DLCV HW1 Report

B07901039 電機四 劉知穎

Problem 1

1. Network architecture

   I ensemble 8 models, and use models from *torchvision.models* as feature extractors. The eight models and extra classification layers are listed as follow.

Feature extractor	Classification layer
Vgg16	Linear(512, 256) / ReLU / Linear(256, 50)
Resnet152	Linear(2048, 50)
dense161	Linear(2088, 50)
dense169	Linear(1664, 50)
resnext101_32x8d	Linear(2048, 50)
vgg19_bn	Linear(25088, 4096) / ReLU/ Dropout(0.5) / Linear(4096, 4096) / ReLU/ Linear(4096, 50)
vgg19_bn	Linear(25088, 4096) / ReLU/ Dropout(0.5) / Linear(4096, 50)
vgg19_bn	Linear(25088, 4096) / ReLU / Dropout(0.5) / Linear(4096, 4096) / ReLU / Dropout(0.5) / Linear(4096, 2048) /ReLU / Dropout(0.5) / Linear(2048, 50)

   To save space, I will only print out the architecture of vgg16, dense161, and resent152.

   Vgg16

```
model loaded from drive/MyDrive/DLCV/hw1/best_models_oldformat/vgg16-best.pth
VGG16_custom(
  (backbone): 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=(1, 1))
  (fc): Sequential(
    (0): Linear(in_features=512, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=50, bias=True)
  )
)
```

Figure 1: model architecture of vgg16

Dense161

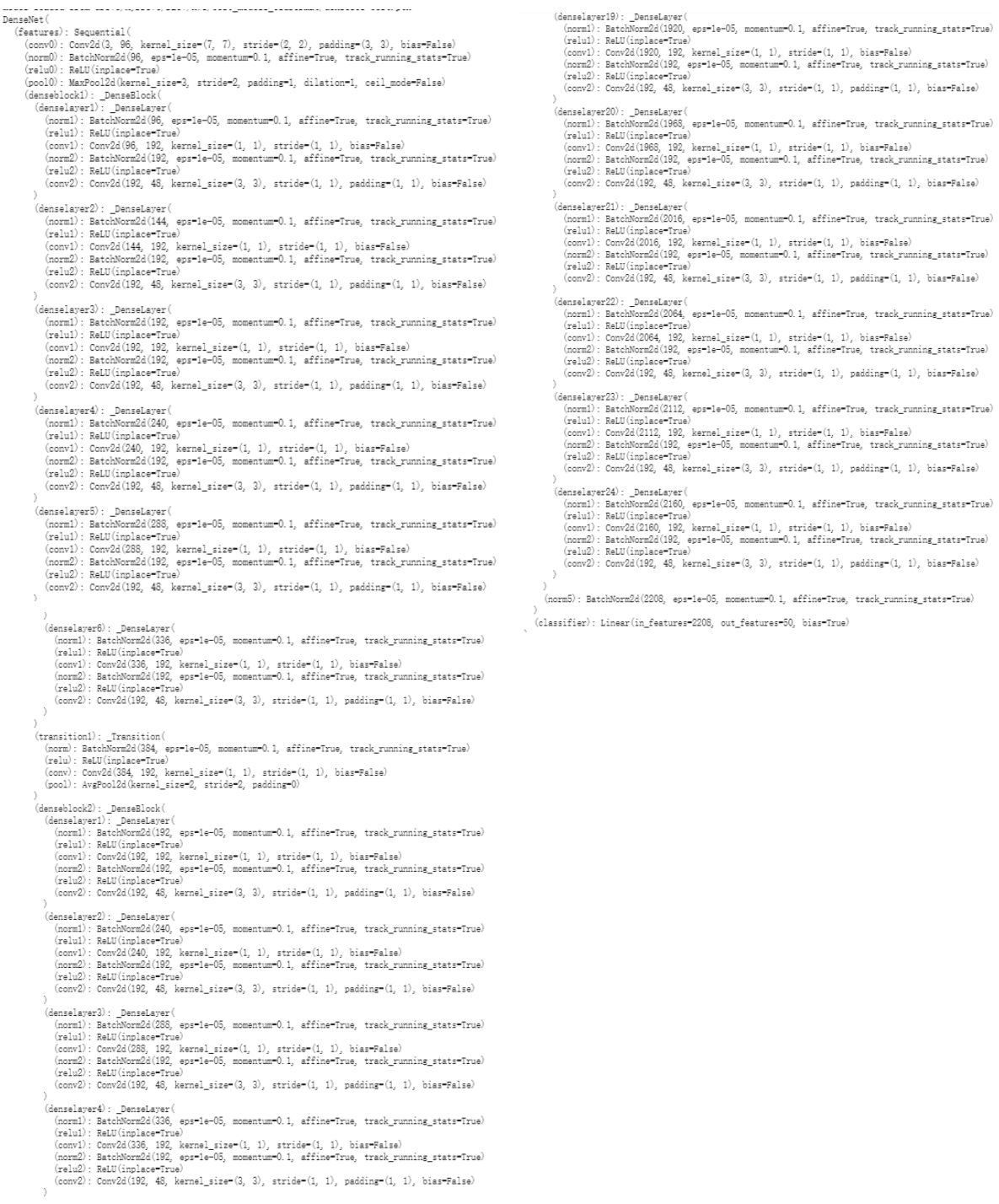


Figure 2: model architecture of dense161

Resnet152

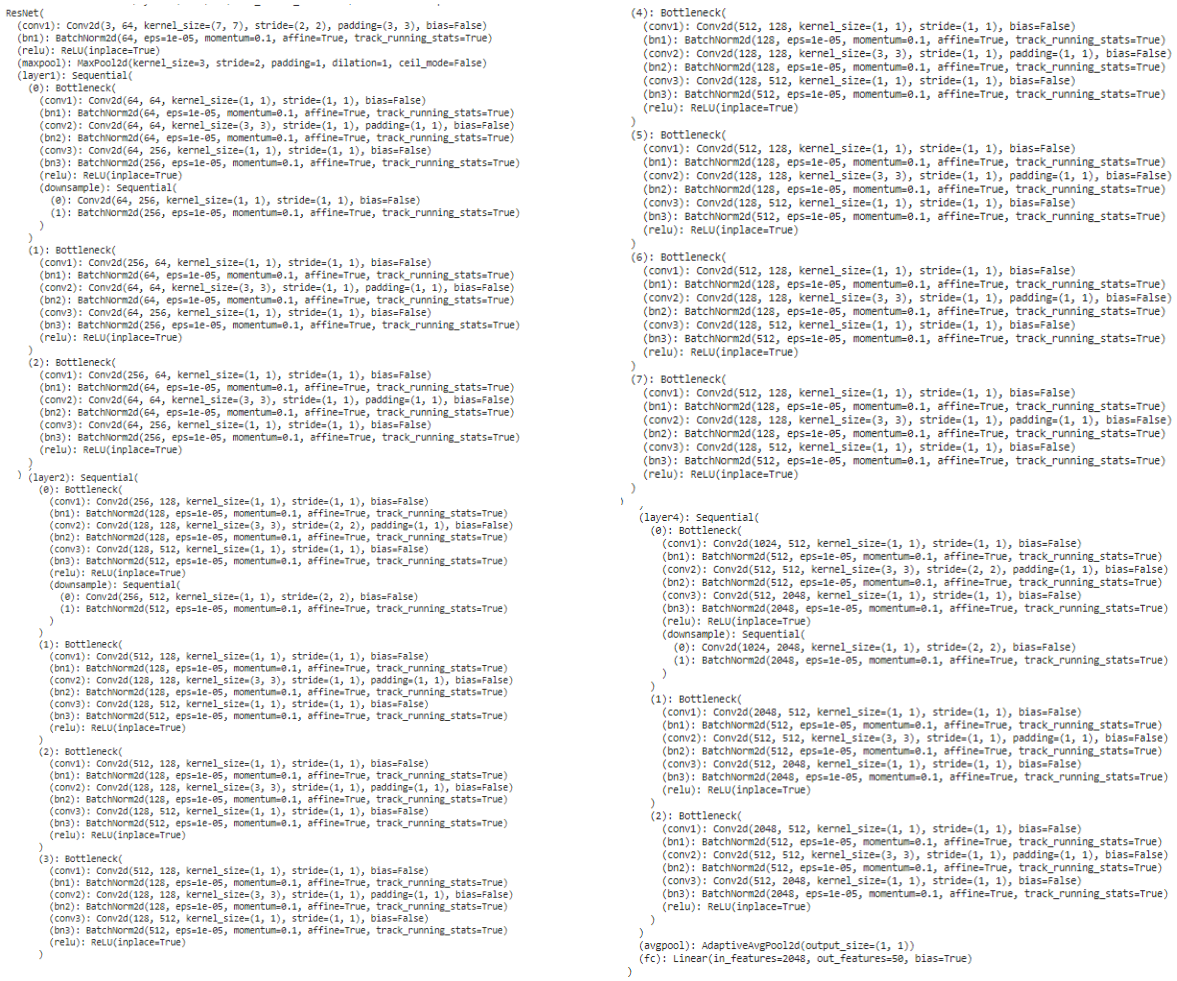


Figure 3: model architecture of resnet152

2. Validation accuracy = 0.8272
3. Visualize result

Choose vgg16 + classification layers. Extract the output of the second last layer, which is *fc*.0. Use *sklearn.manifold.TSNE* to perform TSNE, and transform 256-dim vectors into 2-dim vectors..

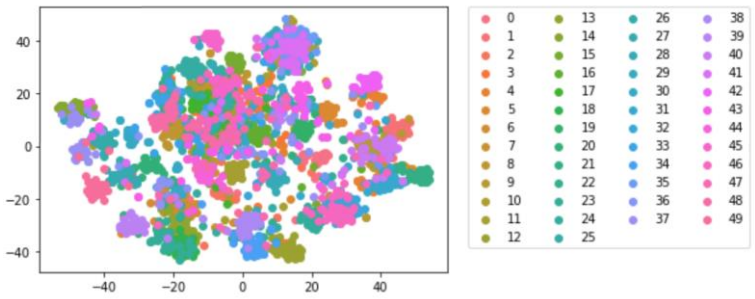
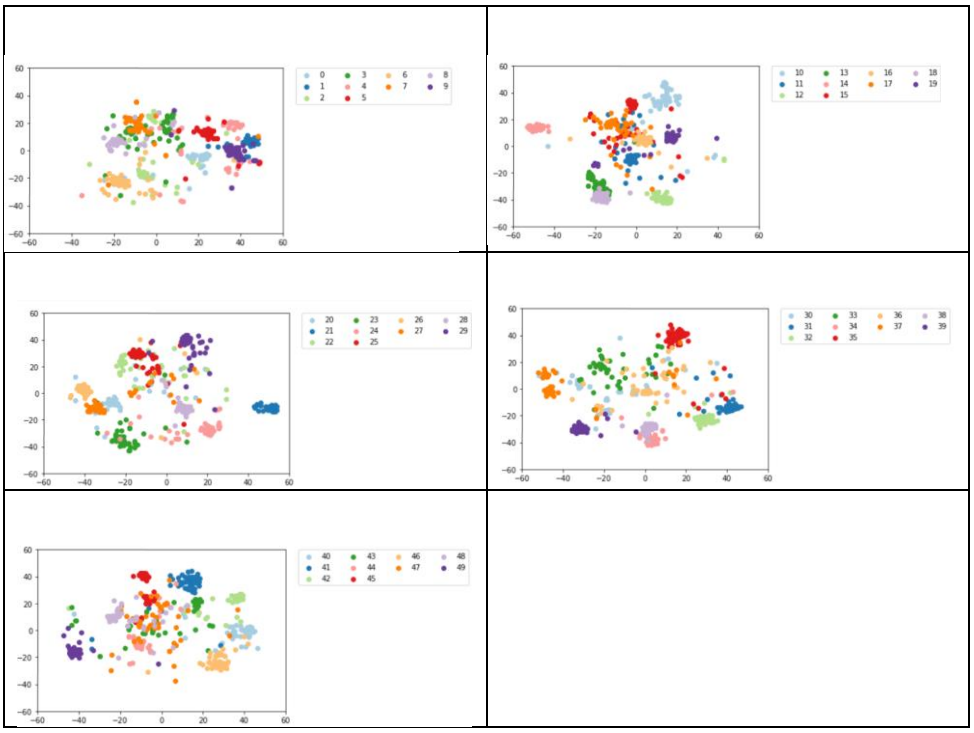


Figure 4: TSNE result

Discussion:

Because it's hard to distinguish 50 colors, I separately plot each 10 class in a plot.



Most class data clustered into one or several clusters, however some classes of data are still are still scattered. For example, class 30, 33, 47 are relatively scattered, which are respectively mushrooms, mouse, and otter. In addition, and different class clusters aren't totally separated. For example, class 18 and 13 overlap each other, which are two kinds of trees. Apparently, these two classes are very similar, so their projected vectors are similar, too.

Problem 2

1. Model Architecture

```
fcn32_1b(
  (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)
  )
  (conv): Sequential(
    (0): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): ReLU(inplace=True)
  )
  (classifier): Sequential(
    (0): ConvTranspose2d(1024, 7, kernel_size=(32, 32), stride=(32, 32))
  )
)
```

Figure 5: model architecture of FCN32s

Add three convolution layers after vgg16 feature extractor, and up sample the feature map by 32 times by a single convolution transpose layer.

2. Improved Model Architecture

Implement FCN8s proposed in [1]. The model architecture is as follow:

```
fcn8s(
  (to_pool3): 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)
  )
  (to_pool4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (to_pool5): Sequential(
    (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (to_pool4_predict): Conv2d(512, 7, kernel_size=(1, 1), stride=(1, 1))
  (to_pool5_predict): Sequential(
    (0): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): Conv2d(1024, 7, kernel_size=(1, 1), stride=(1, 1))
  )
  (pool5_upsample): ConvTranspose2d(7, 7, kernel_size=(2, 2), stride=(2, 2))
  (to_pool3_predict): Conv2d(256, 7, kernel_size=(1, 1), stride=(1, 1))
  (fused_pool4_pool5_upsample): ConvTranspose2d(7, 7, kernel_size=(2, 2), stride=(2, 2))
  (fused_all_upsample): ConvTranspose2d(7, 7, kernel_size=(8, 8), stride=(8, 8))
)
```

Figure 6: model architecture of FCN8s

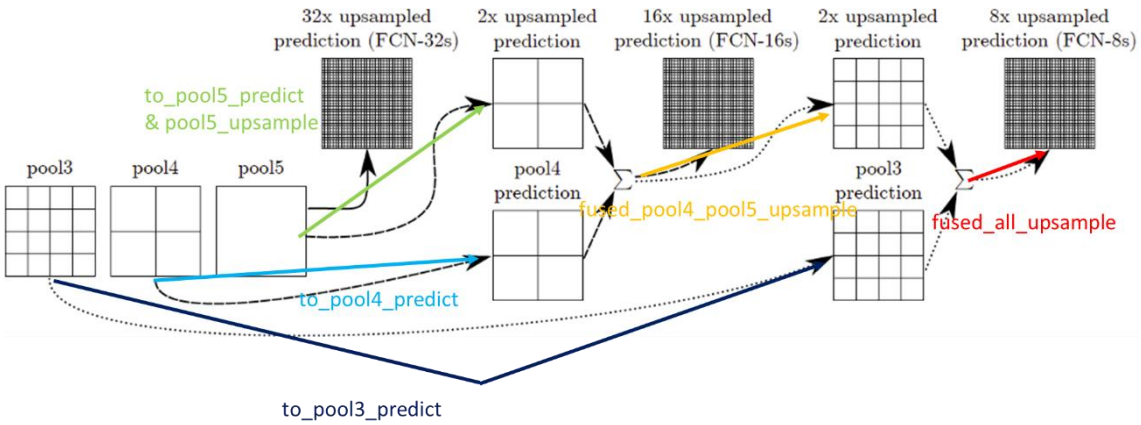
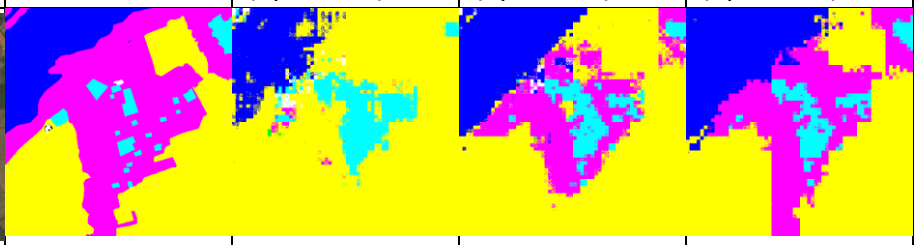
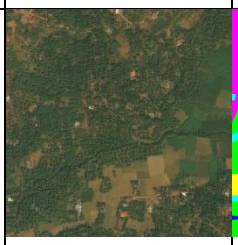
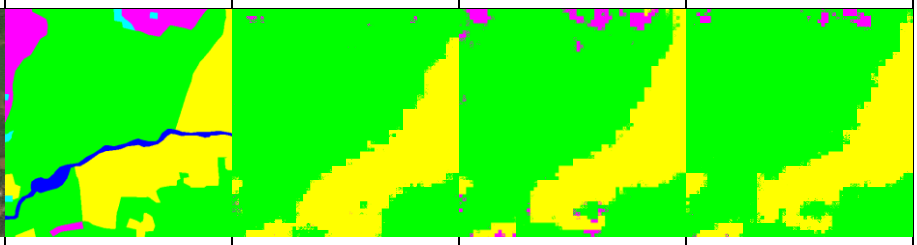
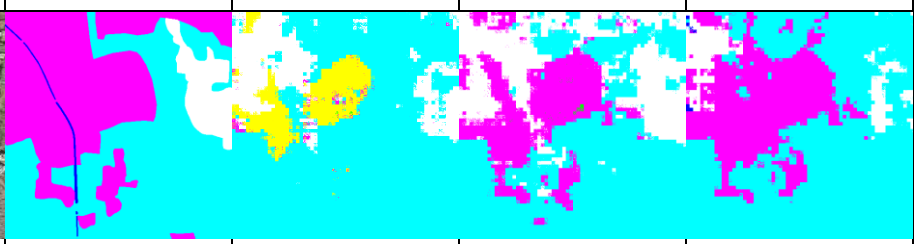


Figure 7: corresponding layers in the original paper [1][2]

3. Validation mIoU = 0.725185
4. Predicted segmentation mask of different stages

	Original image	Ground truth	Early (epoch 15)	Middle (epoch 30)	Final (epoch 55)
0010_sat.jpg					
0097_sat.jpg					
0107_sat.jpg					

Reference

[1] “Fully Convolutional Networks for Semantic Segmentation”, Evan Shelhamer, Jonathan Long, and Trevor Darrell, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 39, NO. 4, APRIL 2017

[2] https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1