vision HW1

contributer <weihsinyeh (https://github.com/DLCV-Fall-2024/dlcv-fall-2024-hw1-weihsinyeh)>
姓名：葉惟欣
學號：R13922043
系級：資工所 113

Problem 1: Self-supervised pre-training for image classification (50%)

(5%) Describe the implementation details of your SSL method for pre-training the ResNet50 backbone.

(including but not limited to the name of the SSL method & data augmentation techniques you used, learning rate schedule, optimizer, and batch size setting for this pre-training phase)

SSL method & data augmentation techniques

我的自監督學習使用的是方法為 Bootstrap Your Own Latent A New Approach to Self-Supervised Learning (https://arxiv.org/pdf/2006.07733)。
source : Bootstrap Your Own Latent A New Approach to Self-Supervised Learning

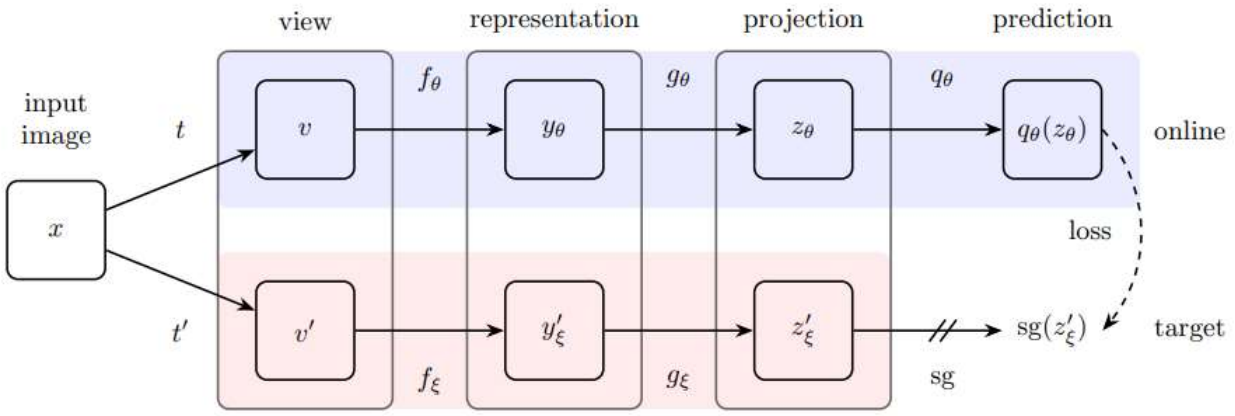


Figure 2: BYOL's architecture. BYOL minimizes a similarity loss between $q_\theta(z_\theta)$ and $\mathrm{sg}(z'_\xi)$, where θ are the trained weights, ξ are an exponential moving average of θ and sg means stop-gradient. At the end of training, everything but f_θ is discarded, and y_θ is used as the image representation.

其將一張圖在最一開始 resize 為寬 128,高 128，接著複製成兩張圖 t 與 t' 分別做不同的資料前處理 data argumentation 的方式包含
(一)改變圖像的顏色屬性(亮度、對比度、飽和度和色調)

(二)轉成灰階圖

(三)水平翻轉

(四)增加 GaussianBlur

(五)先上下左右 padding 二十個像素後隨機切割取寬 128,高 128 的影像範圍

最後做正規化使用 ImageNet 的平均 `[0.485, 0.456, 0.406]` 與標準差 `[0.229, 0.224, 0.225]` 。

```
def data_argument():
    return transforms.Compose([
        transforms.Resize((image_size, image_size)),
        transforms.RandomApply([transforms.ColorJitter(0.8, 0.8, 0.8, 0.2)],
                               p = 0.3),
        transforms.RandomGrayscale(p = 0.2),
        transforms.RandomHorizontalFlip(),
        transforms.RandomApply([transforms.GaussianBlur((3, 3), (1.0, 2.0))],
                               p = 0.2),
        transforms.Pad(20),
        transforms.RandomResizedCrop((image_size, image_size)),
        transforms.ToTensor(),
        transforms.Normalize(mean = torch.tensor([0.485, 0.456, 0.406]),
                             std = torch.tensor([0.229, 0.224, 0.225]))])
```

得到兩張圖的 feature 後 $v = t(x_i)$ 與 $v' = t'(x_i)$ 後。接著都給兩個網路 online network (θ are the trained weights) 與 target network(ξ are an exponential moving average of θ)，這兩個網路為 backbone `resnet50` 如同下面演算法(第五、六行)，將 $t(x_i)$ 與 $t'(x_i)$ 都給 online network 與 target network 作為輸入 features。相當於將兩種 network feature f_θ 函數與 target feature f_ξ 函數都見過兩種 v 與 v' 得到兩者: $f_\theta(v)$、$f_\theta(v')$ 與兩者: $f_\xi(v)$、$f_\xi(v')$。
再將這些 project (g 函數)成 feature $g_\theta(f_\theta(v))$、$g_\theta(f_\theta(v'))$ 與 $g_\xi(f_\xi(v))$、$g_\xi(f_\xi(v'))$ 。

而上面的 project (g 函數) 在實作中也被視為 backbone `resnet50` 的一部分。

前兩者拿去做 predict (q 函數) $q_\theta(g_\theta(f_\theta(v)))$、$q_\theta(g_\theta(f_\theta(v')))$ 與後兩者: $g_\xi(f_\xi(v))$、$g_\xi(f_\xi(v'))$去計算 loss 前還會將兩個 $f_\theta(v)$。這裡在實作中使用 MLP 。

Algorithm 1: BYOL: **B**ootstrap **Y**our **O**wn **L**atent

Inputs :

$\mathcal{D}$, $\mathcal{T}$, and $\mathcal{T}'$	set of images and distributions of transformations
θ, f_θ, g_θ, and q_θ	initial online parameters, encoder, projector, and predictor
ξ, f_ξ, g_ξ	initial target parameters, target encoder, and target projector
optimizer	optimizer, updates online parameters using the loss gradient
K and N	total number of optimization steps and batch size
$\{\tau_k\}_{k=1}^K$ and $\{\eta_k\}_{k=1}^K$	target network update schedule and learning rate schedule

for $k = 1$ **to** K **do**
  $\mathcal{B} \leftarrow \{x_i \sim \mathcal{D}\}_{i=1}^N$ // sample a batch of N images
  for $x_i \in \mathcal{B}$ **do**
    $t \sim \mathcal{T}$ and $t' \sim \mathcal{T}'$ // sample image transformations
    $z_1 \leftarrow g_\theta(f_\theta(t(x_i)))$ and $z_2 \leftarrow g_\theta(f_\theta(t'(x_i)))$ // compute projections
    $z'_1 \leftarrow g_\xi(f_\xi(t'(x_i)))$ and $z'_2 \leftarrow g_\xi(f_\xi(t(x_i)))$ // compute target projections
    $l_i \leftarrow -2 \cdot \left(\frac{\langle q_\theta(z_1), z'_1 \rangle}{\|q_\theta(z_1)\|_2 \cdot \|z'_1\|_2} + \frac{\langle q_\theta(z_2), z'_2 \rangle}{\|q_\theta(z_2)\|_2 \cdot \|z'_2\|_2} \right)$ // compute the loss for x_i
  end
  $\delta\theta \leftarrow \frac{1}{N} \sum_{i=1}^N \partial_\theta l_i$ // compute the total loss gradient w.r.t. θ
  $\theta \leftarrow \text{optimizer}(\theta, \delta\theta, \eta_k)$ // update online parameters
  $\xi \leftarrow \tau_k \xi + (1 - \tau_k)\theta$ // update target parameters
end
Output : encoder f_θ

learning rate : 0.0005

optimizer :

```
optimizer = torch.optim.Adam(finetune_model.parameters(), lr=config.lr)
```

scheduler :

Gradually warm-up(increasing) learning rate in optimizer

Source Code (https://github.com/ildoonet/pytorch-gradual-warmup-lr/blob/master/warmup_scheduler/scheduler.py)

batch size : 64

backbone

ResNet50

```
backbone        = models.resnet50(weights=None)
```

```
<bound method Module.modules of ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bia
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mod
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runnin
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runnin
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runnin
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runnin
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runnin
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runnin
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runni
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runni
      )
    )
```

```
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runni
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runni
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runni
      (relu): ReLU(inplace=True)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_runn
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_runn
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_runn
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_runn
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_runn
      (relu): ReLU(inplace=True)
    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_runn
      (relu): ReLU(inplace=True)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_runn
      (relu): ReLU(inplace=True)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_runn
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_runn
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
```

```
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_runn
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_runn
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=1000, bias=True)
)>
```

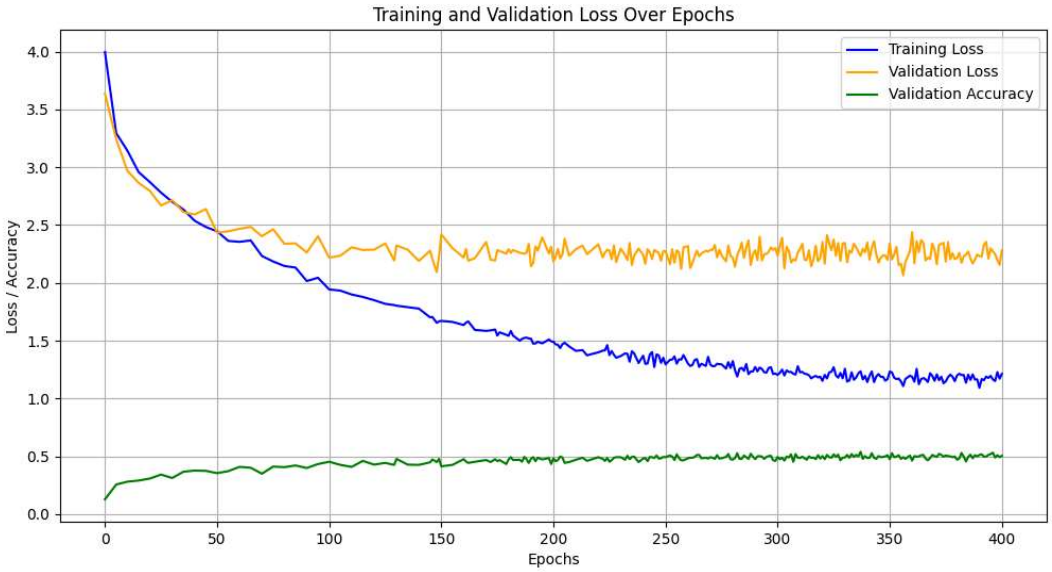
All settings(20%)

Setting	Pre-training (Mini-ImageNet)	Fine-tuning (Office-Home dataset)	Validation accuracy
A	-	Train full model (backbone + classifier)	Epoch 40 : 0.2438423645320197
B	w/ label (TAs have provided this backbone)	Train full model (backbone + classifier)	Epoch 137 : 0.5172413793103449
C	w/o label (Your SSL pre-trained backbone)	Train full model (backbone + classifier)	Epoch 146 : 0.4482758620689655
D	w/ label (TAs have provided this backbone)	Fix the backbone. Train classifier only	Epoch 339 : 0.22413793103448276
E	w/o label (Your SSL pre-trained backbone)	Fix the backbone. Train classifier only	Epoch 109 : 0.29802955665024633

Discuss and Analyze the results

下圖為隨著 epoch 增加，Setting C 的 Training/Validation Loss 與 Validation Accuracy的變化。
原先我是設定最大的訓練 epoch 為 `400` 。其 Validation Accuracy 可以達到 `0.5394088669950738` 。但是在 epoch `50` 之後的時候 validation loss 與 training loss 的差距已經開始變大了。同時在 epoch `150` 之後 validation loss 沒有明顯的下降了，且 validation accuracy 也沒有明顯的提高。所以我最後選擇 Setting C 最好的 epoch

為 epoch 150 附近的最高分的也就是 Epoch 146 : 0.4482758620689655。



fix backbone 的效果較差，沒有 pretrained 在較大的 dataset 上效果也會較差。

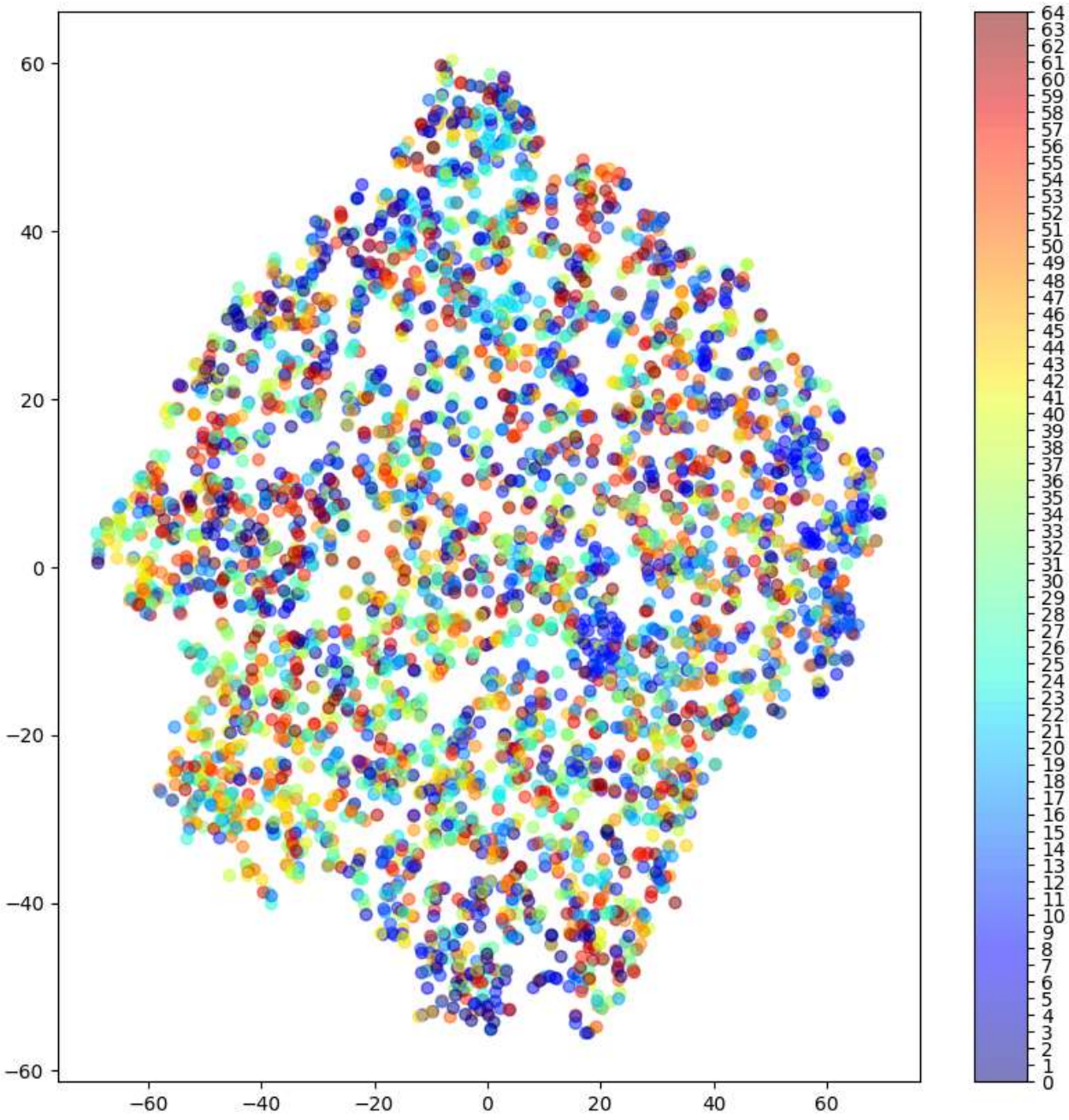
Visualize - T-SNE(5%)

Visualize the learned visual representation of setting C on the train set by implementing t-SNE (t-distributed Stochastic Neighbor Embedding) on the output of the second last layer.
總共六十五個類別。

Setting C

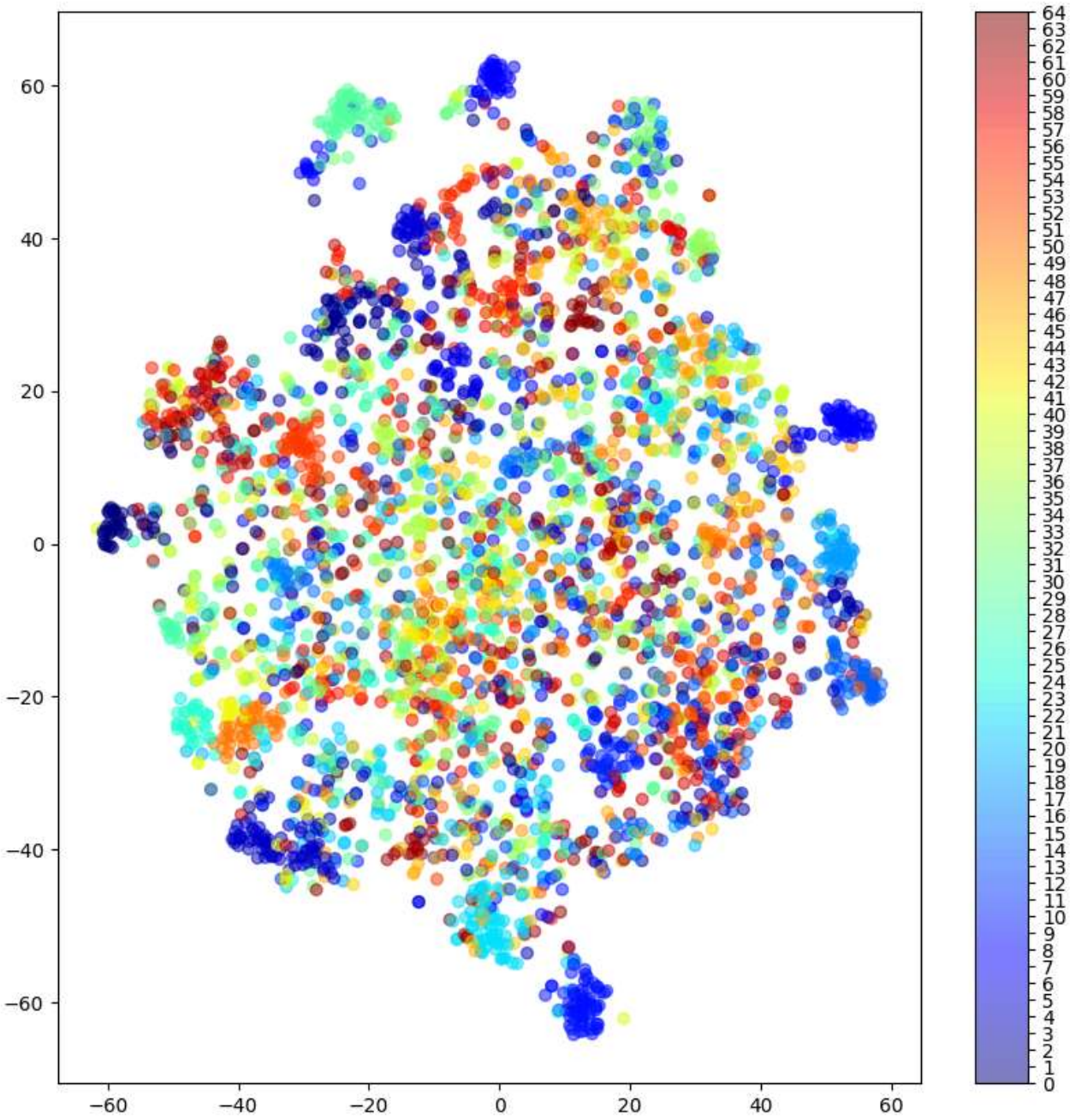
從一開始 epoch 為 0 的時後，不同類別的點點(在下放圖表代表不同顏色)的 embedding/feature 表示可能會很接近，所以在 epoch 為 0 時，他們在 t-SNE 的圖上不同類別的 embedding/feature 會重疊。沒有呈現一團團的。

Epoch 0



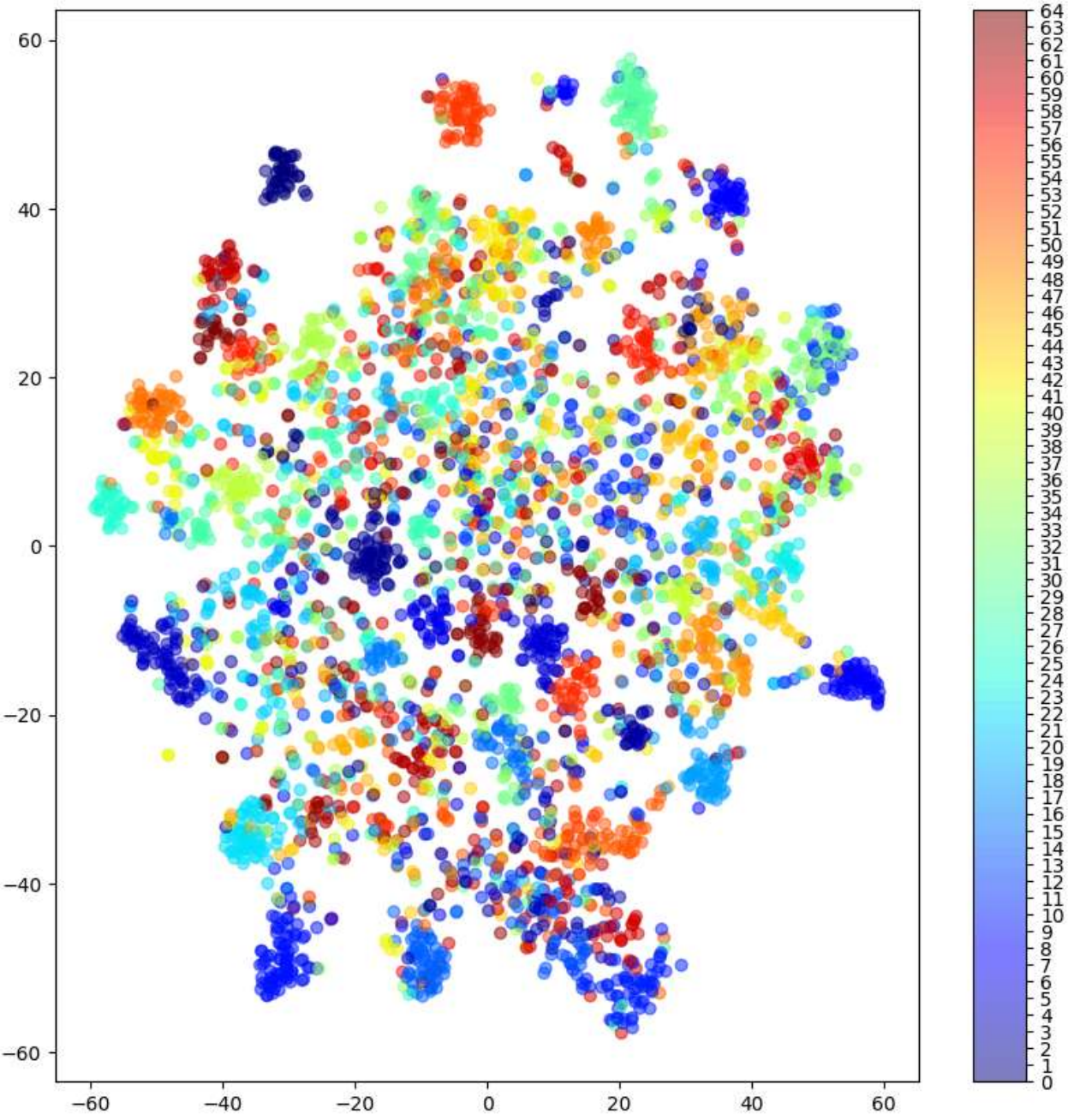
在 epoch 75 可以看到外圍有些顏色相同(相同類別的點)已經開始成一團聚集了。代表有些相同類別的 embedding/feature 會很像。

Epoch 75



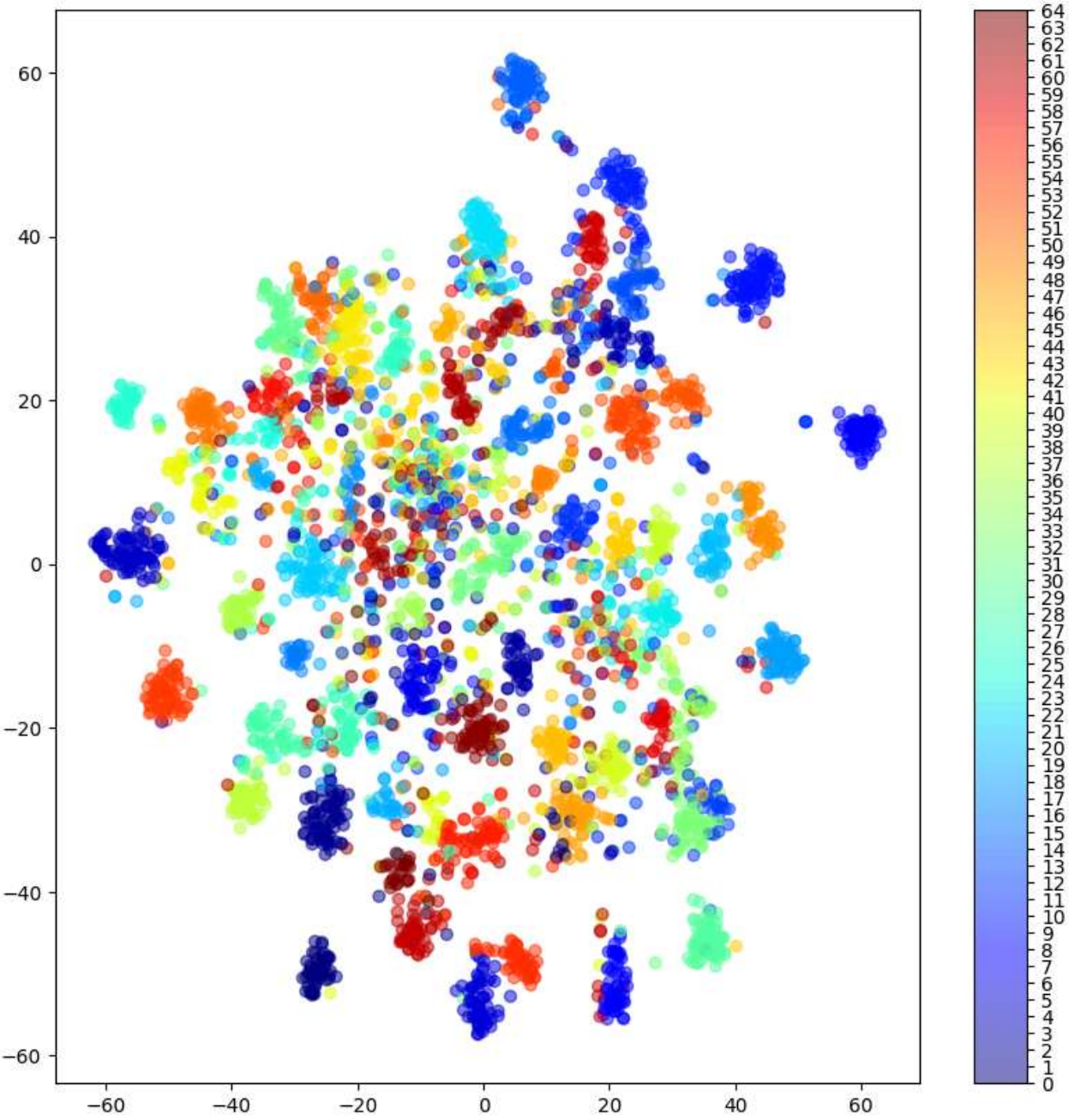
在 epoch 150 的時候可以看到顏色相同(相同類別的點)已經開始成一團聚集的群體數量變多了

Epoch 150



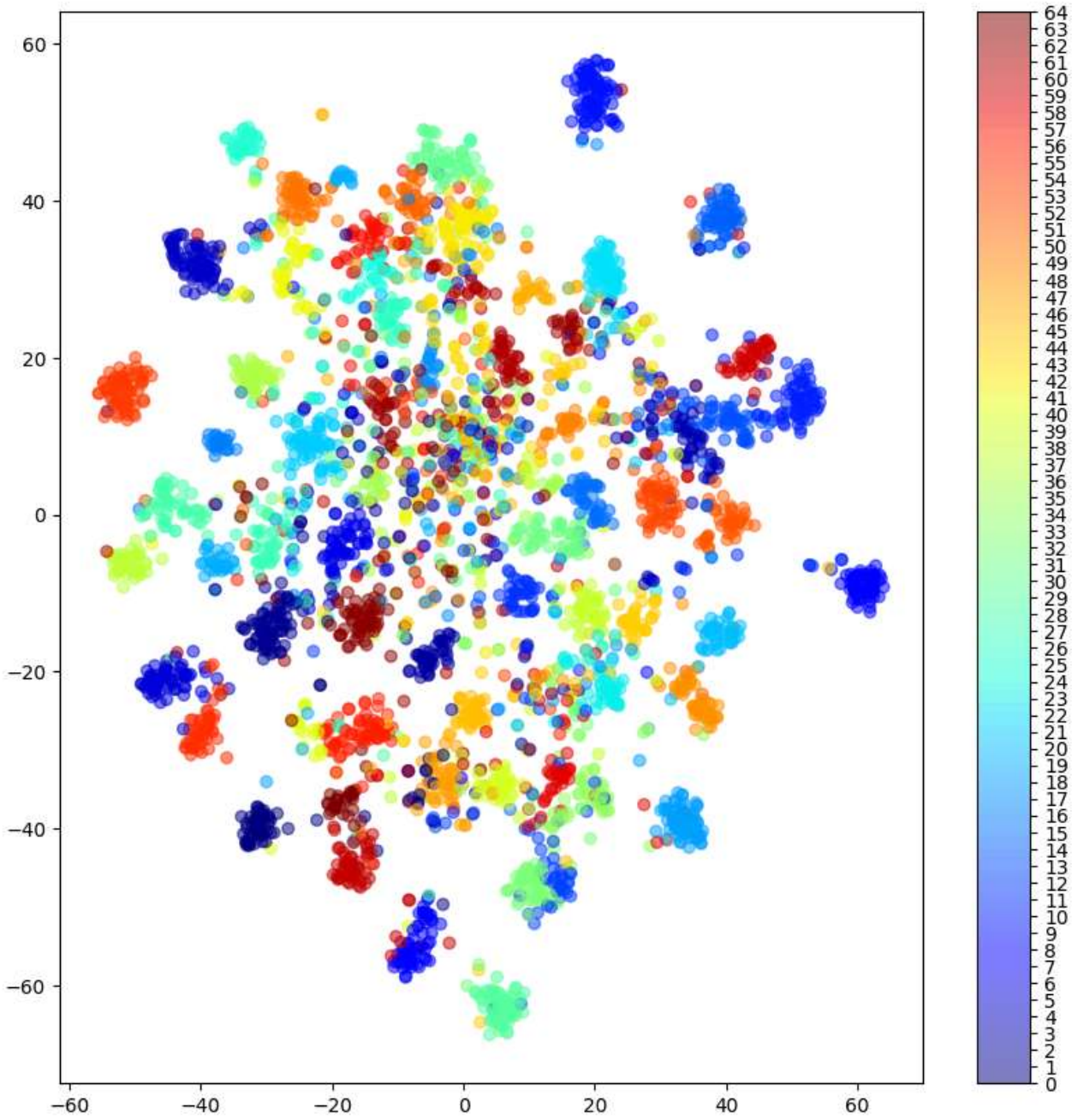
在 epoch 325 的時候中間的區域 embedding/feature 也可以分得更好，也就是原先在中間embedding/feature 不同類別可能也很像。但現在顏色相同(相同類別的點)已經開始成一團聚集了。

Epoch 325



跟 epoch 325 的時候表現差不多。

Epoch 395



Problem 2: Semantic segmentation (50%)

Report (20%)

1. (3%) Draw the network architecture of your VGG16-FCN32s model (model A).

參考 pytorch 的：

https://github.com/pytorch/vision/blob/main/torchvision/models/vgg.py#L43

(https://github.com/pytorch/vision/blob/main/torchvision/models/vgg.py#L43)

並將後面的 Fully Connected Networks 改成 Fully Convoultion Networks。

參考 Fully Convolutional Networks for Semantic Segmentation

(https://arxiv.org/pdf/1605.06211)

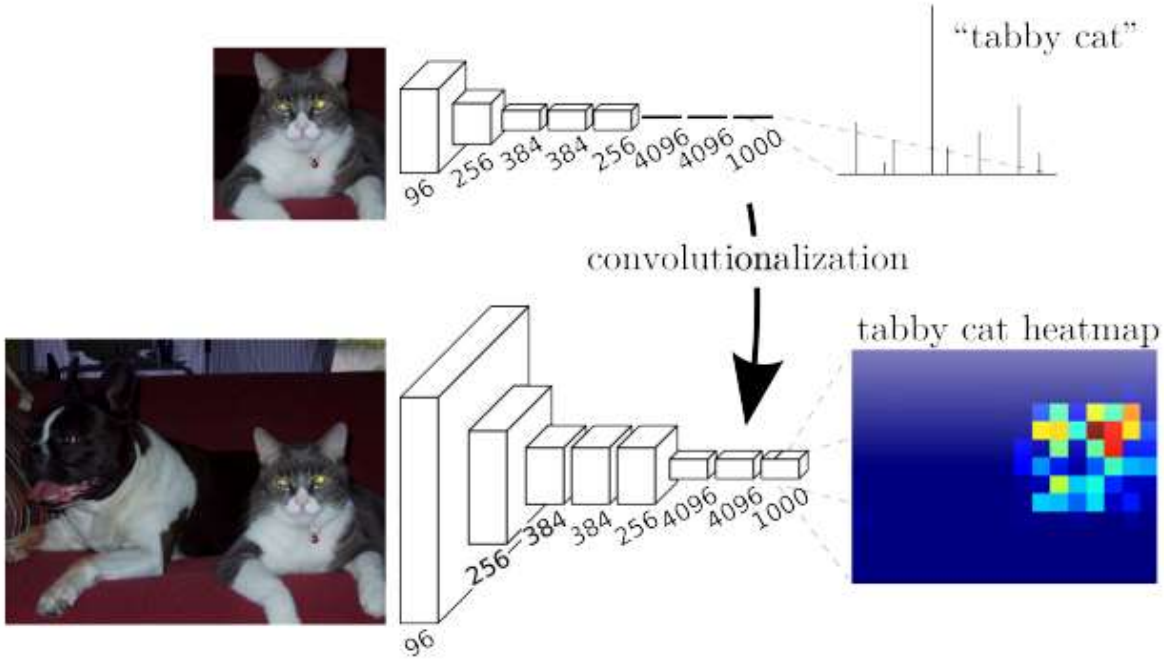
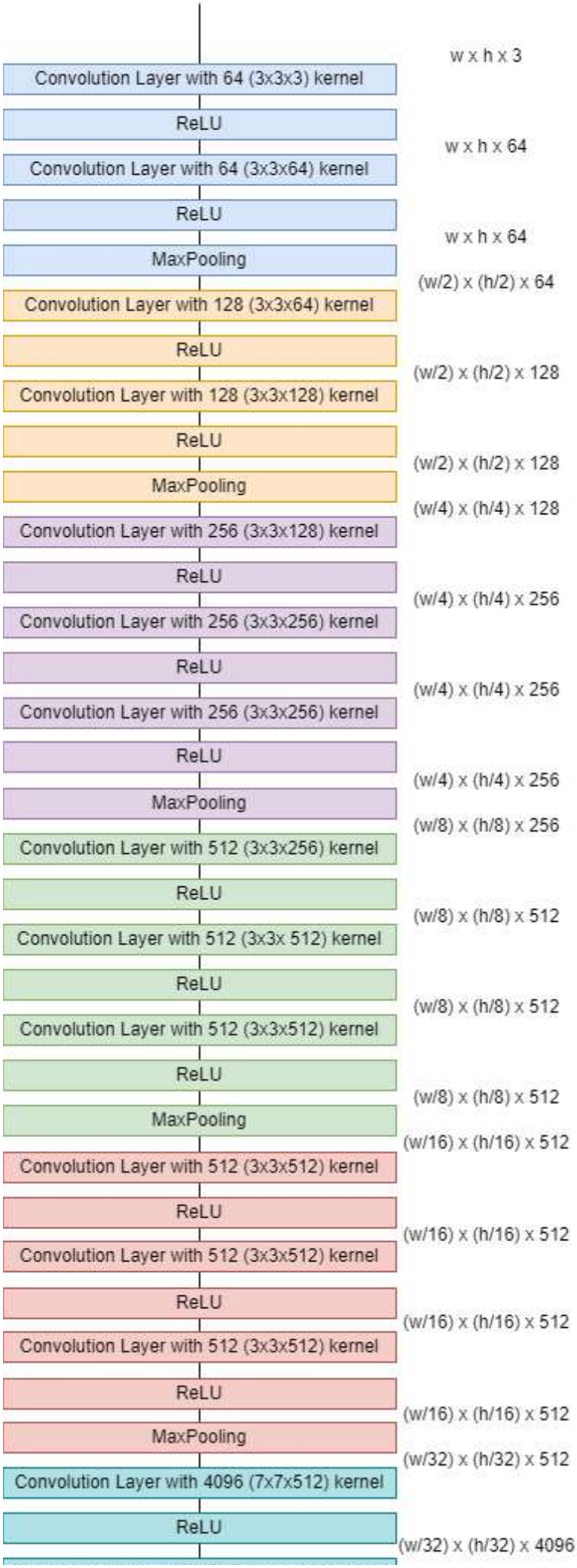
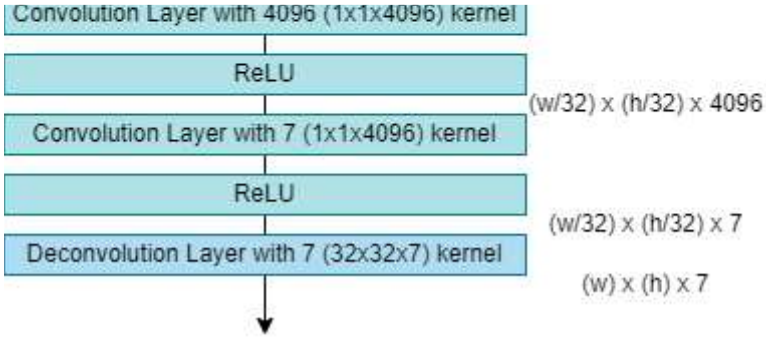


Fig. 2. Transforming fully connected layers into convolution layers enables a classification net to output a spatial map. Adding differentiable interpolation layers and a spatial loss (as in Figure 1) produces an efficient machine for end-to-end pixelwise learning.

架構圖如下：


w x h x 3
Convolution Layer with 64 (3x3x3) kernel
ReLU
w x h x 64
Convolution Layer with 64 (3x3x64) kernel
ReLU
w x h x 64
MaxPooling
(w/2) x (h/2) x 64
Convolution Layer with 128 (3x3x64) kernel
ReLU
(w/2) x (h/2) x 128
Convolution Layer with 128 (3x3x128) kernel
ReLU
(w/2) x (h/2) x 128
MaxPooling
(w/4) x (h/4) x 128
Convolution Layer with 256 (3x3x128) kernel
ReLU
(w/4) x (h/4) x 256
Convolution Layer with 256 (3x3x256) kernel
ReLU
(w/4) x (h/4) x 256
Convolution Layer with 256 (3x3x256) kernel
ReLU
(w/4) x (h/4) x 256
MaxPooling
(w/8) x (h/8) x 256
Convolution Layer with 512 (3x3x256) kernel
ReLU
(w/8) x (h/8) x 512
Convolution Layer with 512 (3x3x 512) kernel
ReLU
(w/8) x (h/8) x 512
Convolution Layer with 512 (3x3x512) kernel
ReLU
(w/8) x (h/8) x 512
MaxPooling
(w/16) x (h/16) x 512
Convolution Layer with 512 (3x3x512) kernel
ReLU
(w/16) x (h/16) x 512
Convolution Layer with 512 (3x3x512) kernel
ReLU
(w/16) x (h/16) x 512
Convolution Layer with 512 (3x3x512) kernel
ReLU
(w/16) x (h/16) x 512
MaxPooling
(w/32) x (h/32) x 512
Convolution Layer with 4096 (7x7x512) kernel
ReLU
(w/32) x (h/32) x 4096



2. (3%) Draw the network architecture of the improved model (model B) and explain it differs from your VGG16-FCN32s model.

```
(model): DeepLabV3(
(backbone): IntermediateLayerGetter(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runnin
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fals
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fals
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
      )
```

```
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
      (relu): ReLU(inplace=True)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fal
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fals
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
```

```
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
      (relu): ReLU(inplace=True)
    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
      (relu): ReLU(inplace=True)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
      (relu): ReLU(inplace=True)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fal
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
```

```
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fa
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_
      (relu): ReLU(inplace=True)
    )
  )
)
(classifier): DeepLabHead(
  (0): ASPP(
    (convs): ModuleList(
      (0): Sequential(
        (0): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
        (2): ReLU()
      )
      (1): ASPPConv(
        (0): Conv2d(2048, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
        (2): ReLU()
      )
      (2): ASPPConv(
        (0): Conv2d(2048, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
        (2): ReLU()
      )
      (3): ASPPConv(
        (0): Conv2d(2048, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
        (2): ReLU()
      )
      (4): ASPPPooling(
        (0): AdaptiveAvgPool2d(output_size=1)
        (1): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
        (2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
        (3): ReLU()
      )
    )
    (project): Sequential(
      (0): Conv2d(1280, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
      (2): ReLU()
      (3): Dropout(p=0.5, inplace=False)
    )
```

```
        )
        (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
        (2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running
        (3): ReLU()
        (4): Conv2d(256, 7, kernel_size=(1, 1), stride=(1, 1))
      )
      (aux_classifier): FCNHead(
        (0): Conv2d(1024, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running
        (2): ReLU()
        (3): Dropout(p=0.1, inplace=False)
        (4): Conv2d(256, 7, kernel_size=(1, 1), stride=(1, 1))
      )
    )
```

我選擇 Rethinking Atrous Convolution for Semantic Image Segmentation (https://arxiv.org/pdf/1706.05587) 論文提出的 Deeplab v3 作為 model B。

`self.model = torch.hub.load('pytorch/vision:v0.10.0', 'deeplabv3_resnet50', pretrained=True)`

DeepLab v3 使用 atrous convolution 的概念相較於 model A 其用 Deconvolutional layers 去恢復 spatial resolution。首先跟 model A 一樣透過 Convolutional Neural Network 去獲得一個 low-resolution 的 feature，接著透過上面模型架構圖 `DeepLabHead` 中的 `ASPPConv` 也就是 Atrous Spatial Pyramid Pooling 。三個 3x3 atrous(dilated) 的 rates 為 [6, 12, 18] 以及ASPPPooling 裡面先做 AdaptiveAvgPool2d，再用一個 1x1 convolution。最後與 model A 一樣同樣使用 Fully Convolutional Network 作為 aux_classifier。

3. (1%) Report mIoUs of two models on the validation set.

Model	VGG16-FCN32s model (model A)	model B
mIoU	0.658093	epoch 585 : 0.741293

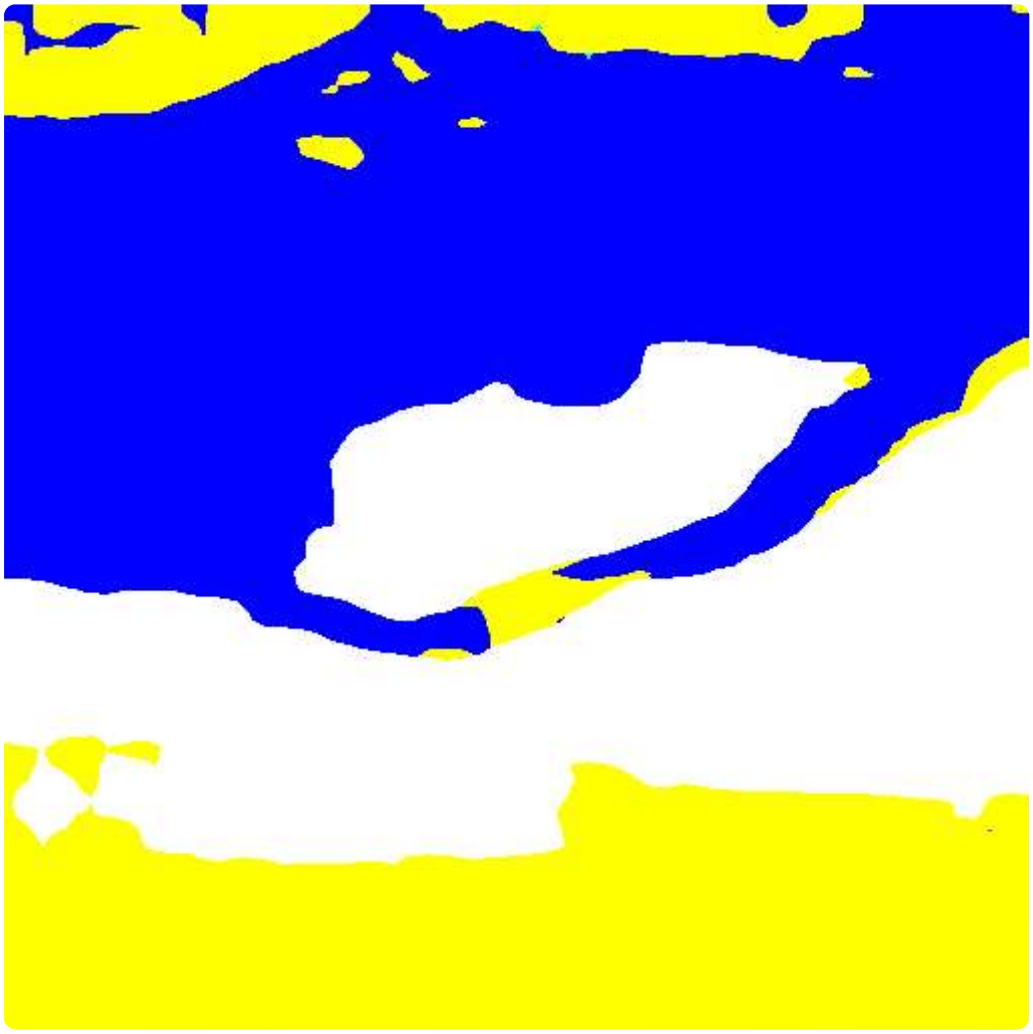
4. (3%) Show the predicted segmentation mask during the early, middle, and the final stage during the training process of the improved model.

“validation/0013_sat.jpg”,

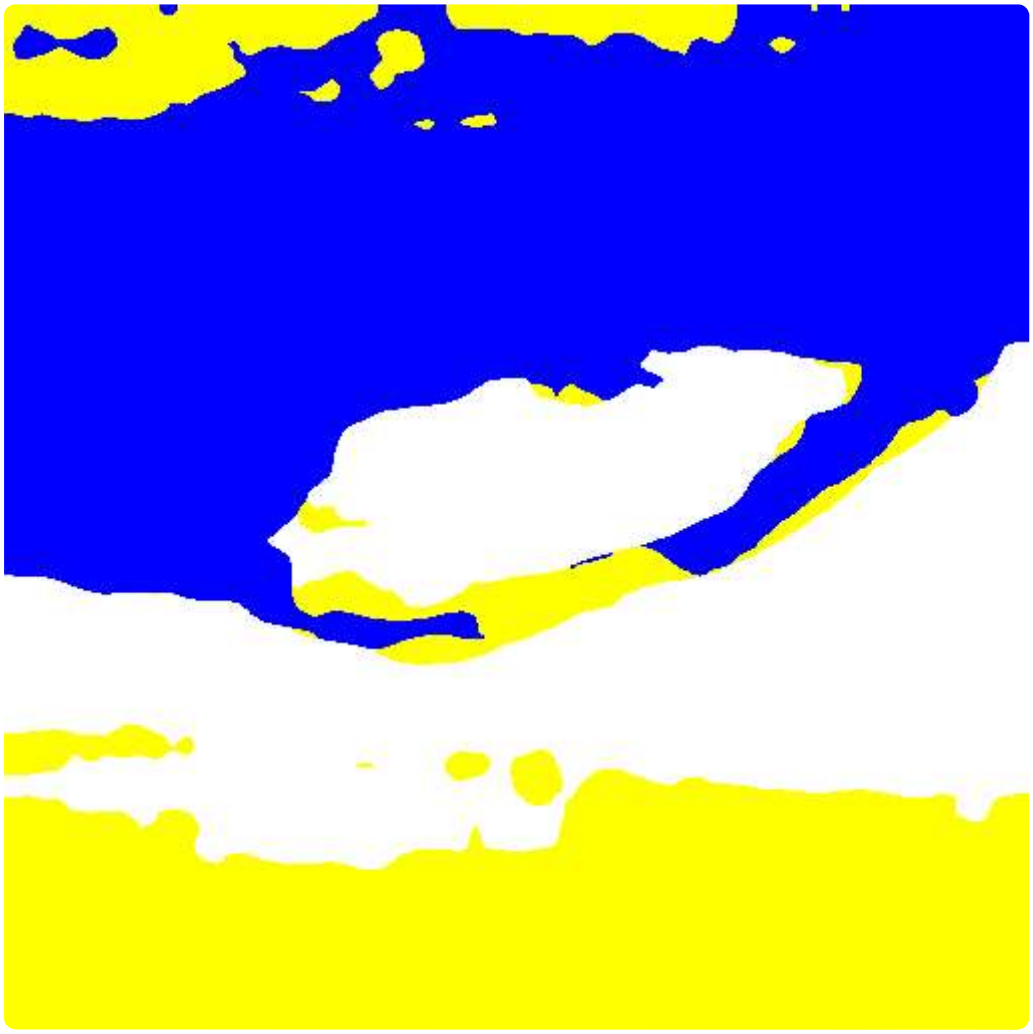
我選擇 epoch 585 作為最後的 final stage。也是我拿來做為 model B 的模型。

epoch 0

epoch 290

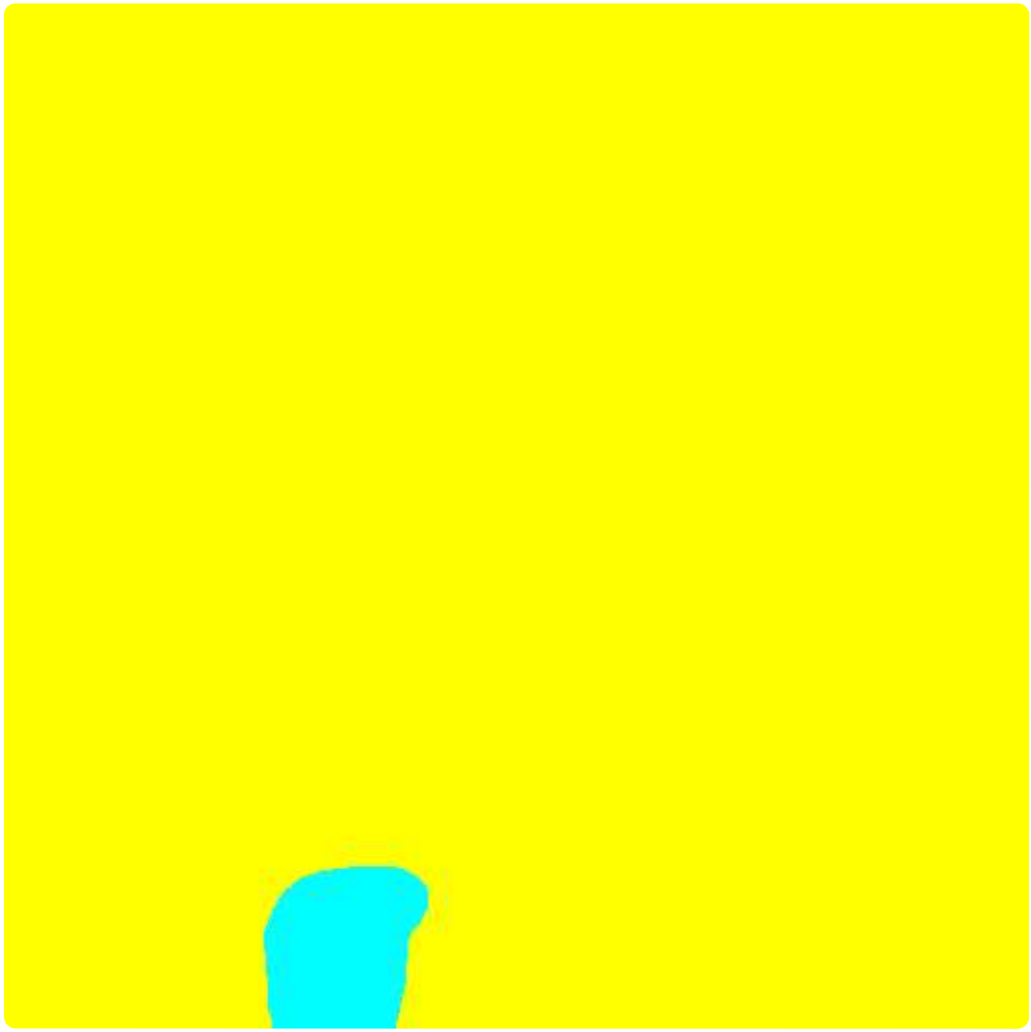


epoch 585

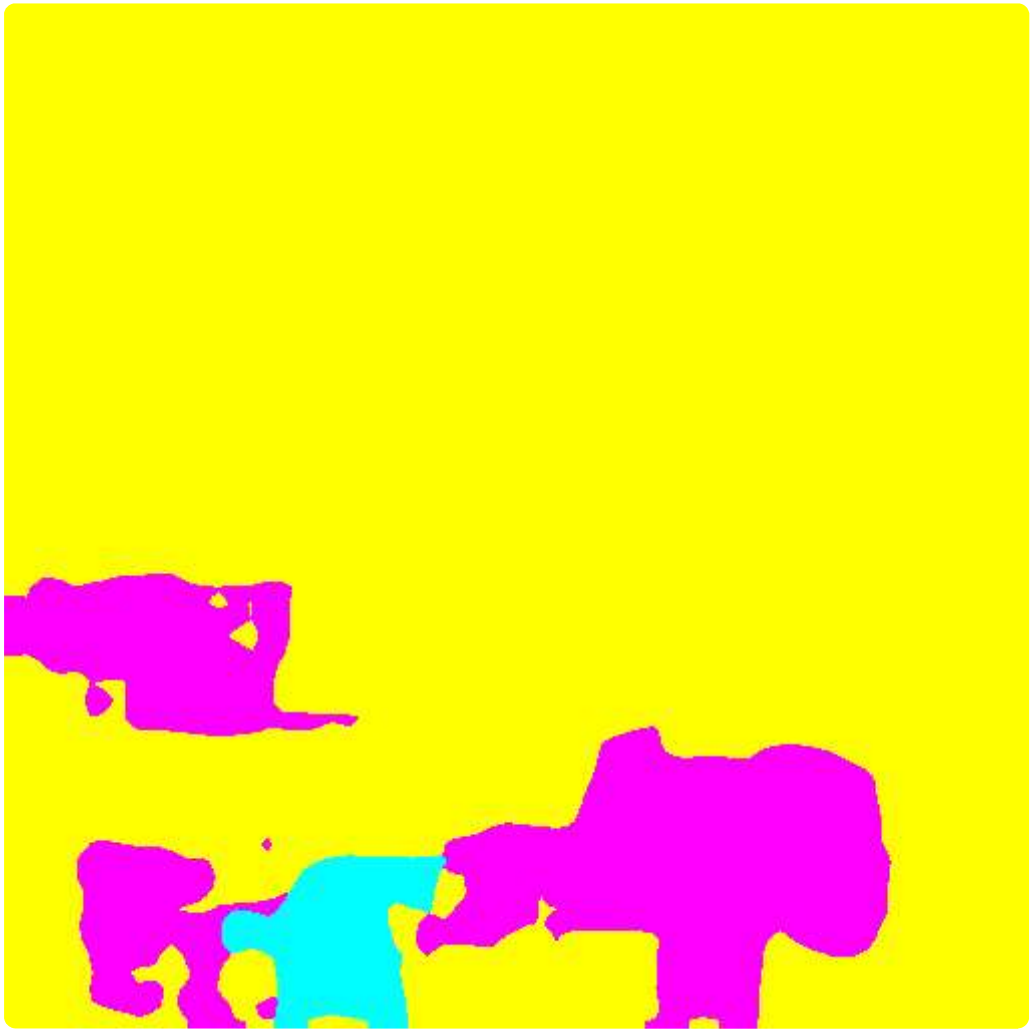


“validation/0062_sat.jpg”,

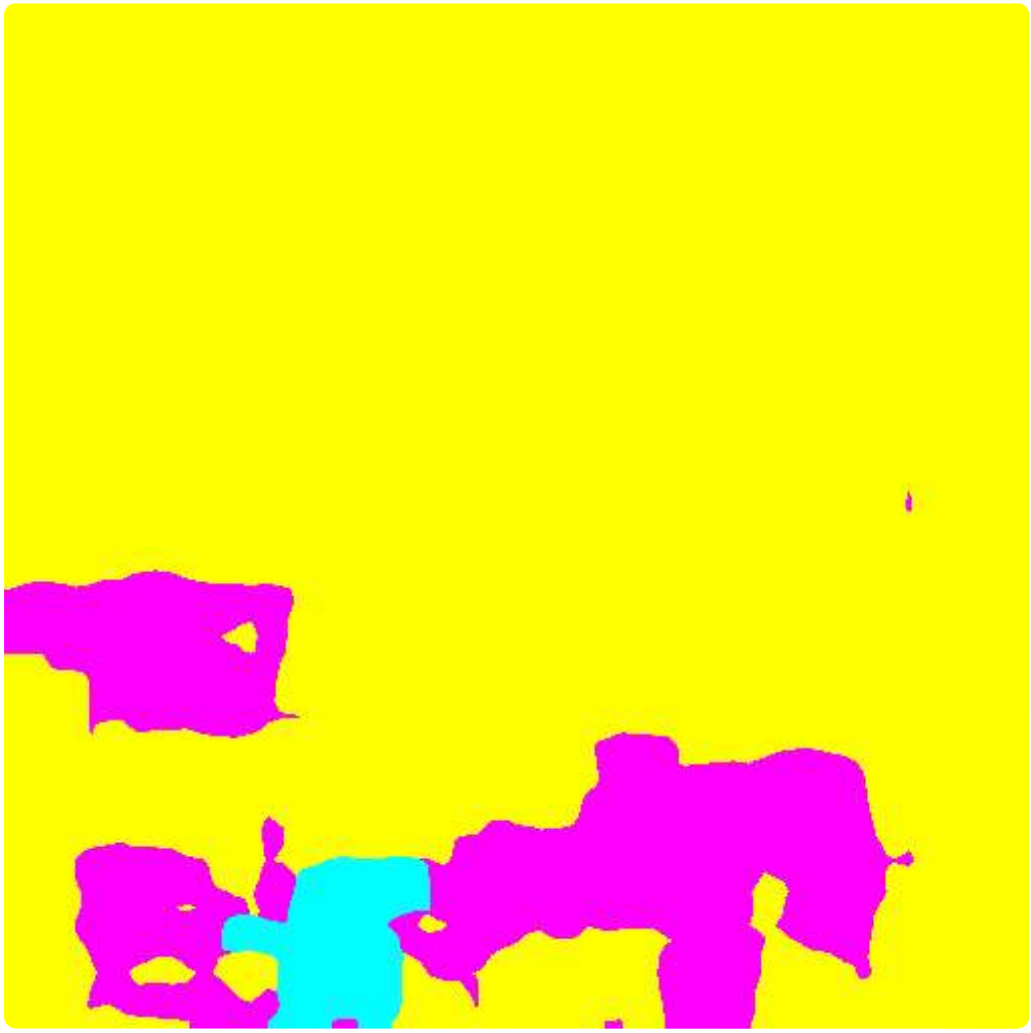
epoch 0



epoch 290

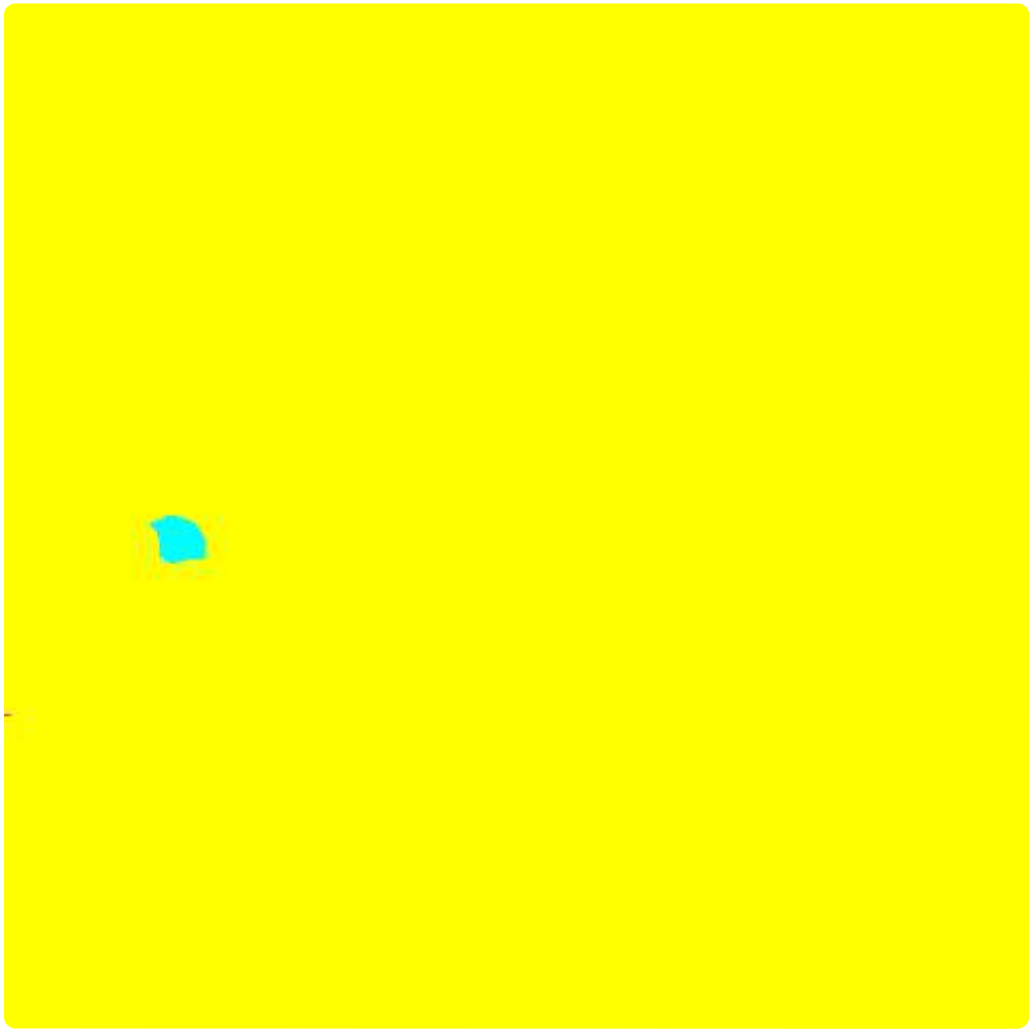


epoch 585

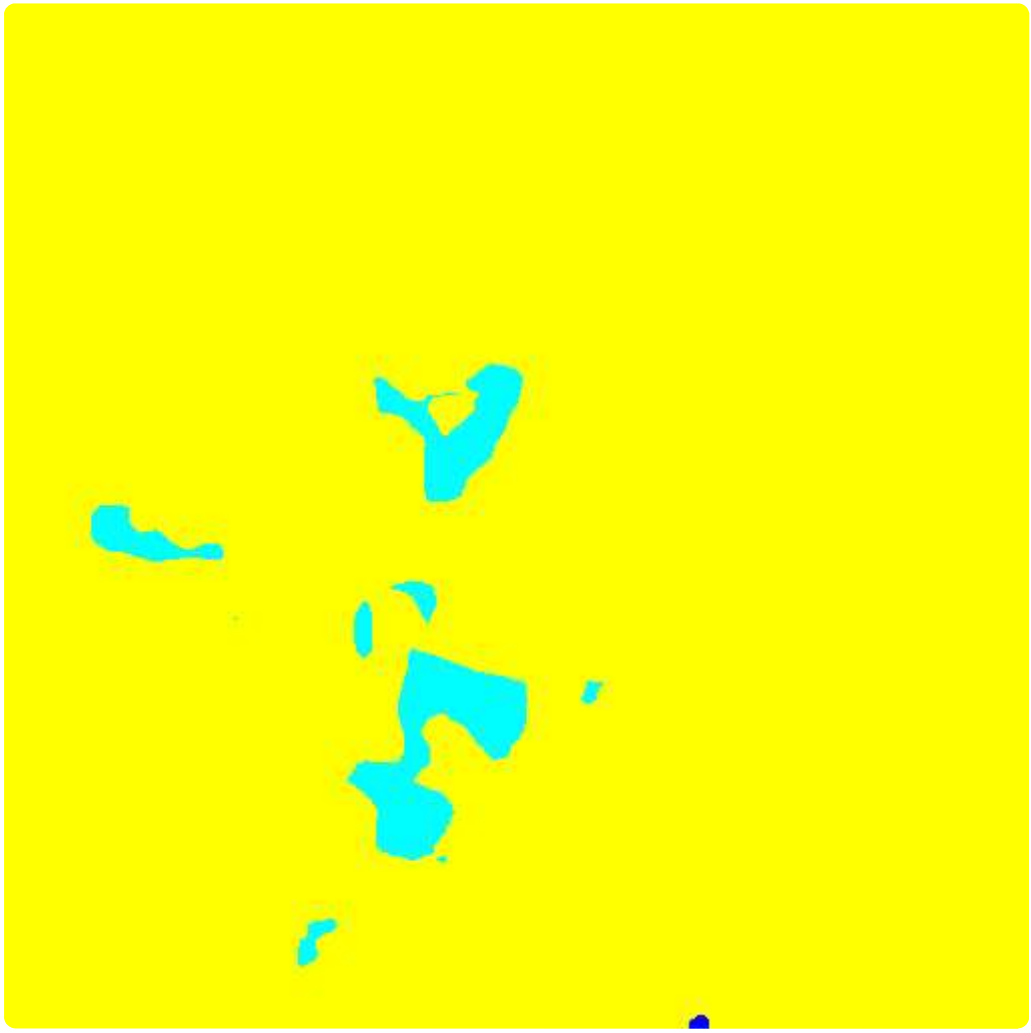


“validation/0104_sat.jpg”

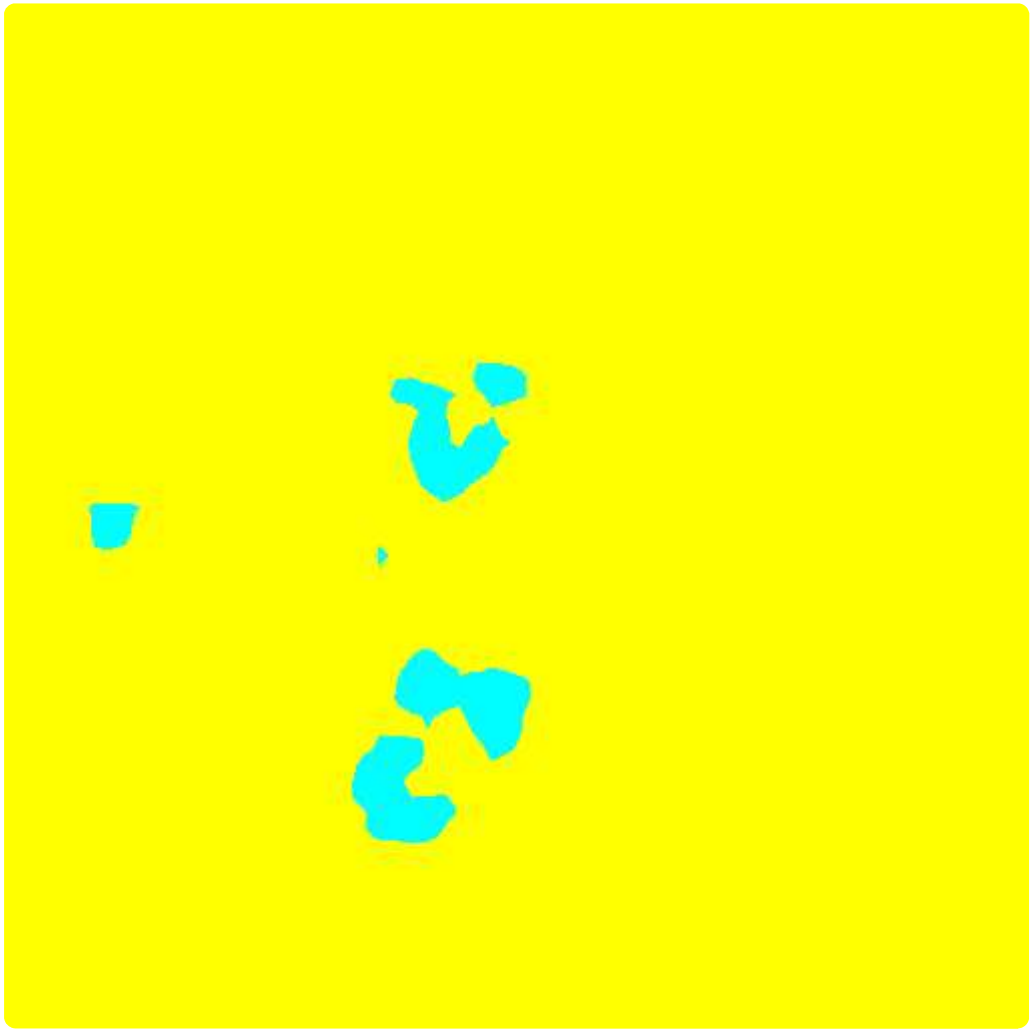
epoch 0



epoch 290



epoch 585



5. (10%) Use segment anything model (SAM) to segment three of the images in the validation dataset, report the result images and the method you use

我使用的程式碼為 `segment-anything/notebooks/automatic_mask_generator_example.ipynb`

Automatically generating object masks with SAM

使用的 pretrained 模型為 `sam_checkpoint = "sam_vit_h_4b8939.pth`

其方法為在網格的圖像取 sample 並對這些取到的樣本給予 prompt。然後去掉重複的 mask，然最後生成 mask。

```
masks = mask_generator.generate(image)
```

“validation/0013_sat.jpg”,

“validation/0062_sat.jpg”,

“validation/0104_sat.jpg”

原始連結：https://github.com/facebookresearch/segment-anything/blob/main/notebooks/automatic_mask_generator_example.ipynb (https://github.com/facebookresearch/segment-anything/blob/main/notebooks/automatic_mask_generator_example.ipynb)

Reference

Deep Residual Learning for Image Recognition (https://allen108108.github.io/blog/2019/10/29/%5B%E8%AB%96%E6%96%87%5D%20Deep%20Residual%20Learning%20for%20Image%20Recognition/)