Robotic Navigation and Exploration

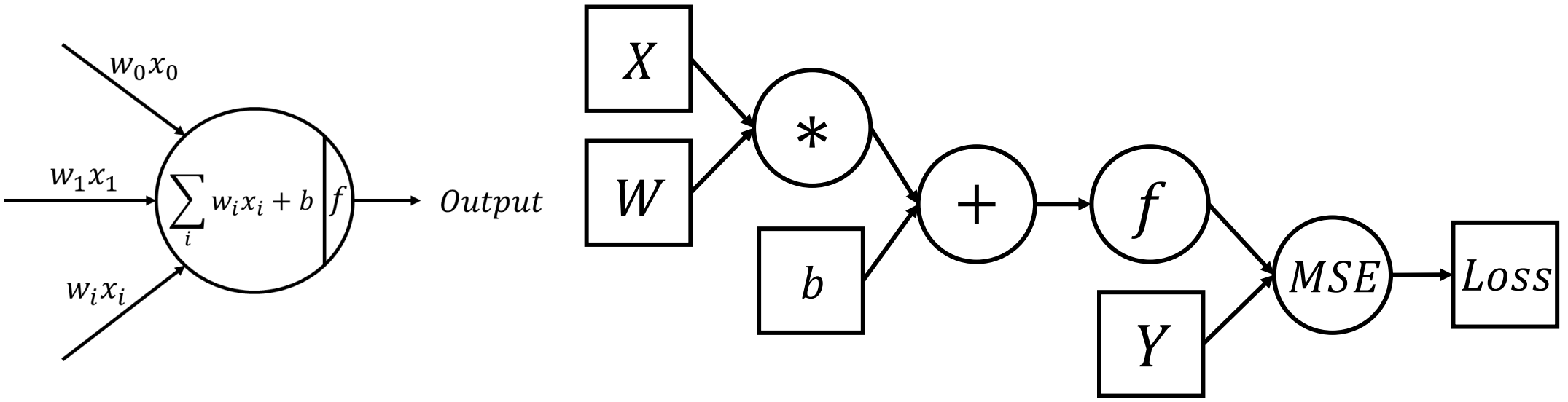
HW3: Semantic Segmentation with PyTorch

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

PyTorch Basics

Computation Graph

- The core of modern deep learning library such as **tensorflow** and **pytorch** is the **differentiable 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.

```
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
```

```
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.

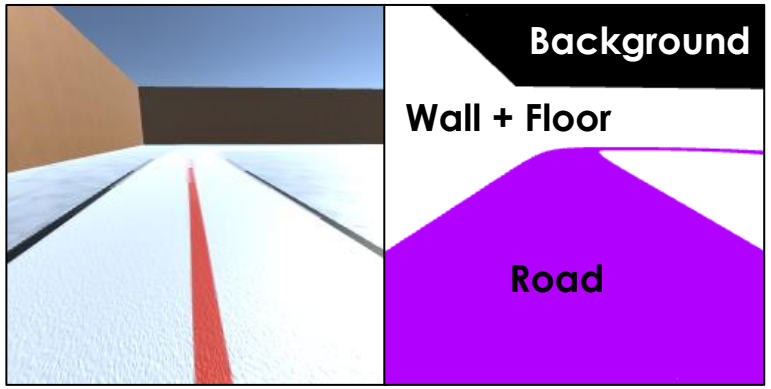
- `torch.utils.data.DataLoader` shuffle the data in the dataset and construct an iterator for us to get a random batch of training data.

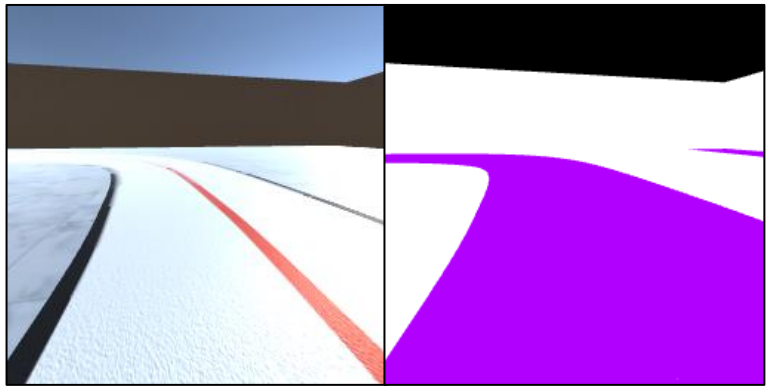
- Ex.

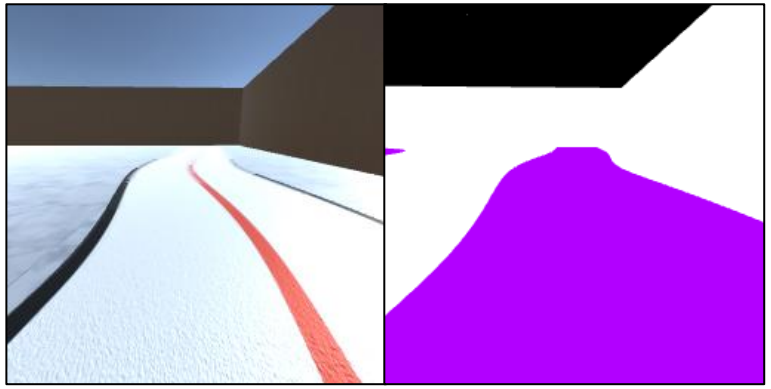
```
train_data = CustomDataset(…)
train_loader = DataLoader(train_data, batch_size=32, shuffle=True)

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

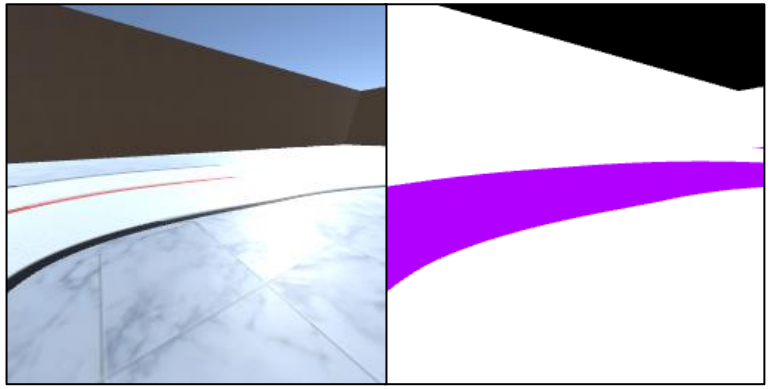
Semantic Segmentation

Simulation Road Dataset



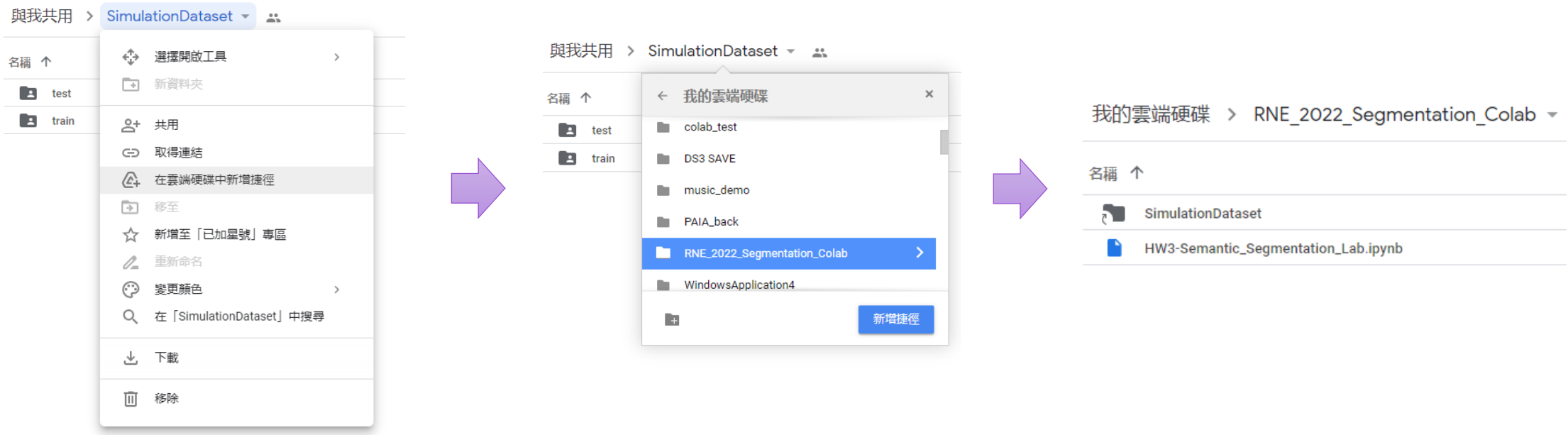






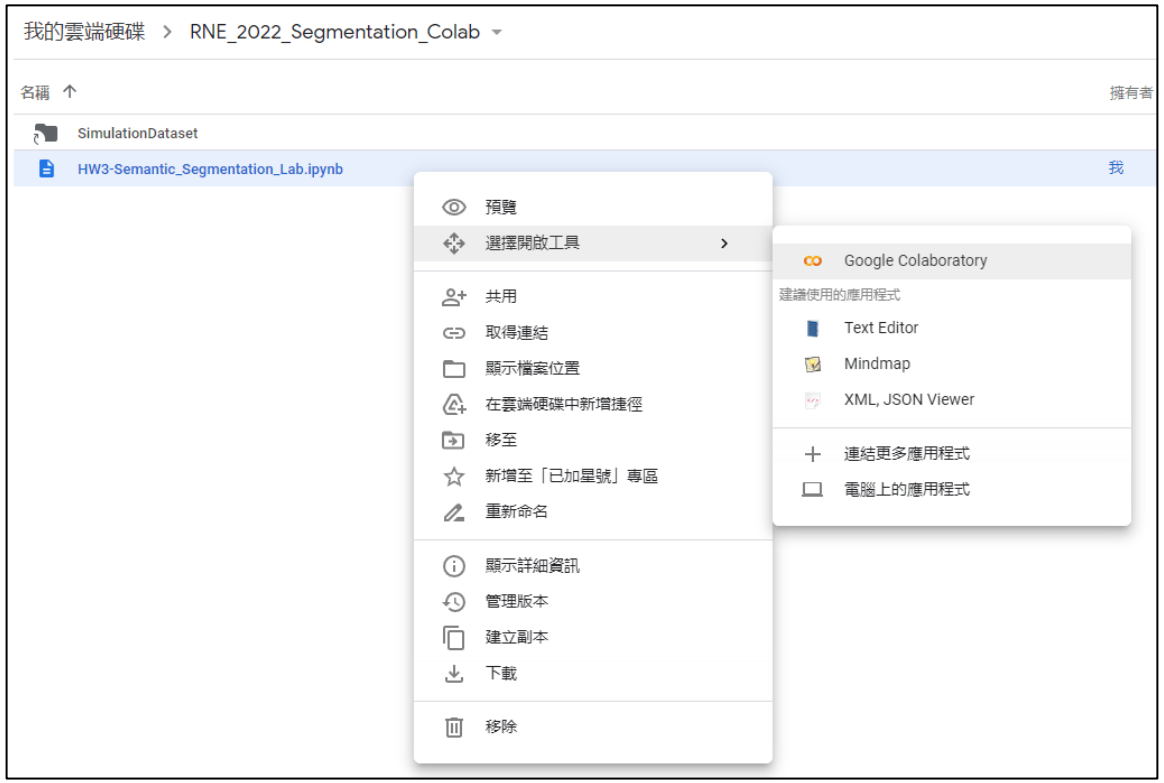
Project and Dataset

- Create a project folder “RNE_2022_Segmentation_Colab” in your google drive and upload the code “HW3-Semantic_Segmentation_Lab.ipynb”.
- Add the shortcut of the dataset to the project folder. https://drive.google.com/drive/folders/1P2fo5DvUVaO7wx0HYs32zWt17HjvhebE?usp=sharing



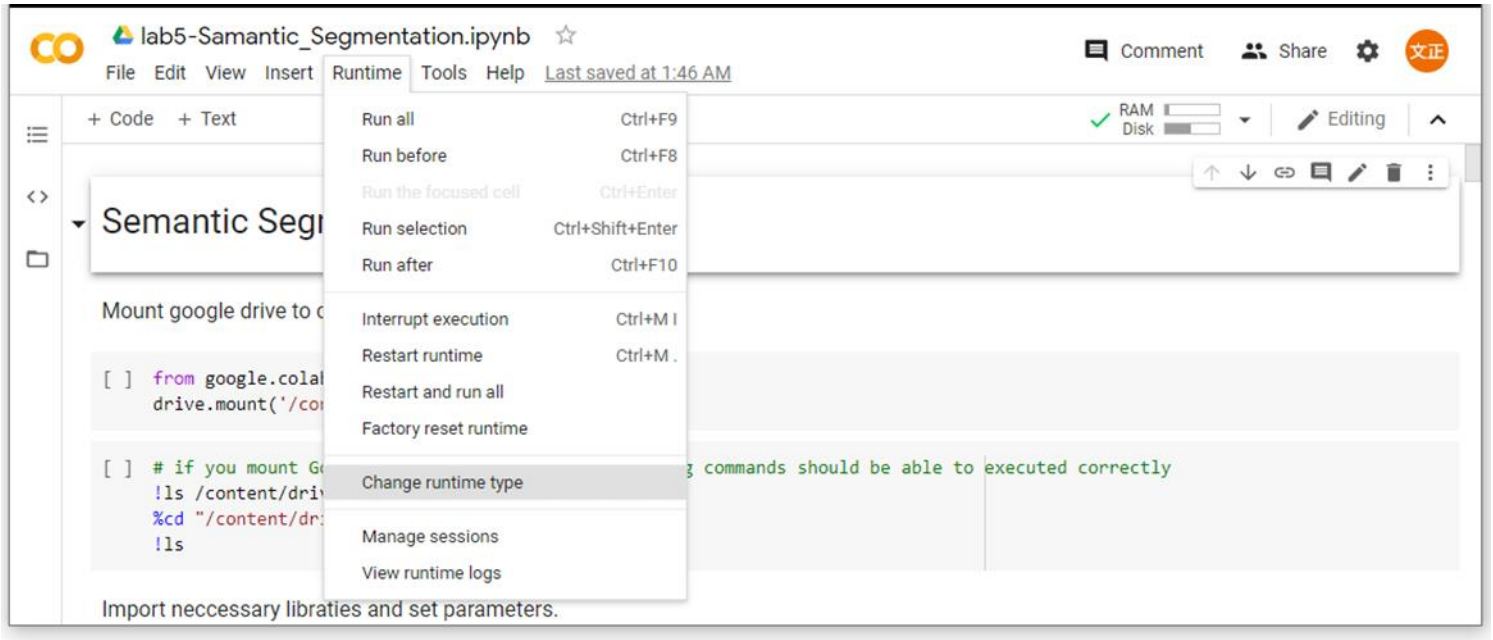
Google Colaboratory

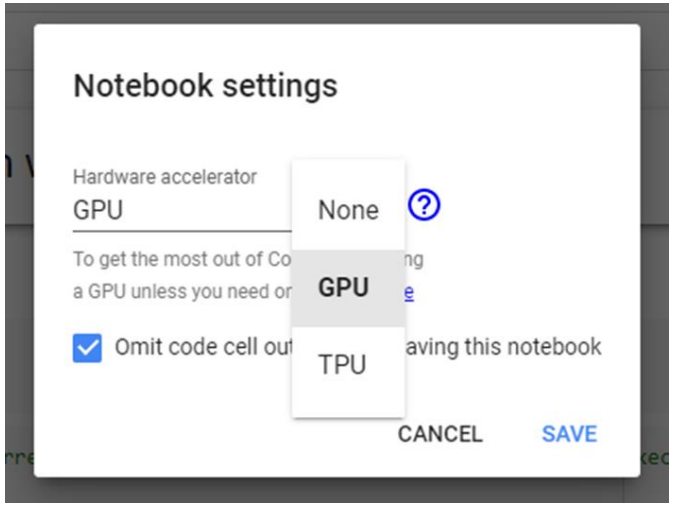
- Put the ipynb file to your google drive, and open it using Google Colaboratory. (If you donnot have Google Colaboratary, you need to click "連結更多應用程式" and install it.)



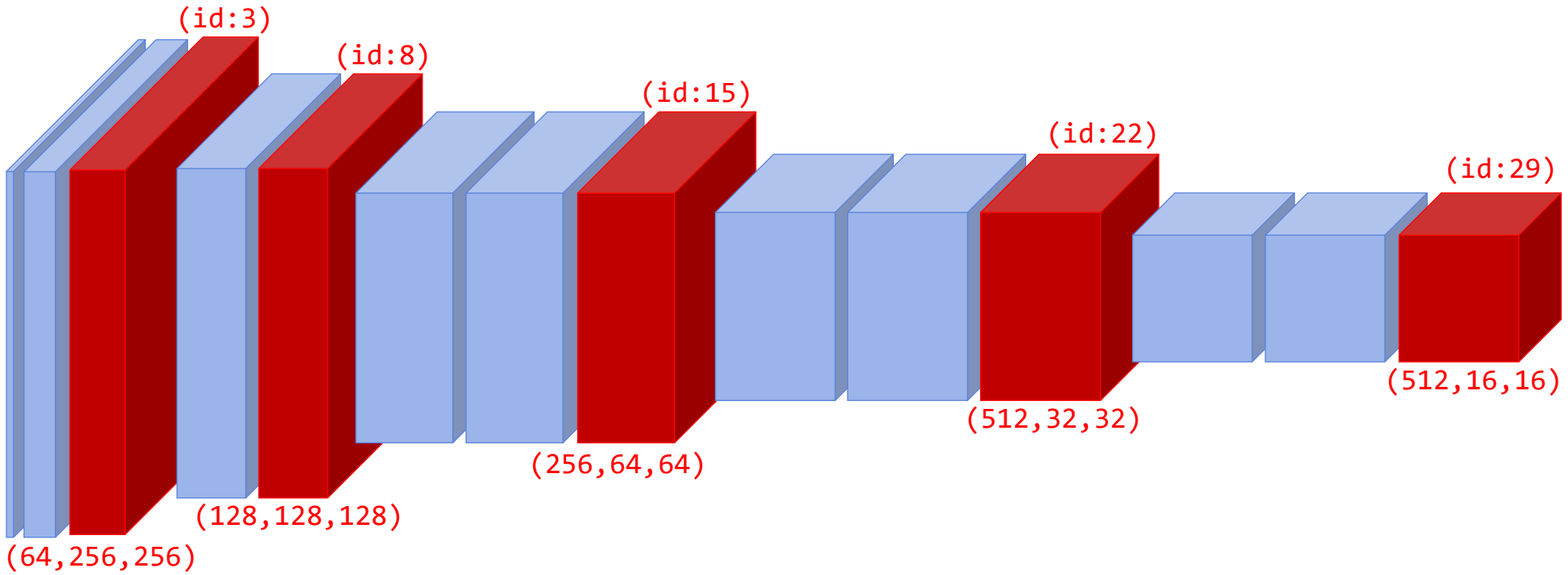
Google Colaboratory

- Set runtime hardware accelerator to “GPU”





VGG-16 Feature Extractor



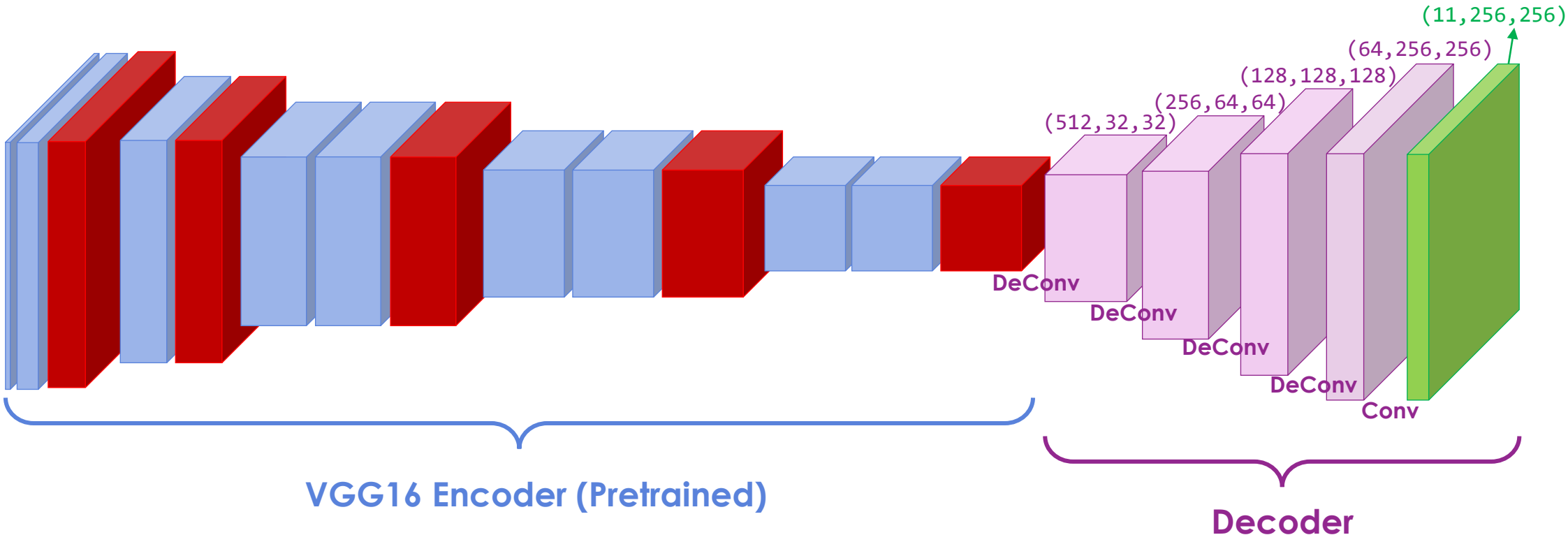
VGG-16 Feature Extractor

```
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

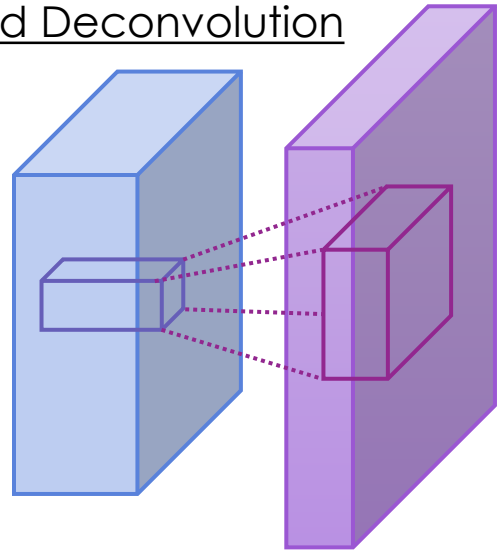
```
ModuleList(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace=True)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace=True)
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): 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

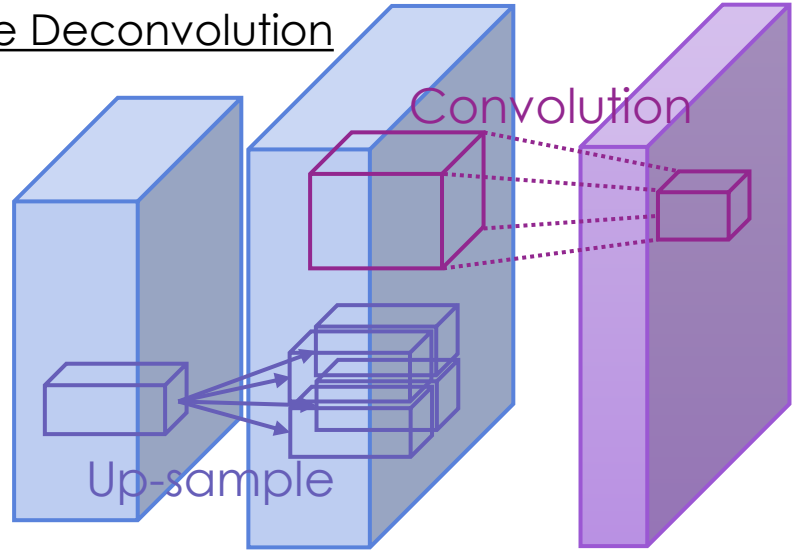


Up-sample Deconvolution

Standard Deconvolution



Up-sample Deconvolution

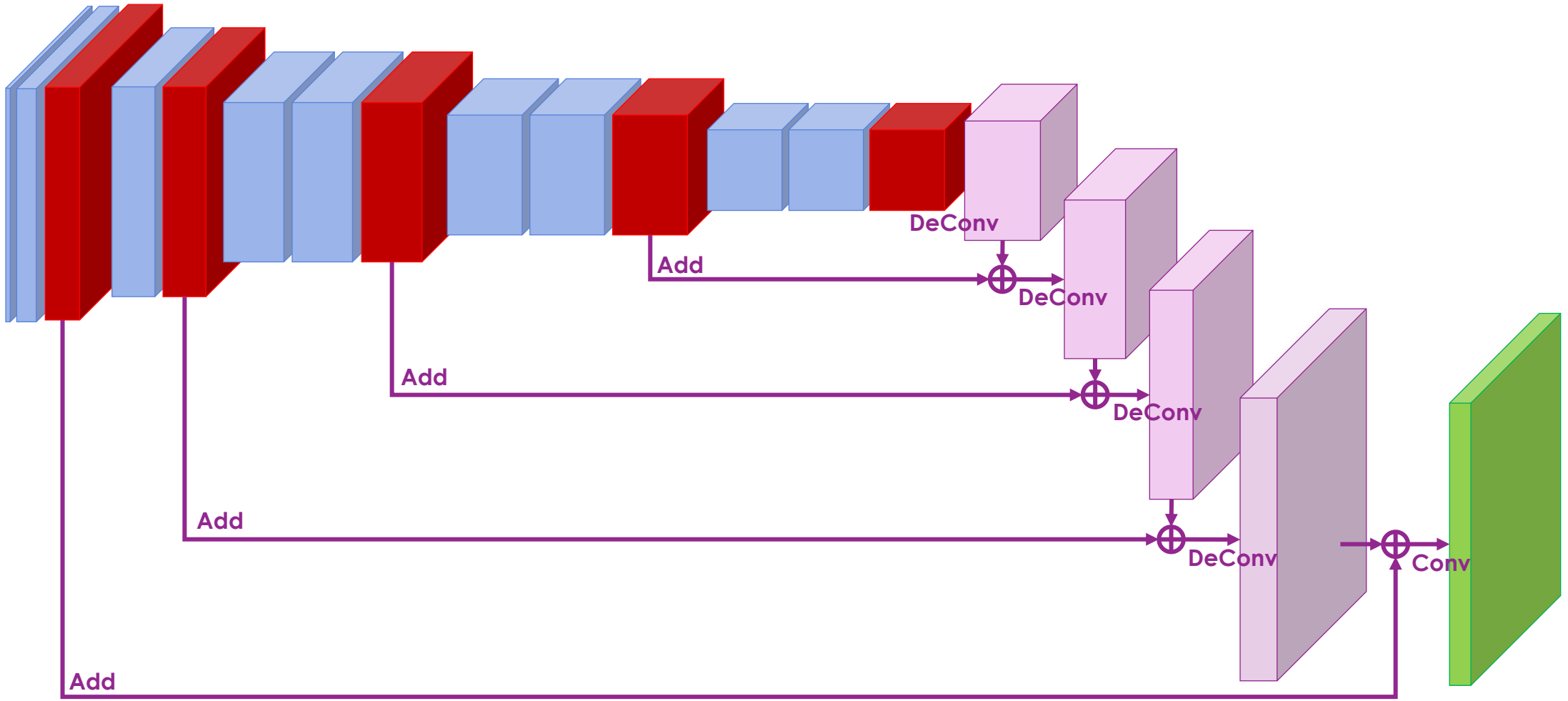



```
class DeConv2d(nn.Module):
    def __init__(self, in_channel, out_channel, kernel_size, stride, padding, dilation):
        super().__init__()
        self.up = nn.Upsample(scale_factor=2, mode='nearest')
        self.conv = nn.Conv2d(in_channel, out_channel, kernel_size=kernel_size, \
         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

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

Encoder-Decoder Architecture

```
class EncoderDecoder(nn.Module):
    def __init__(self, pretrained_net, n_class):
        super().__init__()
        self.n_class = n_class
        self.pretrained_net = pretrained_net
        self.relu = nn.ReLU(inplace=True)

        self.deconv1 = DeConv2d(512, 512, kernel_size=3, stride=1, padding=1, dilation=1)
        self.bn1 = nn.BatchNorm2d(512)
        self.deconv2 = DeConv2d(512, 256, kernel_size=3, stride=1, padding=1, dilation=1)
        self.bn2 = nn.BatchNorm2d(256)
        self.deconv3 = DeConv2d(256, 128, kernel_size=3, stride=1, padding=1, dilation=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.deconv4 = DeConv2d(128, 64, kernel_size=3, stride=1, padding=1, dilation=1)
        self.bn4 = nn.BatchNorm2d(64)

        self.classifier = nn.Conv2d(64, n_class, kernel_size=1)

    def forward(self, x):
        pre_output = self.pretrained_net(x)
        output = self.bn1(self.relu(self.deconv1(pre_output[4]))) #(512,32,32)
        output = self.bn2(self.relu(self.deconv2(output))) #(256,64,64)
        output = self.bn3(self.relu(self.deconv3(output))) #(128,128,128)
        output = self.bn4(self.relu(self.deconv4(output))) #(64,256,256)
        output = self.classifier(output)
        return output
```

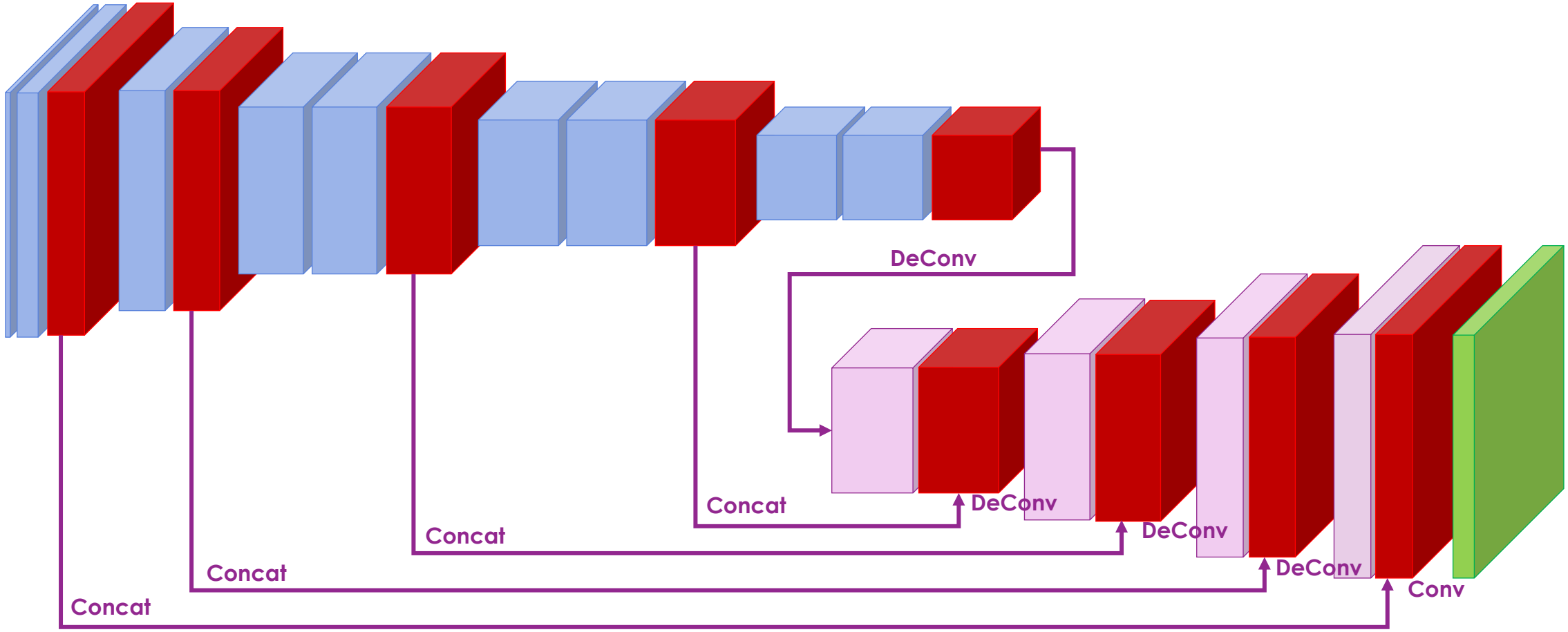
Practice - FCN



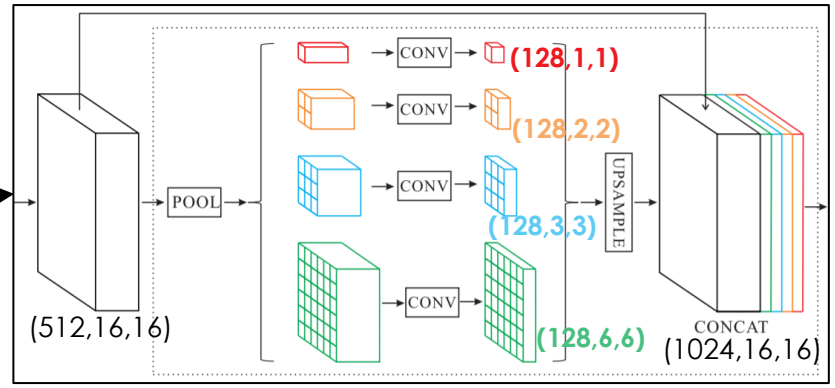
Practice - U-Net

[Hint] Tensor Concatenation

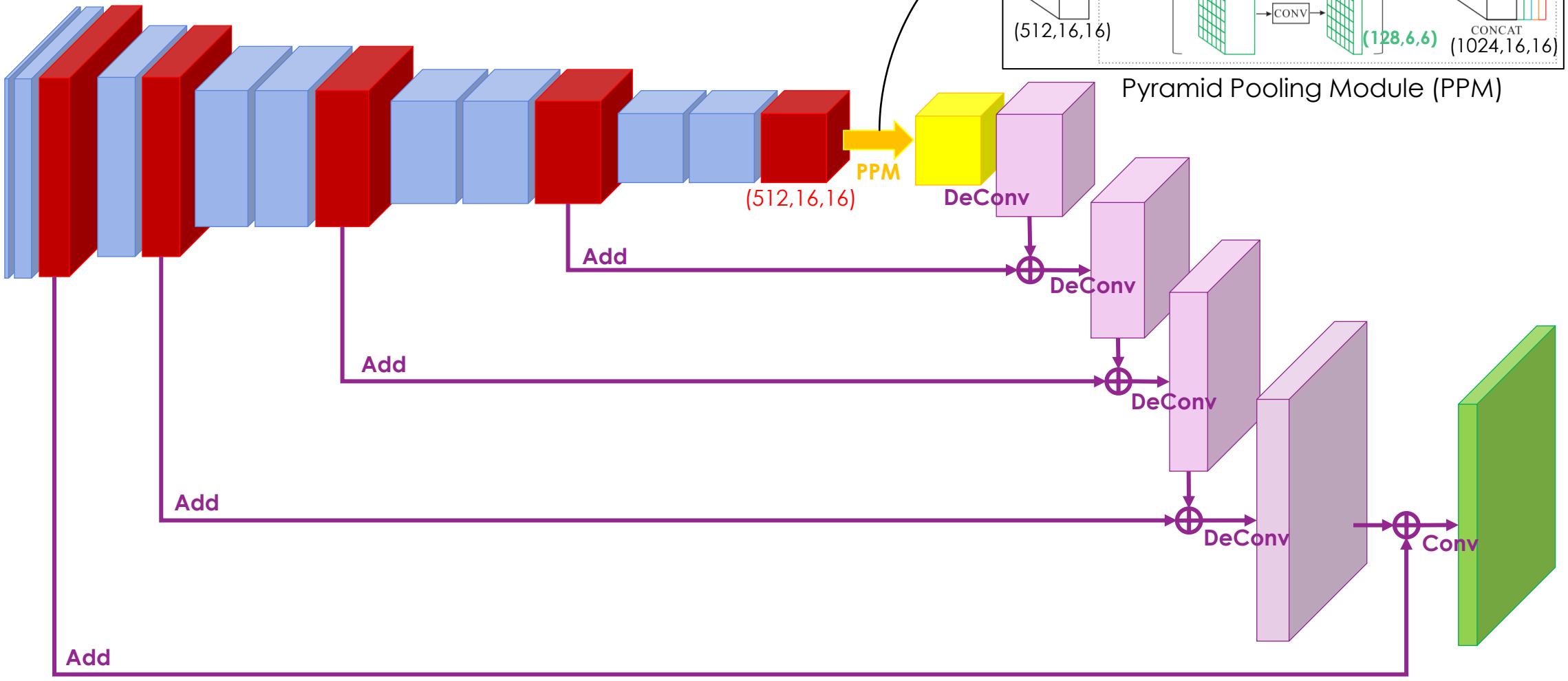
```
tc = torch.cat([t1,t2,…],dim)
```



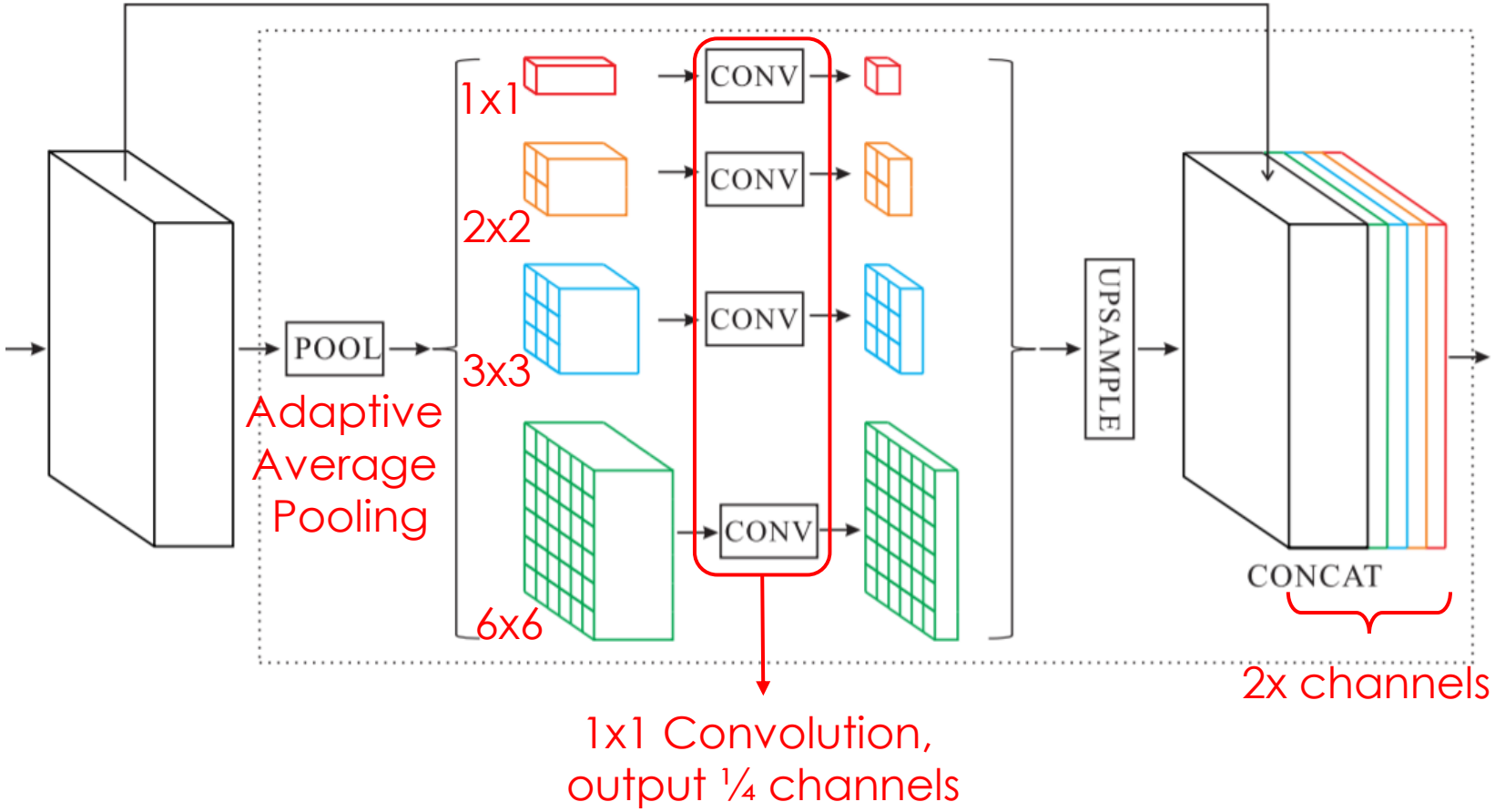
Practice - PSPNet



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)
```

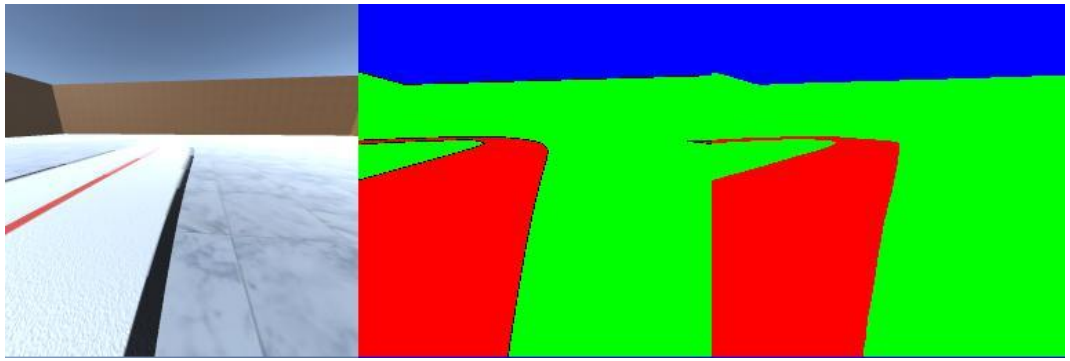
Select Model

```
[ ]  project_name = "RNE_2022_Segmentation_Colab"
     project_path = "/content/drive/My Drive/" + project_name
     train_dataset_path = project_path + "/SimulationDataset/train"
     test_dataset_path = project_path + "/SimulationDataset/test"

     model_type = "encdec" # encdec / fcn / unet / pspnet

     # Create folder to store training results.
     if model_type == "encdec":
            results_path = project_path + "/results_encdec"
     elif model_type == "fcn":
            results_path = project_path + "/results_fcn"
     elif model_type == "unet":
            results_path = project_path + "/results_unet"
     elif model_type == "pspnet":
            results_path = project_path + "/results_pspnet"

     if os.path.isdir(results_path) == False:
          os.mkdir(results_path)
```

Modify this line to call different models.

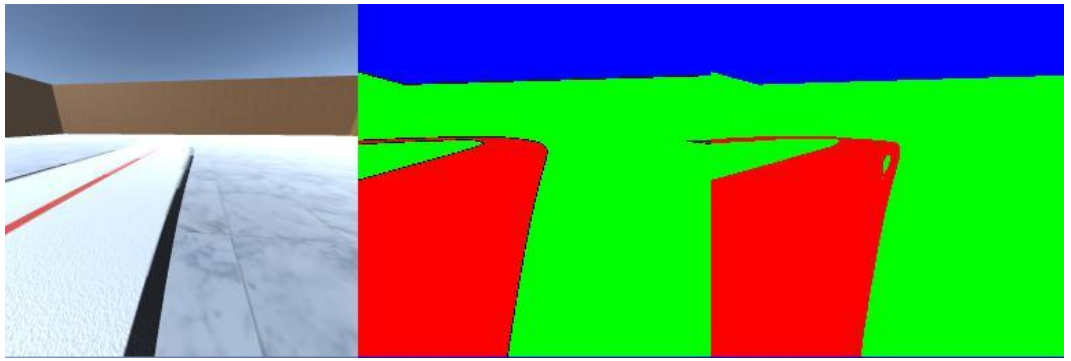
Experimental Results

Encoder-Decoder



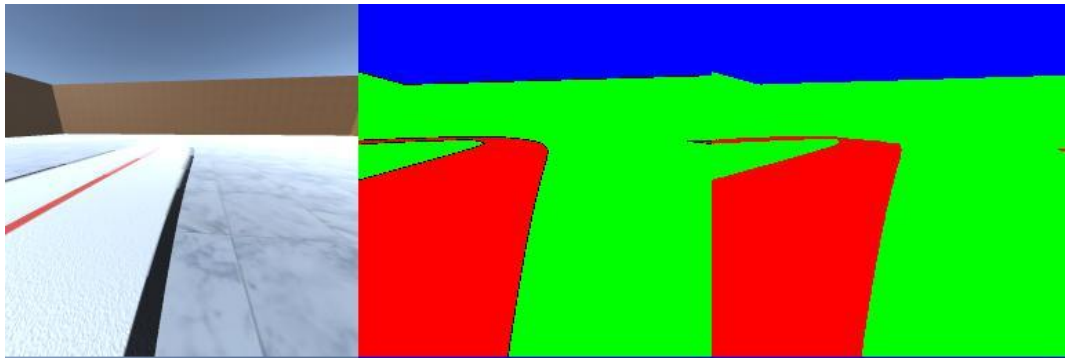
Acc: 0.9923, IOU: 0.9850

U-Net



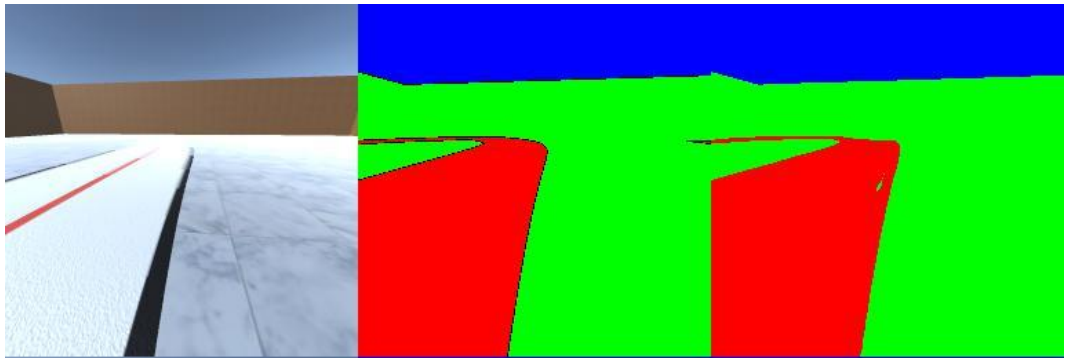
Acc: 0.9934, IOU: 0.9866

FCN



Acc: 0.9930, IOU: 0.9860

PSPNet



Acc: 0.9927, IOU: 0.9855

Score & Requirement

- You should complete the code of “FCN”, “U-Net” and “PSPNet” and train the models.
  - Score: FCN(40%), U-Net(40%), PSPNet(20%)
- Submit the zip of the project folder without the dataset. It should include:
  - Code (.ipynb)
  - Results of each methods (”results_fcn/”, ”results_unet/”, ”results_pspnet/”)
  - Each result folder should include:
    - Result images (“001.jpg”~”010.jpg”)
    - Evaluation results (“record.json”)
    - Weightings (“segnet.pt”)
- **Do not upload the dataset !!**

Deadline

- **Deadline: 2022/05/17 (10:00 pm)**
- **Homework Upload**
  - IP: 140.114.79.183 / Port 21
  - Directory: /RNE/HW3
  - Username: RNE_guest
  - Password: RNE2022nthu
  - 請將完整程式碼壓縮成zip並命名為 [學號]_[姓名]_v[版本].zip (如果要更新上傳檔案請設不同的版本號)