Robotic Navigation and Exploration

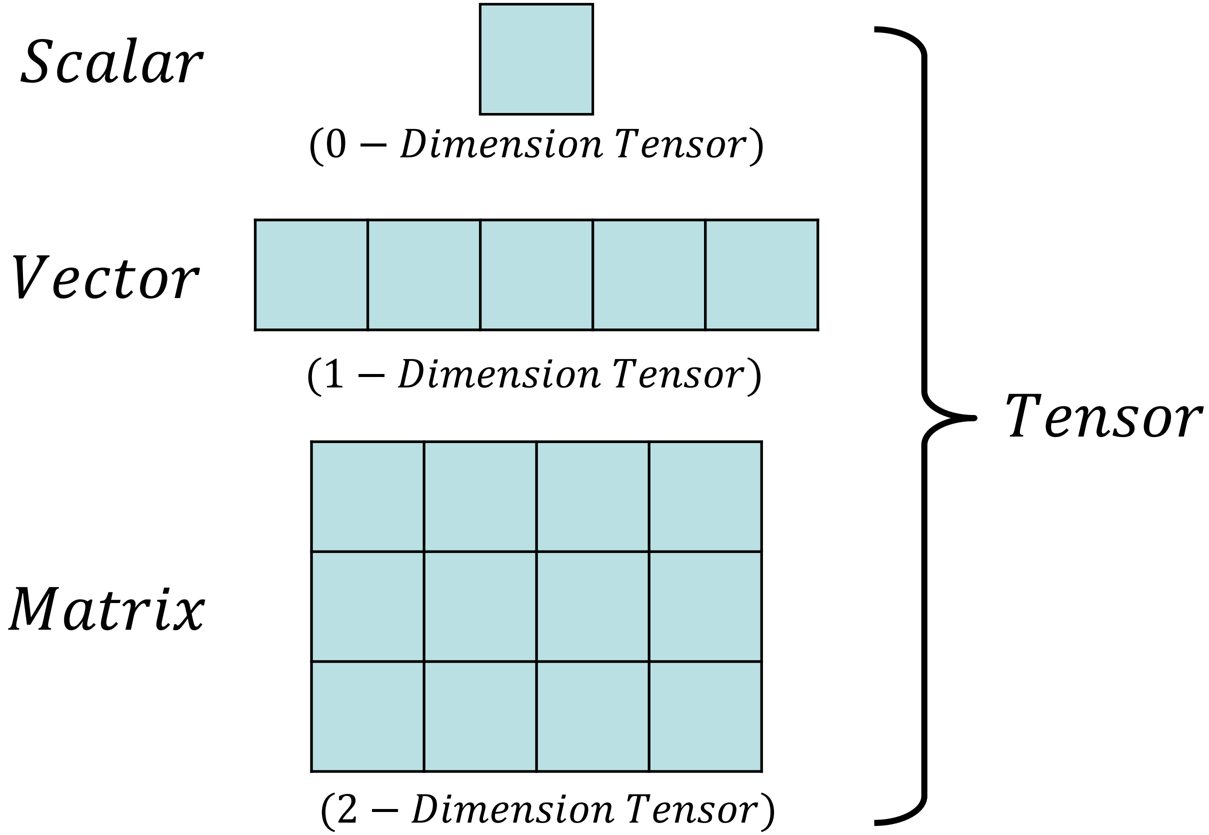
Lab5: Semantic Segmentation with PyTorch

Min-Chun Hu anitahu@cs.nthu.edu.tw
CS, NTHU

PyTorch Basics

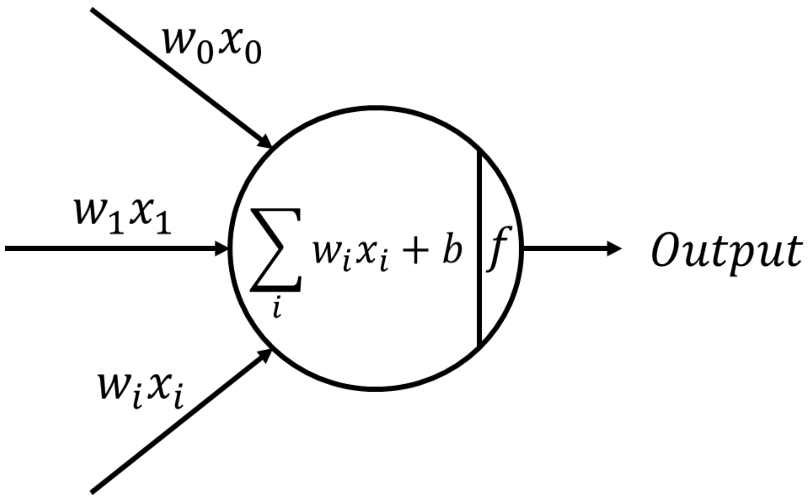
Tensor

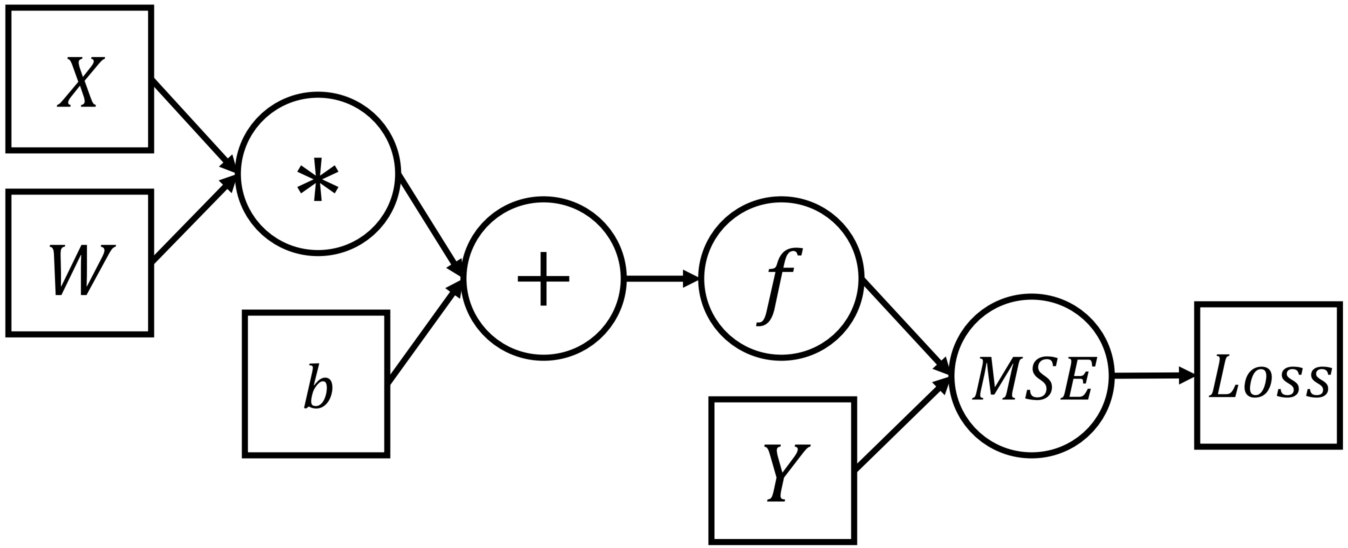
- In Pytorch , tensor is the basic element which can be represented as multi-dimensional array.



Computation Graph

- The core of modern deep learning library such as **Tensorflow** and **PyTorch** is the **Automatic Differentiator on Computation Graph**.
- Every neural network model can be represented as a graph. By passing the error message through the graph, the library can get the updating direction of each variables.





Computation Graph

- Linear Regression

$$[y_1 \quad y_2 \quad y_3 \quad y_4] = [w \quad b]\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

```
import torch
x = torch.tensor([[0,2,3,4],[1,1,1,1]], dtype=float)
y = torch.tensor([[1,3,5,8]], dtype=float)
w = torch.tensor([[0.1,0.1]], dtype=float, requires_grad=True)

for i in range(20):
    # Forward
    y_out = torch.matmul(w,x)
    loss = (0.5 * (y - y_out)**2).sum()
    print(loss)

    # Backward
    loss.backward()
    with torch.no_grad():
        w -= 0.01 * w.grad
    w.grad.zero_()
```

Create the gradient buffer.

Every operation dynamically create the graph.

Backward operation will delete the graph.

Update parameters, prevent to building graph.

Notification

- The backward node should be single value, otherwise you need to give the weighting parameter with same size of the backward node.
  - Ex.
    ```
    loss = 0.5*(y-y_out)**2
    loss.backward(torch.tensor([[1,1,1,1]], dtype=float))
    ```
- Backward operation will delete the graph by default, you can set the backward parameter to retain the graph.
  - Ex. `loss.backward(retain_graph=True)`
- After backward operation, each leaf node that requires gradient will "add" the gradient at the buffer, thus you need to set the gradient buffer to zero in general usage before backward. (However, you can also use this feature to achieve special effects.)

Commonly Used Libraries

- torch.nn: Neural Network Operation.
- torch.nn.functional: Functional Operation.
- torch.optim: Optimizer.
- torch.utils.data: Dataset / Data Loader.
- torchvision: Comoputer vision-related operation/datasets/models.

PyTorch Documentation: https://pytorch.org/docs/stable/index.html

PyTorch LeNet Example

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

class LeNet(nn.Module):
    def __init__(self):
        super().__init__()
        # Create Parameters Here
        self.conv1 = nn.Conv2d(1,6,kernel_size=5)
        self.conv2 = nn.Conv2d(6,16,kernel_size=5)
        self.fc1 = nn.Linear(16*5*5,128)
        self.fc2 = nn.Linear(128,10)

    def forward(self, x):
        # Construct Forward Path
        output = F.relu(self.conv1(x)) # (1,32,32) -> (6,28,28)
        output = F.max_pool2d(output) # (6,28,28) -> (6,14,14)
        output = F.relu(self.conv2(output)) # (6,14,14) -> (16,10,10)
        output = F.max_pool2d(output) # (16,10,10) -> (16,5,5)
        output = F.relu(self.fc1(output.view(-1,16*5*5))) # (16*5*5) -> (128)
        output = self.fc2(output) # (128) -> (10)
        return output
```

PyTorch LeNet Example

```
net = LeNet() # Create Network
criterion = nn.BCEWithLogitsLoss() # Loss Function
optimizer = optim.Adam(net.parameters(), lr=0.001) # Create Optimizer

for i in range(1000):
    optimizer.zero_grad() # Clear the gradient buffer
    inputs, labels = # Get data batch
    outputs = net(inputs) # Forward Propagation
    loss = criterion(outputs, labels) # Compute Loss
    loss.backward() # Backward Propagation
    optimizer.step() # Update Parameters
```

Module Block

- Inheritat “nn.Module” to define new computation block to make sure the parameters can be found recursively.

```python
class SubBlock(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(16,2,kernel_size=3)
        self.conv2 = nn.Conv2d(2,1,kernel_size=3)

    def forward(self, x):
        output = self.conv1(x)
        output = self.conv2(output)
        return output
```

```python
class MainNet(nn.Module):
    def __init__(self):
        super().__init__()
        # Create Parameter Here
        self.conv1 = nn.Conv2d(3,8,kernel_size=3)
        self.conv2 = nn.Conv2d(8,16,kernel_size=3)
        self.subblock = SubBlock()

    def forward(self, x):
        # Construct Forward Path
        output = self.conv1(output)
        output = F.relu(output)
        output = self.conv2(output)
        output = self.subblock(output)
        output = F.sigmoid(output)
        return output
```

Dataset and Dataloader

- `torch.utils.data.Dataset` is an abstract class representing a dataset. Your custom dataset should inherit Dataset and override the following methods:
  - `__len__` : returns the size of the dataset.
  - `__getitem__` : support the indexing that can be used to get ith sample.

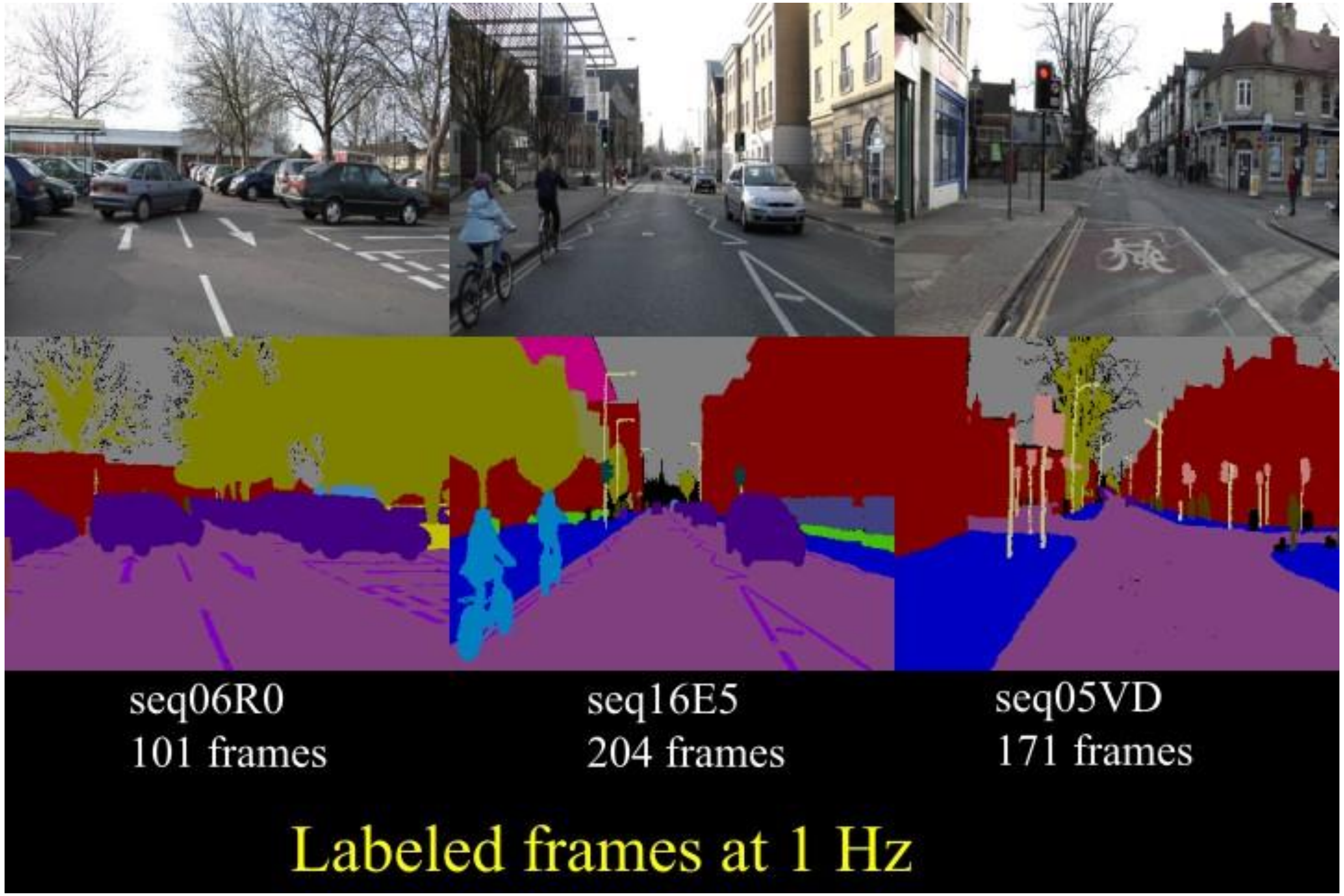
- `torch.utils.data.DataLoader` shuffle the data in the dataset and construct an iterator for us to get a random batch of training data.

- Ex.

```
train_data = CustomDataset(…)
train_loader = DataLoader(train_data, batch_size=32, shuffle=True)

for iter, batch in enumerate(train_loader):
    inputs = torch.FloatTensor(batch['X'])
    labels = torch.FloatTensor(batch['Y'])
```

Semantic Segmentation

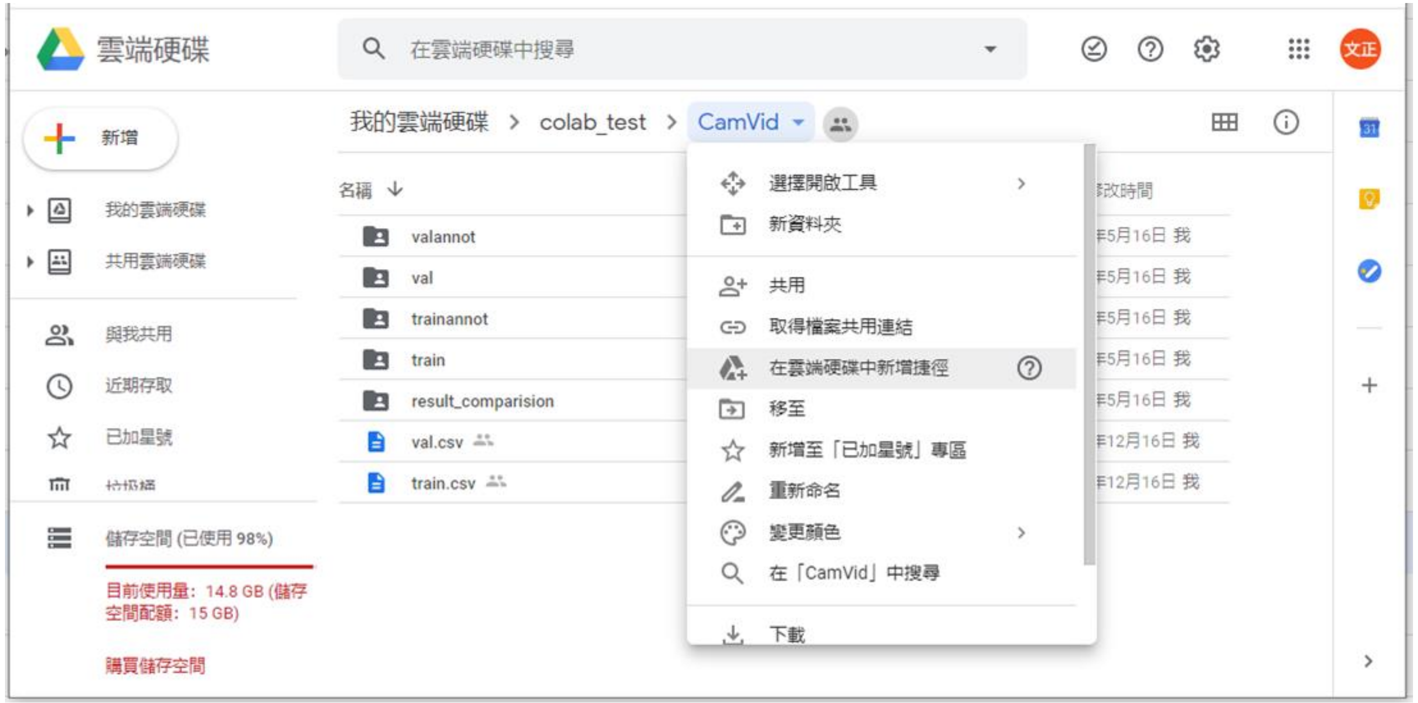
CamVid Dataset



Prepare Dataset

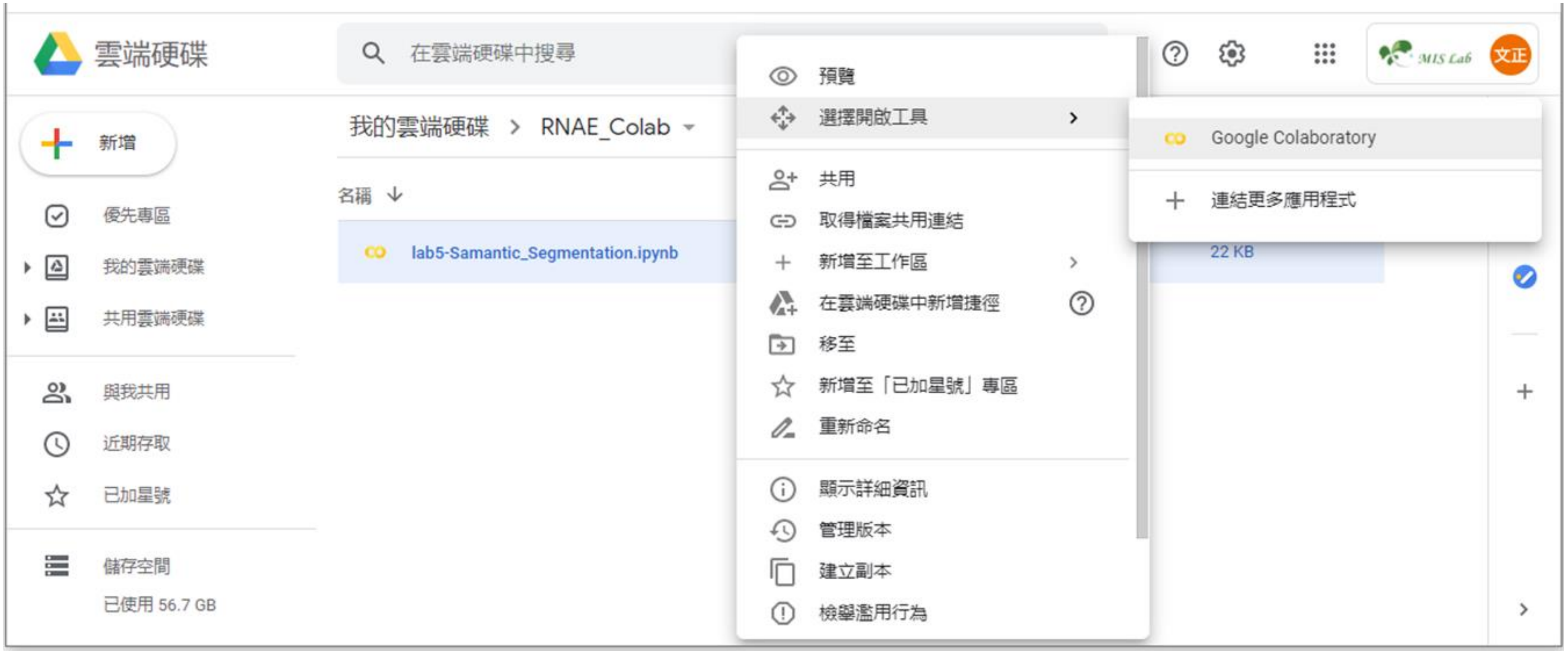
- Link: https://drive.google.com/drive/folders/1K8kys7gWfl-As3lCz6O4Y6ETdn_OVTd_?usp=sharing

- Login your google account and add the shortcut to the dataset folder to your cloud drive.



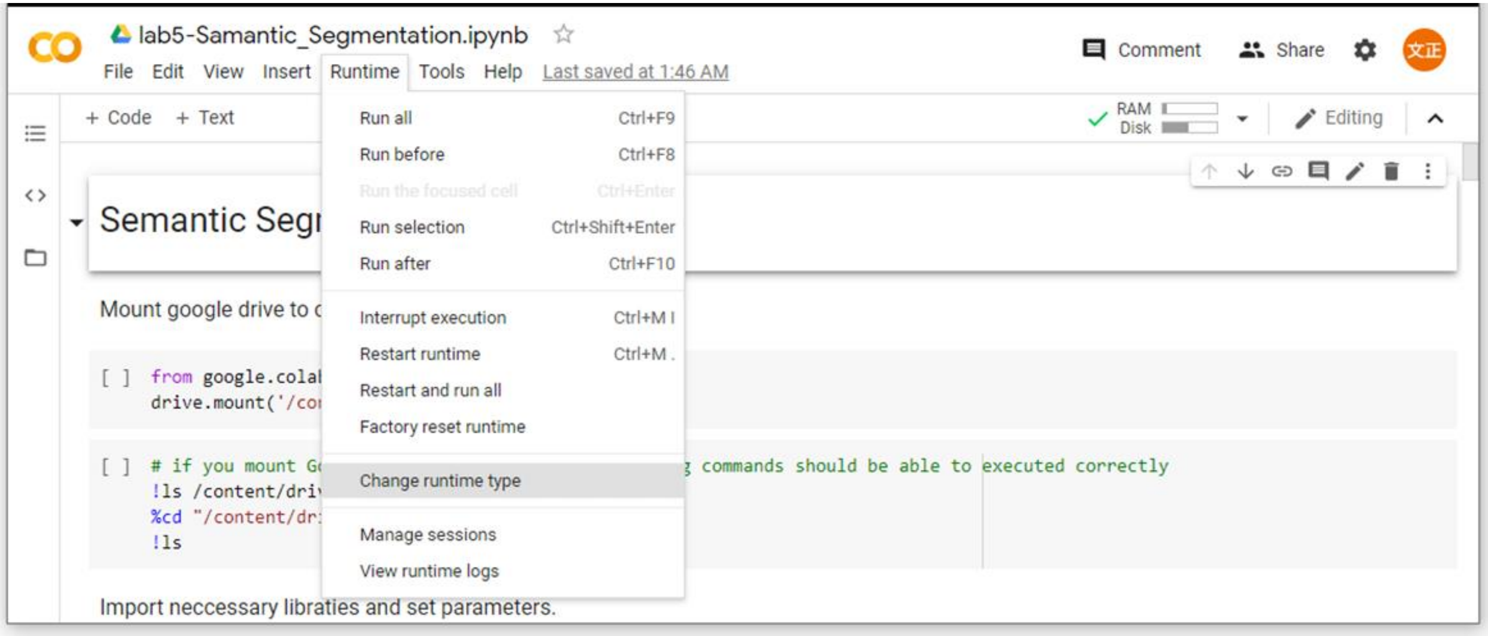
Google Colaboratory

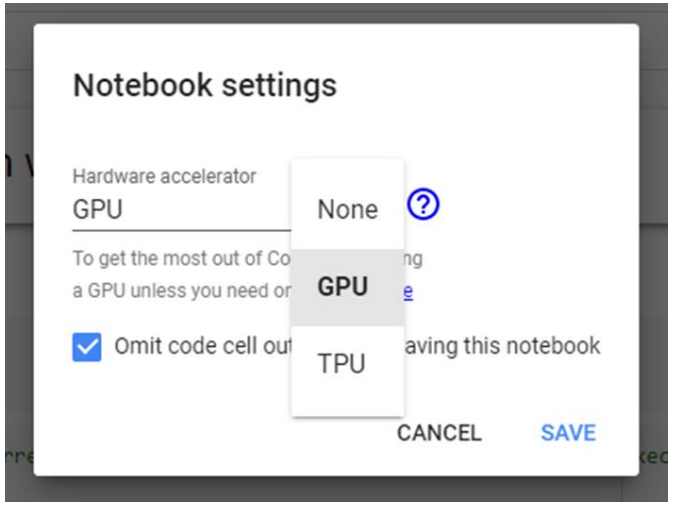
- Upload the ipynb file to your google drive, and open it using Google Colaboratory. (If you donot have Google Colaboratary, you need to click “連結更多應用程式” and install it.)



Google Colaboratory

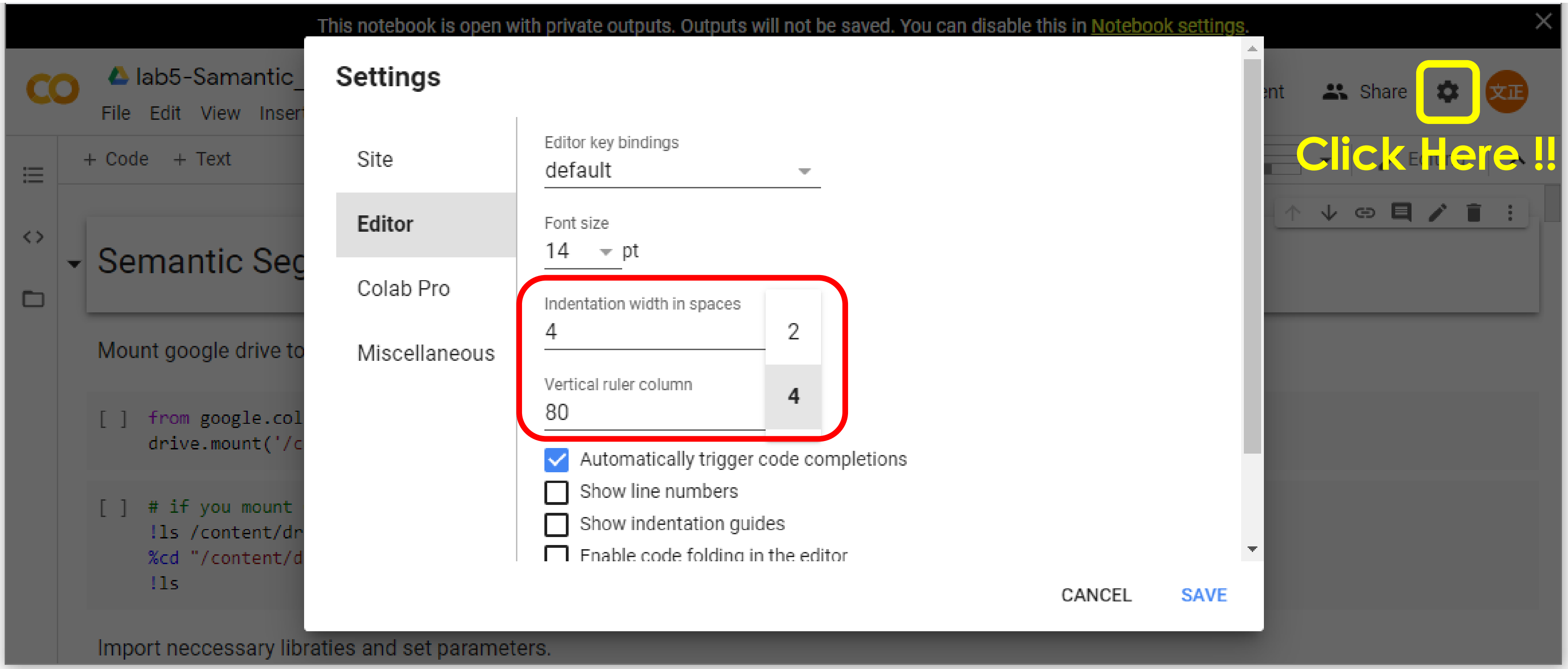
- Set runtime hardware accelerator to “GPU”





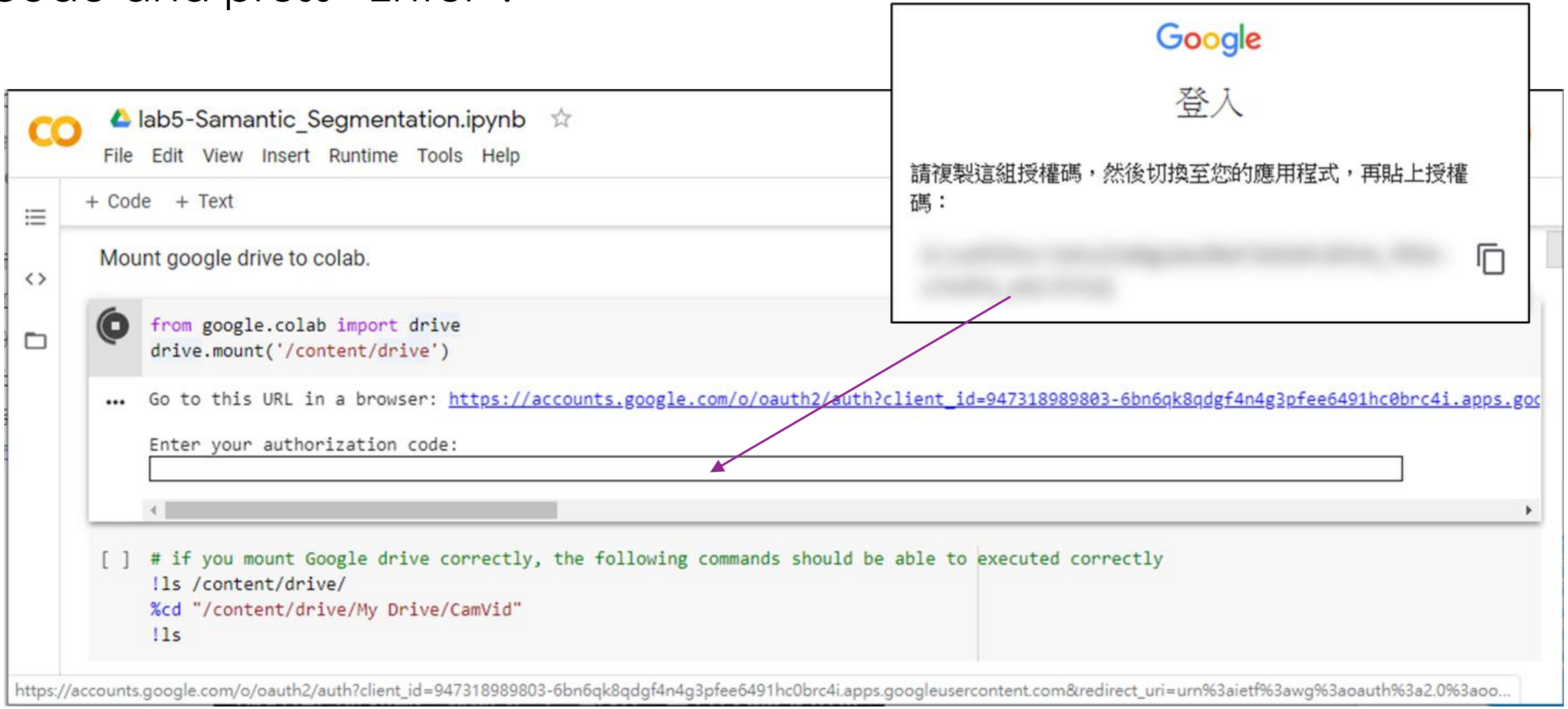
Google Colaboratory

- Set the indentation size to 4.



Mount Google Drive

- Run the first cell and go to the URL on the message. Copy the authorization code and press “Enter”.



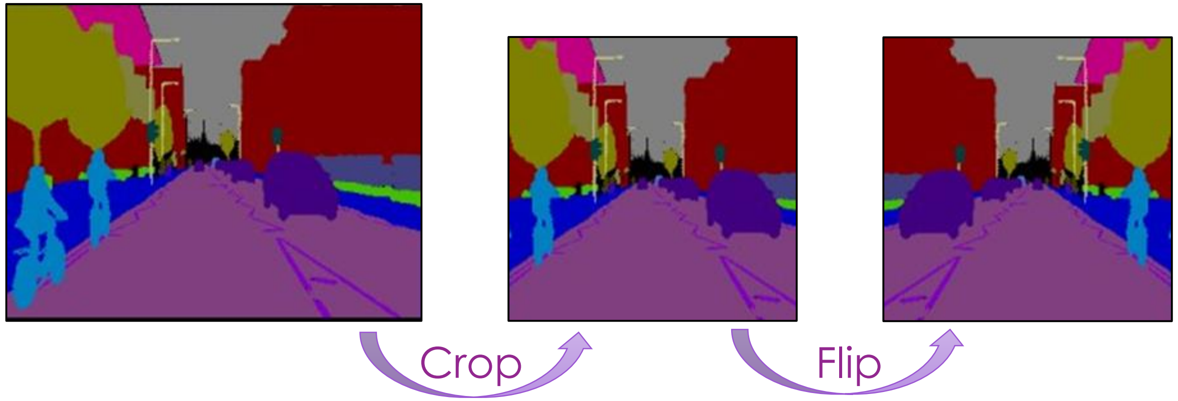
Mount Google Drive

- Run the second cell to check the dataset file.

```
# if you mount Google drive correctly, the following commands should be able to executed correctly
!ls /content/drive/
%cd "/content/drive/My Drive/CamVid"
!ls
```

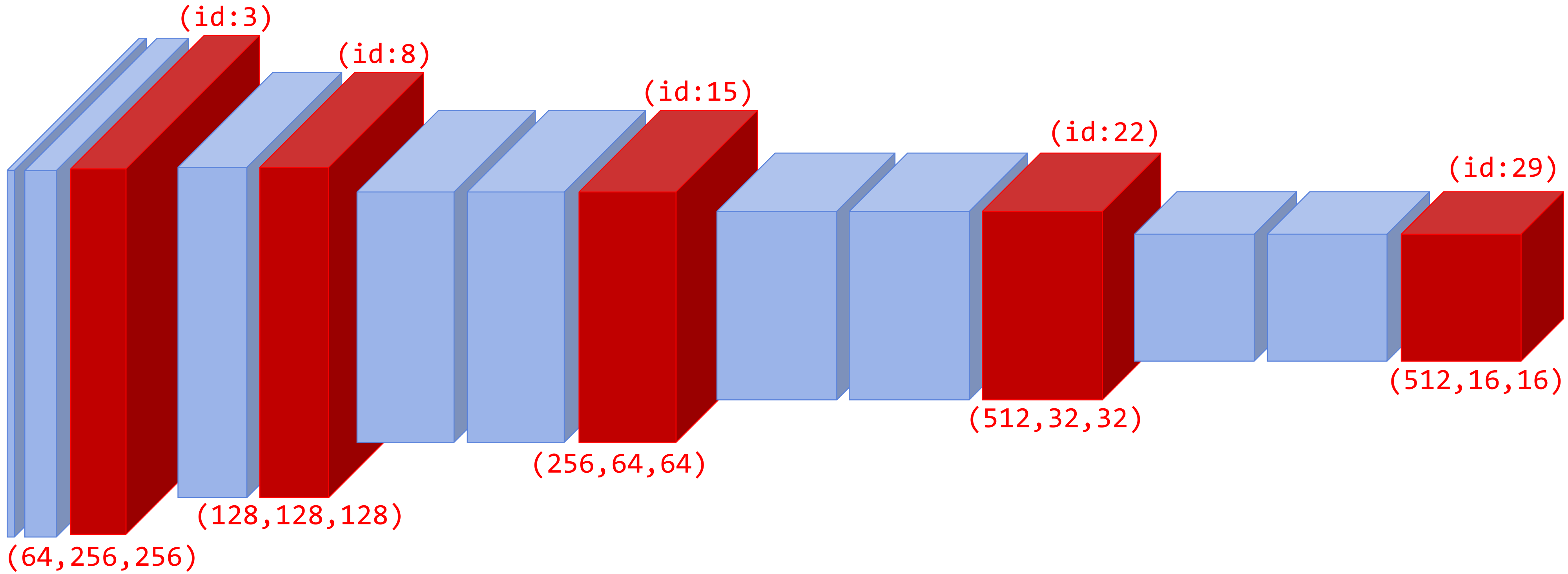
```
'My Drive'  'Shared drives'
/content/drive/.shortcut-targets-by-id/1K8kys7gWfI-As3lCz6O4Y6ETdn_OVTd_/CamVid
result_comparision  train  trainannot  train.csv  val  valannot  val.csv
```

Data Augmentation




```python
# crop images and labels
w, h = img.size
if self.rand_crop:
    A_x_offset = np.int32(np.random.randint(0, w - self.new_w + 1, 1))[0]
    A_y_offset = np.int32(np.random.randint(0, h - self.new_h + 1, 1))[0]
else:
    A_x_offset = int((w - self.new_w)/2)
    A_y_offset = int((h - self.new_h)/2)

img = img.crop((A_x_offset, A_y_offset, A_x_offset + self.new_w, A_y_offset + self.new_h))
label_image = label_image.crop((A_x_offset, A_y_offset, A_x_offset \
                                  + self.new_w, A_y_offset + self.new_h))

# flip images and labels
img = np.transpose(img, (2, 0, 1)) / 255.
label = np.asarray(label_image)
if np.random.sample() < self.flip_rate:
    img = np.fliplr(img)
    label = np.fliplr(label)
```

VGG-16 Feature Extractor



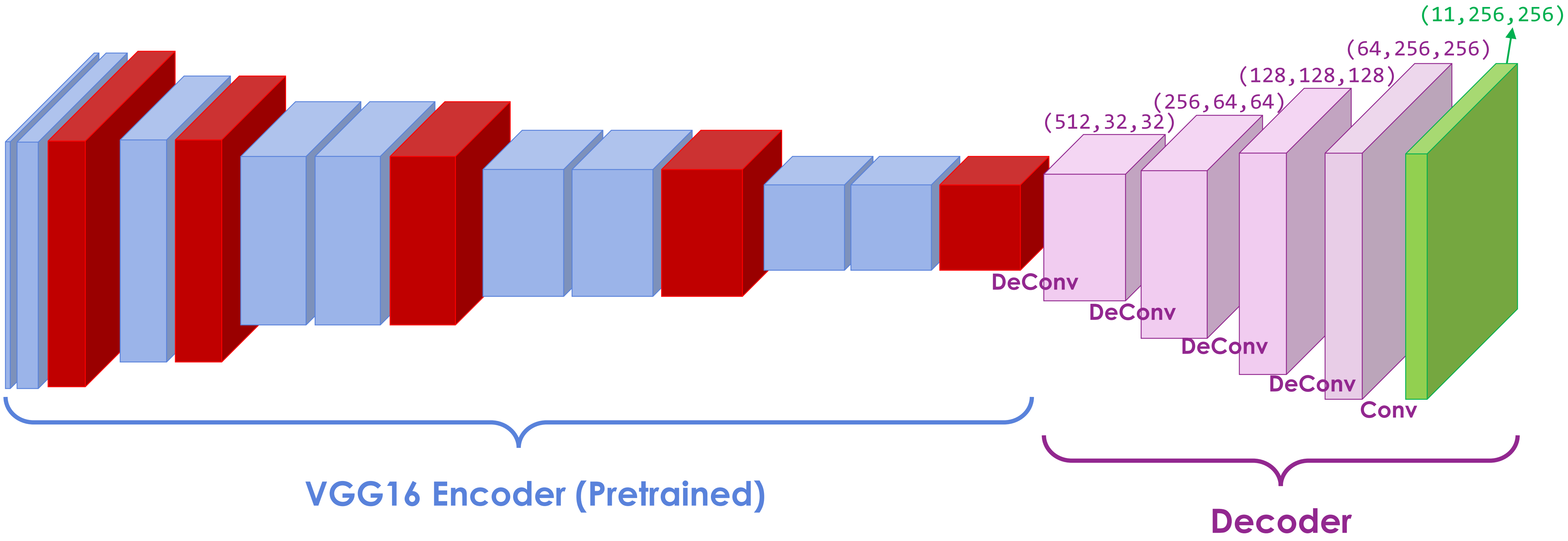
VGG-16 Feature Extractor

```python
class Vgg16(nn.Module):
    def __init__(self, pretrained = True):
        super(Vgg16, self).__init__()
        self.vggnet = models.vgg16(pretrained)
        del(self.vggnet.classifier) # Remove fully connected layer to save memory.
        features = list(self.vggnet.features)
        self.layers = nn.ModuleList(features).eval()

    def forward(self, x):
        results = []
        for ii,model in enumerate(self.layers):
            x = model(x)
            if ii in [3,8,15,22,29]:
                results.append(x) #(64,256,256),(128,128,128),(256,64,64),(512,32,32),(512,16,16)
        return results

vgg_model = Vgg16()
vgg_model = vgg_model.cuda()
print(vgg_model.layers)
```

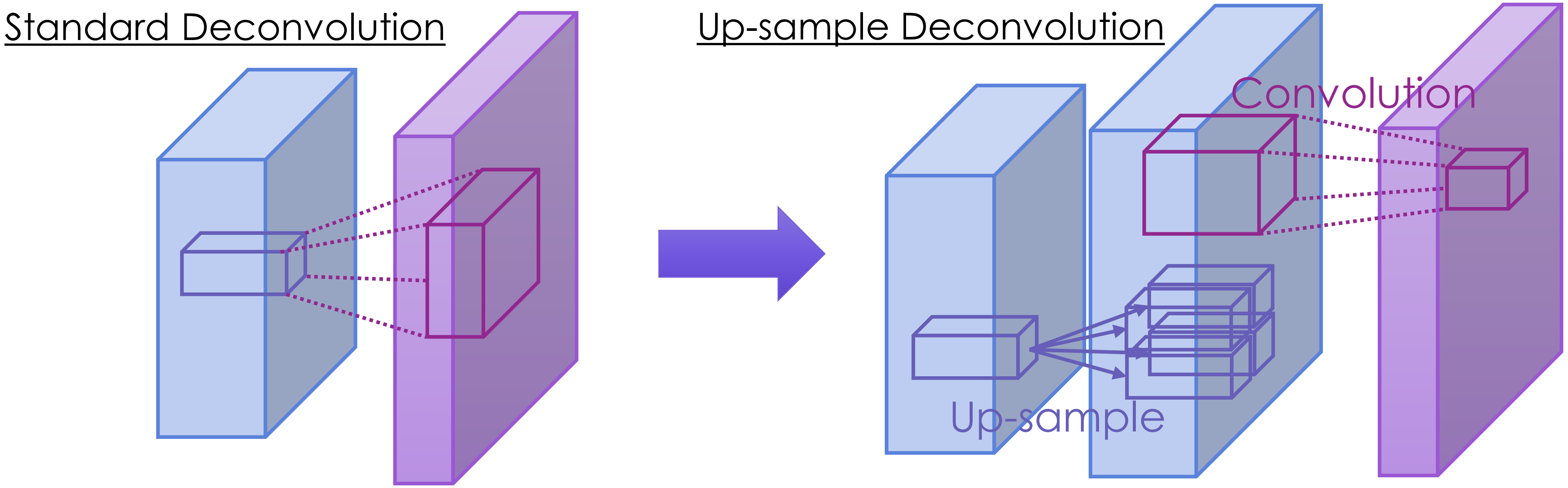
VGG-16 Feature Extractor

```
ModuleList(
  (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)
)
```

Encoder-Decoder Architecture

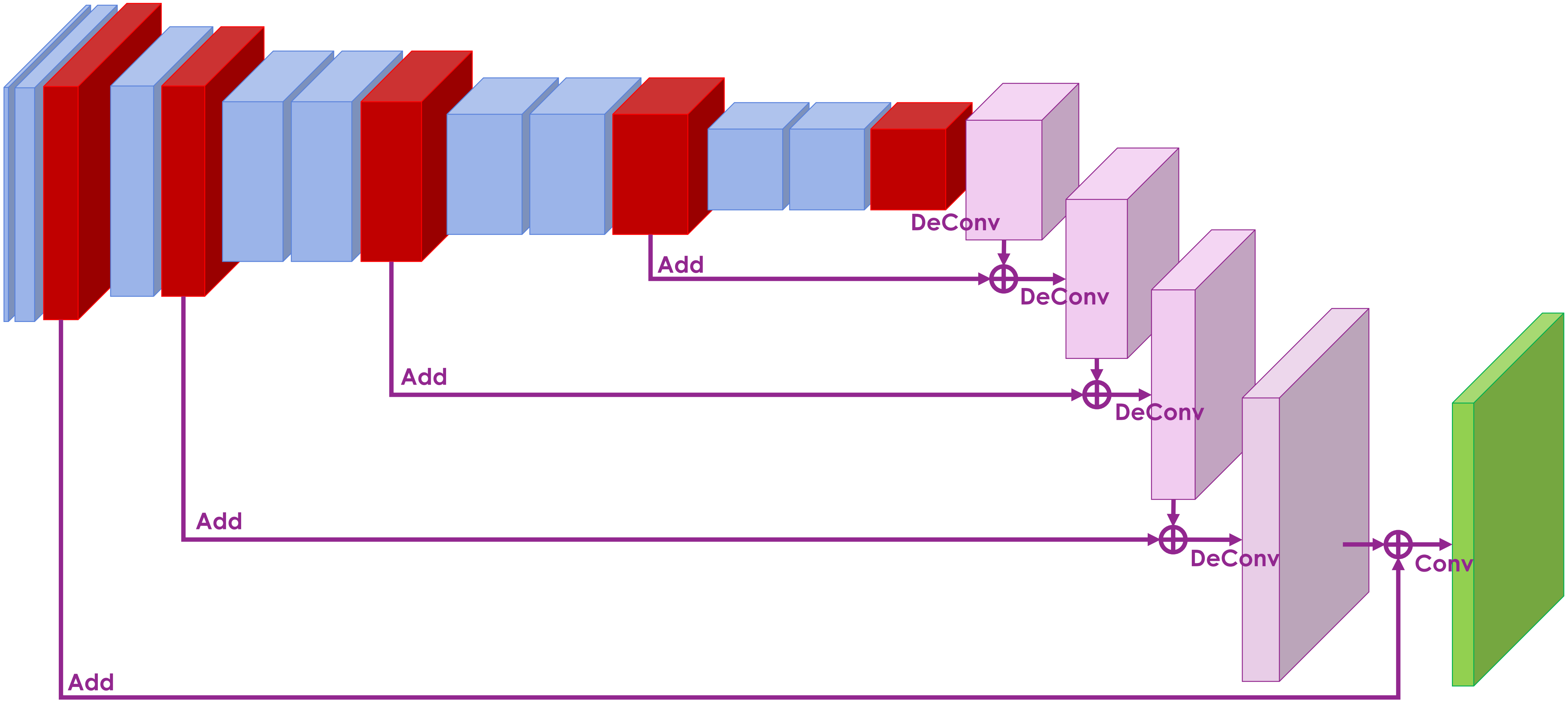


Up-sample Deconvolution




```python
class DeConv2d(nn.Module):
    def __init__(self, in_channel, out_channel, kernel_size, stride, padding, dilation):
        super().__init__()
        self.up = nn.Upsample(scale_factor=2, mode='nearest')
        self.conv = nn.Conv2d(in_channel, out_channel, kernel_size=kernel_size, \
                              stride=stride, padding=padding, dilation=dilation)

    def forward(self, x):
        output = self.up(x)
        output = self.conv(output)
        return output
```

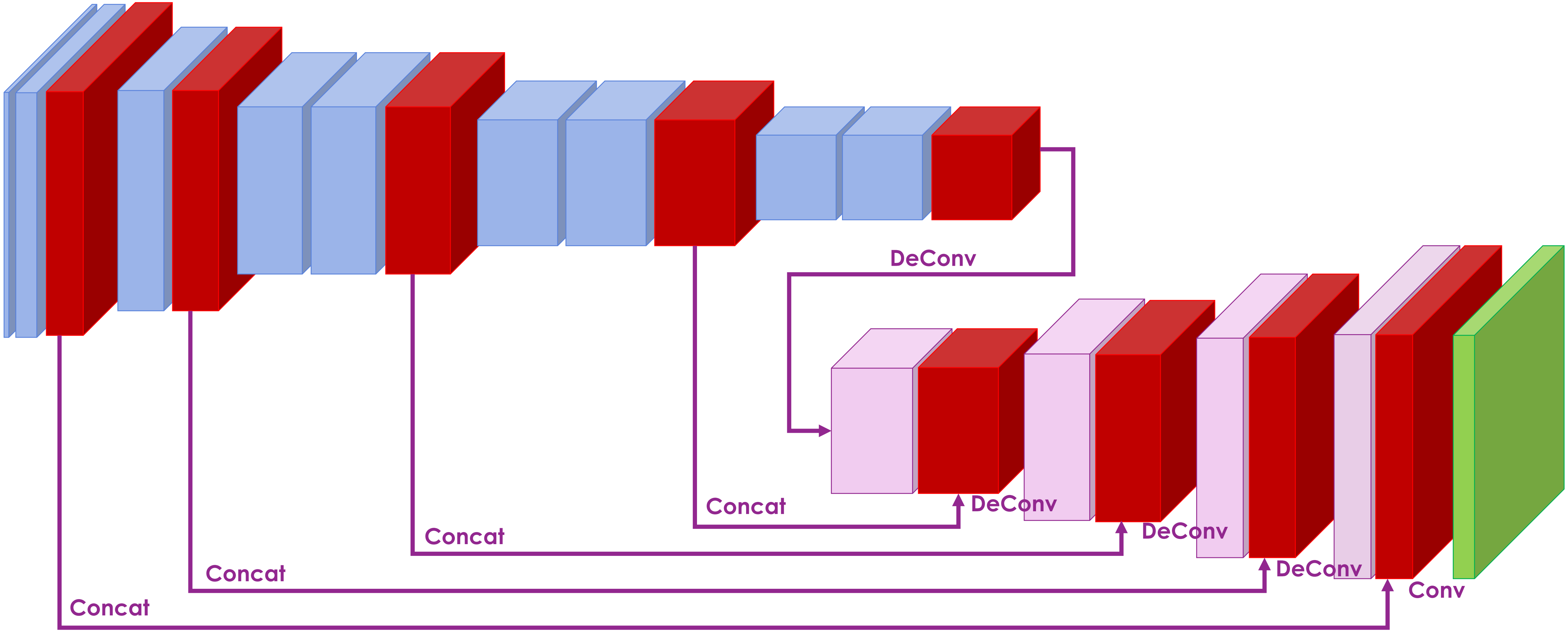
Practice - FCN



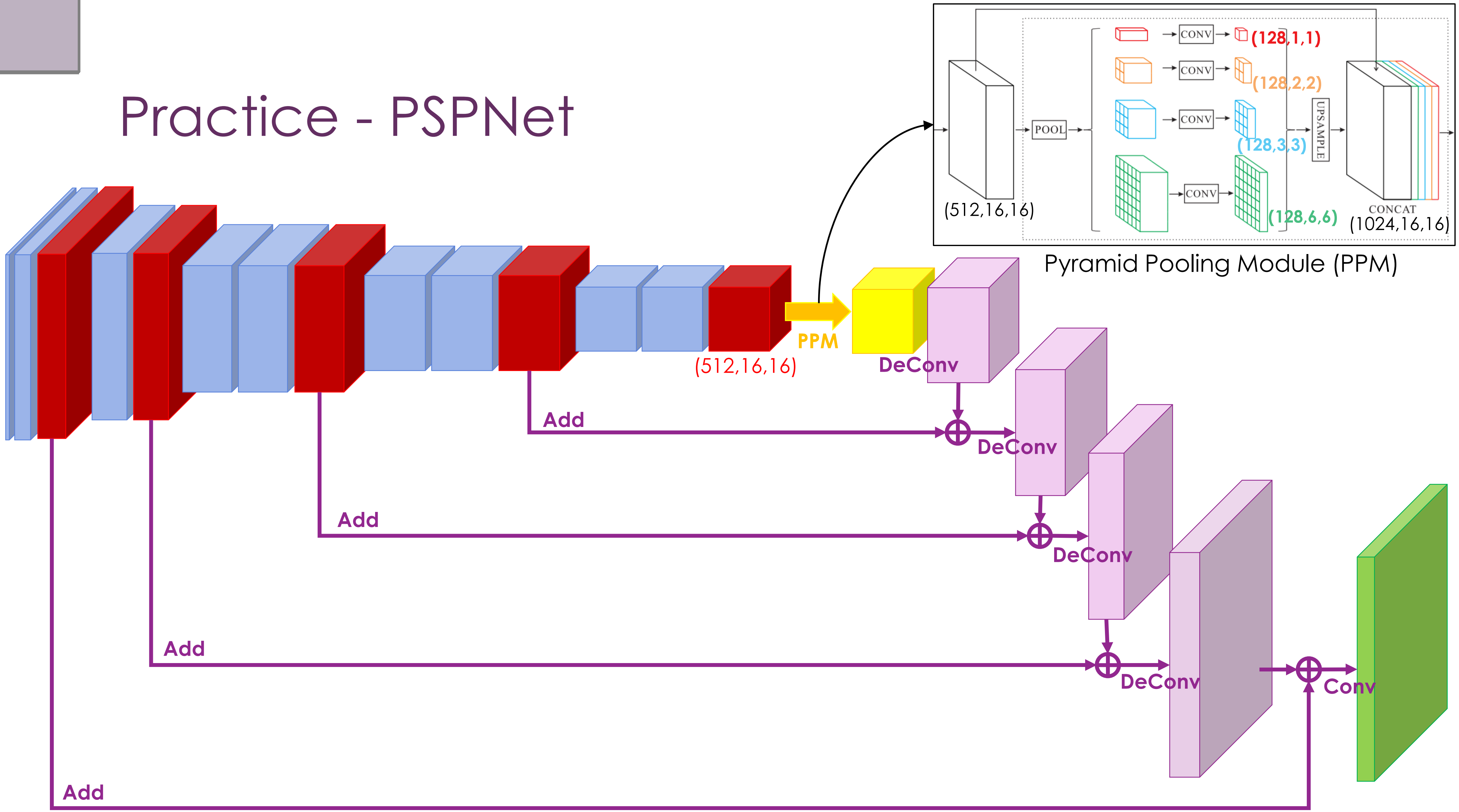
Practice - U-Net

[Hint] Tensor Concatenation

```
tc = torch.cat([t1,t2,…],dim)
```

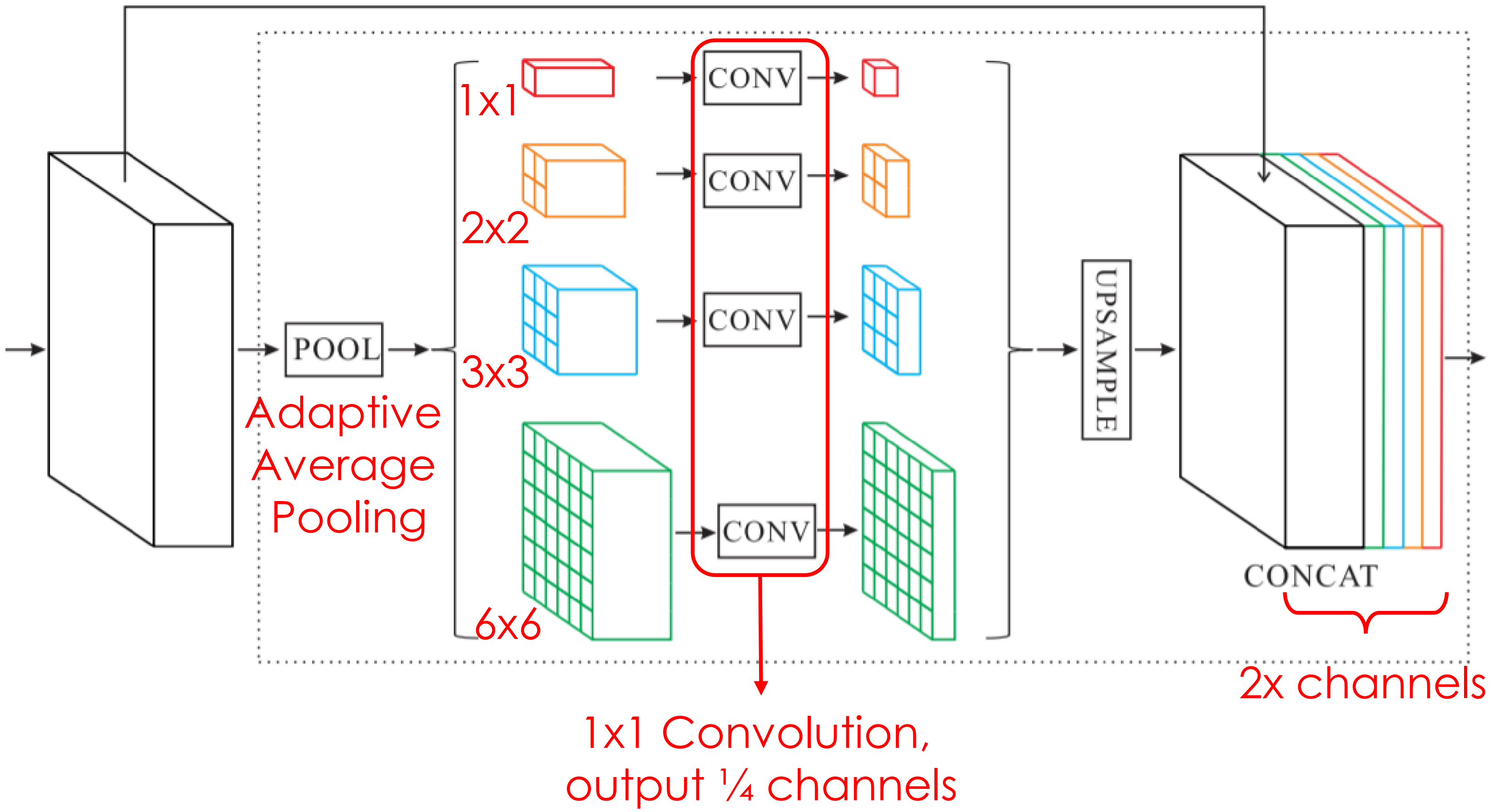


Practice - PSPNet



Pyramid Pooling Module (PPM)

Pyramid Pooling Module



Pyramid Pooling Module

```
# Pyramid Pooling Moudule
self.ppm_size = (16,16)
self.ppm_channel = 512
self.ppm_psize = [1,2,3,6]

self.ppm_pool, self.ppm_conv, self.ppm_up = [], [], []
for psize in self.ppm_psize:
    self.ppm_pool.append(nn.AdaptiveAvgPool2d((psize,psize)))
    self.ppm_conv.append(nn.Conv2d(int(self.ppm_channel), int(self.ppm_channel/len(self.ppm_psize)), kernel_size=1))
    self.ppm_up.append(nn.Upsample(size=self.ppm_size, mode='bilinear', align_corners=True))

self.ppm_pool = nn.ModuleList(self.ppm_pool)
self.ppm_conv = nn.ModuleList(self.ppm_conv)
self.ppm_up = nn.ModuleList(self.ppm_up)
```

```
# Forward
ppm_list = [pre_output[4]]
for i in range(len(self.ppm_psize)):
    output = self.ppm_pool[i](pre_output[4])
    output = self.ppm_conv[i](output)
    output = self.ppm_up[i](self.relu(output))
    ppm_list.append(output)
output = torch.cat(ppm_list,1)
```

Select Your Model

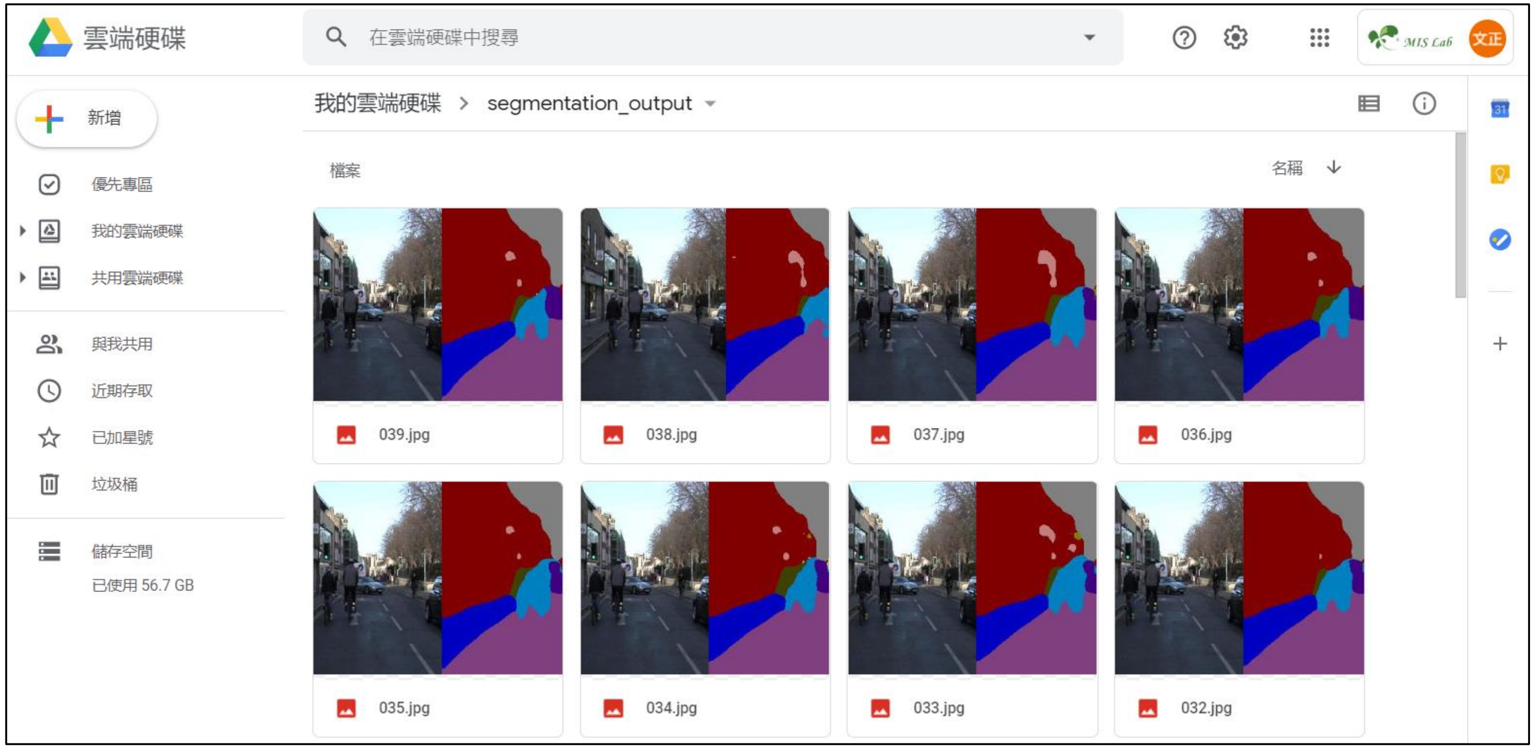
```
[ ] seg_model = EncoderDecoder(pretrained_net=vgg_model, n_class=num_class)
    #seg_model = FCN(pretrained_net=vgg_model, n_class=num_class)
    #seg_model = UNet(pretrained_net=vgg_model, n_class=num_class)
    #seg_model = PSPNet(pretrained_net=vgg_model, n_class=num_class)

    seg_model = seg_model.cuda()
    criterion = nn.BCEWithLogitsLoss()
    optimizer = optim.Adam(seg_model.parameters(), lr=lr)
```

Training and Validation

```
# perform training and validation
train()
```

```
epoch: 0, iter: 0, loss: 0.6840
epoch: 0, iter:10, loss: 0.5426
epoch: 0, iter:20, loss: 0.4711
Finish epoch: 0, time elapsed: 73.9815
pix_acc: 0.6223, meanIoU: 0.2010
========================================
epoch: 1, iter: 0, loss: 0.5281
epoch: 1, iter:10, loss: 0.1586
epoch: 1, iter:20, loss: 0.1838
Finish epoch: 1, time elapsed: 21.9091
pix_acc: 0.6395, meanIoU: 0.1937
========================================
epoch: 2, iter: 0, loss: 0.1554
epoch: 2, iter:10, loss: 0.1228
epoch: 2, iter:20, loss: 0.1209
Finish epoch: 2, time elapsed: 22.0490
pix_acc: 0.7600, meanIoU: 0.3024
========================================
epoch: 3, iter: 0, loss: 0.1339
epoch: 3, iter:10, loss: 0.1036
epoch: 3, iter:20, loss: 0.1152
Finish epoch: 3, time elapsed: 21.8800
pix_acc: 0.7946, meanIoU: 0.3423
========================================
epoch: 4, iter: 0, loss: 0.1201
```

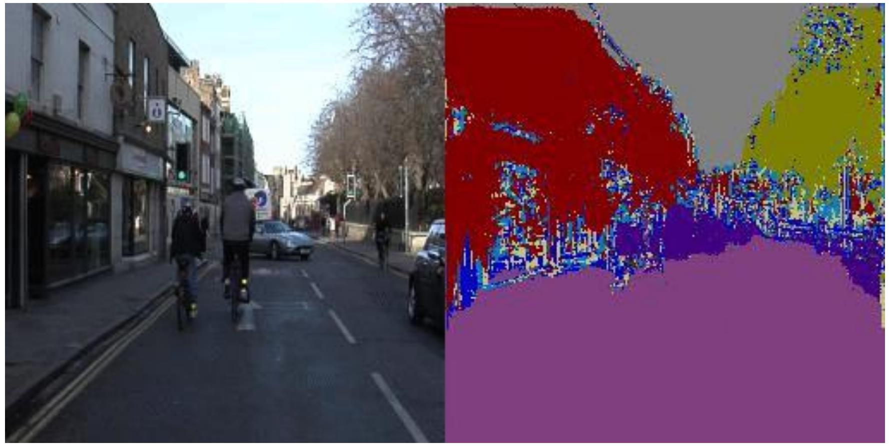
Output Images:



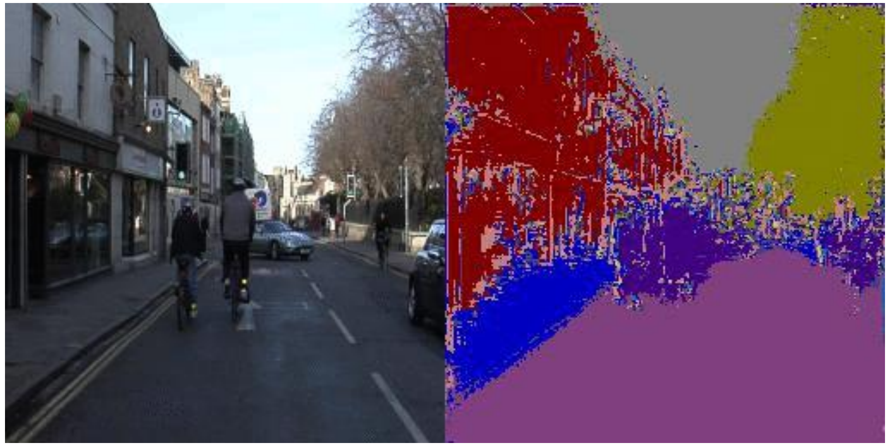
Experimental Results

Encoder-Decoder

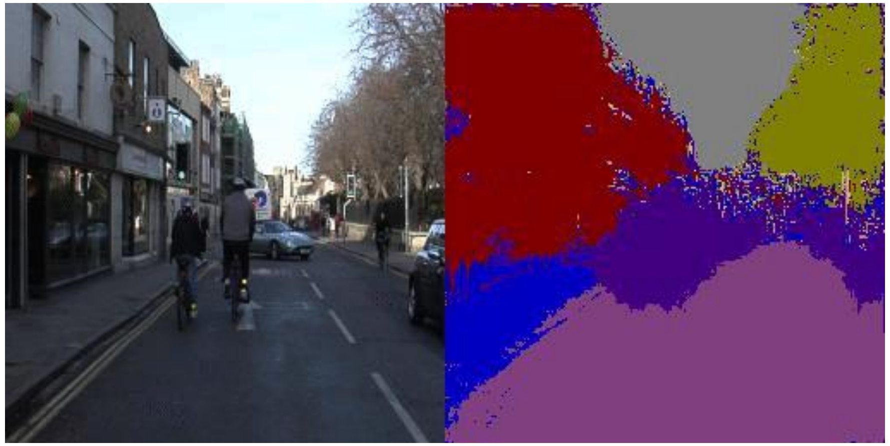
U-Net

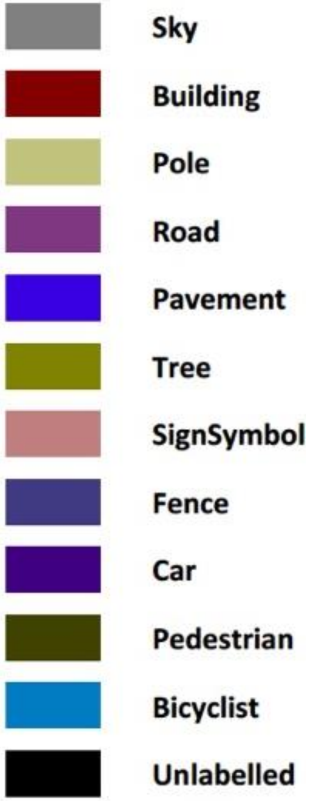


FCN



PSPNet





Q&A