Semantic Segmentation Competition (30%)

For this competition, we will use a small autonomous driving dataset. The dataset contains 150 training images and 50 testing images.

We provide baseline code that includes the following features:

- Loading the dataset using PyTorch.
- Defining a simple convolutional neural network for semantic segmentation.
- How to use existing loss function for the model learning.
- Train the network on the training data.
- Test the trained network on the testing data.

The following changes could be considered:

1. Data augmentation
2. Change of advanced training parameters: Learning Rate, Optimizer, Batch-size, and Drop-out.
3. Architectural changes: Batch Normalization, Residual layers, etc.
4. Use of a new loss function.

Your code should be modified from the provided baseline. A pdf report of a maximum of two pages is required to explain the changes you made from the baseline, why you chose those changes, and the improvements they achieved.

Marking Rules:

We will mark the competition based on the final test accuracy on testing images and your report.

Final mark (out of 50) = acc_mark + efficiency mark + report mark

Acc_mark 10:

We will rank all the submission results based on their test accuracy. Zero improvement over the baseline yields 0 marks. Maximum improvement over the baseline will yield 10 marks. There will be a sliding scale applied in between.

Efficiency mark 10:

Efficiency considers not only the accuracy, but the computational cost of running the model (flops: https://en.wikipedia.org/wiki/FLOPS). Efficiency for our purposes is defined to be the ratio of accuracy (in %) to Gflops. Please report the computational cost for your final model and include the efficiency calculation in your report. Maximum improvement over the baseline will yield 10 marks. Zero improvement over the baseline yields zero marks, with a sliding scale in between.

Report mark 30:

Your report should comprise:

1. An introduction showing your understanding of the task and of the baseline model: [10 marks]
2. A description of how you have modified aspects of the system to improve performance. [10 marks]

A recommended way to present a summary of this is via an "ablation study" table, eg:

Method1	Method2	Method3	Accuracy
N	N	N	60%
Y	N	N	65%
Y	Y	N	77%
Y	Y	Y	82%

3. Explanation of the methods for reducing the computational cost and/or improve the trade-off between accuracy and cost: [5 marks]
4. Limitations/Conclusions: [5 marks]

1. Download data and set configs

```
##########################################################################################################################
### Subject: Computer Vision
### Year: 2025
### Student Name: ABC, XYZ
### Student ID: a123456, a654321
### Comptetion Name: Semantic Segmentation Competition
### Final Results:
### ACC:         GFLOPs:
##########################################################################################################################
```

```
#1.1 Download the dataset.
# dowanload and unzip the dataset


#1.2 Set configs
#Use Colab or install PyTorch 1.9 on your local machine to run the code.
import os
import numpy as np
import cv2
import torchvision.models.segmentation
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as tf
from PIL import Image
import shutil
from torch.utils.data import Dataset, DataLoader

#--------Data path----------
# Use your data path to replace the following path if you use Google drive.
from google.colab import drive
drive.mount('/content/drive')

# Dataset path. Ensure that the file path correspond to the path you have here. It is expected that you unzip the data folde
dataFolder = '/content/drive/MyDrive/Datasets/seg_data'


# To access Google Colab GPU; Go To: Edit >>> Notebook Settings >>> Hardware Accelarator: Select GPU.
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
print('device: {}'.format(device))

#---------Config----------
learning_rate = 3e-4 #Tips: design a strategy to adjust the learning rate
width = 864 # image width and height
height = 256 #
batchSize = 4 #can be adjusted
epochs = 180 #can be adjusted

# if not os.path.exists(dataFolder):
#    print('Data Path Error! Pls check your data path')
if not torch.cuda.is_available():
  print('WARNING! The device is CPU NOT GPU! Pls avoid using CPU for training')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remoun
device: cuda
```

2. Define a dataloader to load data

```
#The class to load images and labels
class ExpDataSet(Dataset):
    def __init__(self, dataFolder):
        self.image_path = os.listdir(os.path.join(dataFolder, "training/image"))
        self.label_path = os.listdir(os.path.join(dataFolder, "training/image"))#Image name only
        print('load info for {} images'.format(len(self.image_path)))
        assert len(self.image_path) == 150
        for idx in range(0, len(self.image_path)):
            assert self.image_path[idx] == self.label_path[idx] #same
            self.image_path[idx] = os.path.join(dataFolder, "training/image", self.image_path[idx])
            self.label_path[idx] = os.path.join(dataFolder, "training/label", self.label_path[idx])
        # --------------------Transformation functions----------------
        #-------------Tips: data augmentation can be used (for example flip, resize)-------------
        self.transformImg = tf.Compose([tf.ToPILImage(), tf.Resize((height, width)), tf.ToTensor(),
                                        tf.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))])
        self.transformLabel = tf.Compose([tf.ToPILImage(), tf.Resize((height, width), tf.InterpolationMode.NEAREST)])
    def __getitem__(self, idx):
        img = cv2.imread(self.image_path[idx])[:, :, 0:3]
        label = cv2.imread(self.label_path[idx], 0)
        img = self.transformImg(img)  #3*H*W
        label = self.transformLabel(label)
        label = torch.tensor(np.array(label))  #H*W
        return img, label
    def __len__(self):
        return len(self.image_path)

#Get the predefined dataloader
exp_data = ExpDataSet(dataFolder)
train_loader = DataLoader(exp_data, batch_size=batchSize, shuffle=True, num_workers=2)
```

```
load info for 150 images
```

3. Define a convolutional neural network

```
#Define the semantic segmentation network. Tips: a new network can be used
class SegNetwork(nn.Module):
    def __init__(self, n_class=19):
        super(SegNetwork, self).__init__()
        #stage 1
        self.conv1_1 = nn.Conv2d(3, 64, 3, padding=1)
        self.relu1_1 = nn.ReLU(inplace=True)
        self.conv1_2 = nn.Conv2d(64, 64, 3, padding=1)
        self.relu1_2 = nn.ReLU(inplace=True)
        self.pool1 = nn.MaxPool2d(2, stride=2, ceil_mode=True)  #1/2
        #stage 2
        self.conv2_1 = nn.Conv2d(64, 128, 3, padding=1)
        self.relu2_1 = nn.ReLU(inplace=True)
        self.pool2 = nn.MaxPool2d(2, stride=2, ceil_mode=True)  #1/4
        #stage 3
        self.conv3_1 = nn.Conv2d(128, 256, 3, padding=1)
        self.relu3_1 = nn.ReLU(inplace=True)
        self.pool3 = nn.MaxPool2d(2, stride=2, ceil_mode=True)  #1/8
        #stage 4
        self.conv4_1 = nn.Conv2d(256, 512, 3, padding=1)
        self.relu4_1 = nn.ReLU(inplace=True)
        self.pool4 = nn.MaxPool2d(2, stride=2, ceil_mode=True)  #1/16
        #stage 5
        self.conv5_1 = nn.Conv2d(512, 2048, 3)
        self.relu5_1 = nn.ReLU(inplace=True)
        self.drop5_1 = nn.Dropout2d()
        self.conv5_2 = nn.Conv2d(2048, 2048, 1)
        self.relu5_2 = nn.ReLU(inplace=True)
        self.drop5_2 = nn.Dropout2d()
        self.conv5_3 = nn.Conv2d(2048, n_class, 1)
        #upsample
        self.upsample_cov1 = nn.ConvTranspose2d(n_class, n_class, 4, stride=2, output_padding=1, bias=False) #upsample
        self.upsample_cov2 = nn.ConvTranspose2d(n_class, n_class, 4, stride=2, bias=False) #upsample  -> 1/4
        self.upsample_cov3 = nn.ConvTranspose2d(n_class, n_class, 4, stride=2, padding=1, bias=False)  # upsample  -> 1/2

    def forward(self, x):
        inp_shape = x.shape[2:]
        x = self.relu1_1(self.conv1_1(x))
        x = self.relu1_2(self.conv1_2(x))
        x = self.pool1(x)

        x = self.relu2_1(self.conv2_1(x))
        x = self.pool2(x)

        x = self.relu3_1(self.conv3_1(x))
        x = self.pool3(x)

        x = self.relu4_1(self.conv4_1(x))
        x = self.pool4(x)

        x = self.relu5_1(self.conv5_1(x))
        x = self.drop5_1(x)
        x = self.relu5_2(self.conv5_2(x))
        x = self.drop5_2(x)
        x = self.conv5_3(x)

        x = self.upsample_cov1(x)
        x = self.upsample_cov2(x)
        x = self.upsample_cov3(x)
        x = F.interpolate(x, size=inp_shape, mode="bilinear", align_corners=True)#resize (1/2 -> original size)
        return x

#Get the predefined network
segNet = SegNetwork(n_class=19).to(device)
```

4. Define a loss function and optimizer

```
criterion = torch.nn.CrossEntropyLoss(ignore_index=255)
optimizer = torch.optim.Adam(params=segNet.parameters(), lr=learning_rate)
```

5. The function used to compare the precision

```
#------------------Modification of this function is ***NOT*** allowed---------------
def cal_acc(pred_folder, gt_folder, classes=19):
    class AverageMeter(object):
        def __init__(self):
            self.reset()
        def reset(self):
            self.val, self.avg, self.sum, self.count = 0, 0, 0, 0
        def update(self, val, n=1):
            self.val = val
            self.sum += val * n
            self.count += n
            self.avg = self.sum / self.count
    def intersectionAndUnion(output, target, K, ignore_index=255):
        assert (output.ndim in [1, 2, 3])
        assert output.shape == target.shape
        output = output.reshape(output.size).copy()
        target = target.reshape(target.size)
        output[np.where(target == ignore_index)[0]] = ignore_index
        intersection = output[np.where(output == target)[0]]
        area_intersection, _ = np.histogram(intersection, bins=np.arange(K + 1))
        area_output, _ = np.histogram(output, bins=np.arange(K + 1))
        area_target, _ = np.histogram(target, bins=np.arange(K + 1))
        area_union = area_output + area_target - area_intersection
        return area_intersection, area_union, area_target
    data_list = os.listdir(gt_folder)
    intersection_meter = AverageMeter()
    union_meter = AverageMeter()
    target_meter = AverageMeter()
    for i, image_name in enumerate(data_list):
        pred = cv2.imread(os.path.join(pred_folder, image_name), cv2.IMREAD_GRAYSCALE)
        target = cv2.imread(os.path.join(gt_folder, image_name), cv2.IMREAD_GRAYSCALE)
        intersection, union, target = intersectionAndUnion(pred, target, classes)
        intersection_meter.update(intersection)
        union_meter.update(union)
        target_meter.update(target)
    iou_class = intersection_meter.sum / (union_meter.sum + 1e-10)
    mIoU = np.mean(iou_class)
    print('Eval result: mIoU {:.4f}.'.format(mIoU))
    return mIoU
```

6. Define functions to get and save predictions

```
def make_folder(dir_name):#make a folder
    if not os.path.exists(dir_name):
        os.makedirs(dir_name)

def move_folders(grey_temp, color_temp, grey_rs, color_rs):#move folders
    if os.path.exists(grey_temp):
        make_folder(grey_rs)
        for file in os.listdir(grey_temp):
            shutil.move(os.path.join(grey_temp, file), os.path.join(grey_rs, file))
        if os.path.exists(grey_temp):
            shutil.rmtree(grey_temp)
    if os.path.exists(color_temp):
        make_folder(color_rs)
        for file in os.listdir(color_temp):
            shutil.move(os.path.join(color_temp, file), os.path.join(color_rs, file))
        if os.path.exists(color_temp):
            shutil.rmtree(color_temp)

def colorize(gray, palette):#visualize predictions results
    color = Image.fromarray(gray.astype(np.uint8)).convert('P')
    color.putpalette(palette)
    return color

#-------Perform evaluation for a network and save prediction results--------
def get_predictions(segNet, dataFolder, device):#params: a network, data path, device
    gray_folder, color_folder = './temp_grey', './temp_color'
    listImages, gt_folder = os.listdir(os.path.join(dataFolder, "testing/image")), os.path.join(dataFolder, "testing/label")
    colors_path  = os.path.join(dataFolder, "colors.txt") #colors for visualizing greyscale images
    print('Begin testing')
    make_folder(gray_folder)
    make_folder(color_folder)
    colors = np.loadtxt(colors_path).astype('uint8')
    #Tips: muti-scale testing can be used
    transformTest = tf.Compose([tf.ToPILImage(), tf.ToTensor(),
                                tf.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))])
    for idx in range(0, len(listImages)):
        img = cv2.imread(os.path.join(dataFolder, "testing/image", listImages[idx]))[:, :, 0:3]
        img = transformTest(img).unsqueeze(0)  #1*3*H*W
        prediction = segNet(img.to(device))
        prediction = prediction[0].cpu().detach().numpy()
        prediction = np.argmax(prediction, axis=0)
        gray = np.uint8(prediction)
        color = colorize(gray, colors)
        gray_path = os.path.join(gray_folder, listImages[idx])
        color_path = os.path.join(color_folder, listImages[idx])
        cv2.imwrite(gray_path, gray)
        color.save(color_path)
    return gray_folder, color_folder #return folders (paths) which contain grey and color predictions.
```

7. Train the network

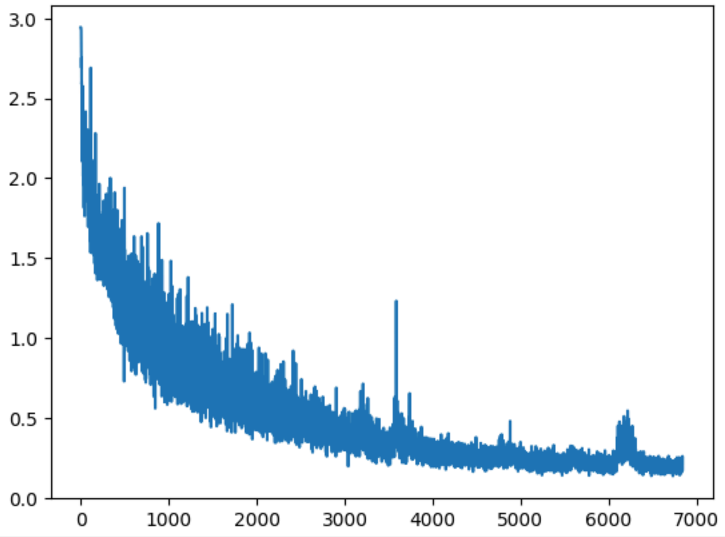
```
#The training will take ~1 h.
mIoU = 0.0
IoU_list = []
train_loss = []
evl_each = True #Perform evaluation after each epoch. You can define the false case to save training time
for epoch in range(epochs):
    for iter, (imgs, labels) in enumerate(train_loader):
        pred = segNet(imgs.to(device))
        segNet.zero_grad()
        loss = criterion(pred, labels.long().to(device)) #calculate the loss
        train_loss.append(loss.detach().cpu().numpy().item())
        loss.backward()
        optimizer.step()
        print('epoch {} iter {} loss={}'.format(epoch, iter, loss.data.cpu().numpy()))

#-----Evaluation------
    if evl_each and epoch>100:
        segNet.eval()  # The eval() must be called before evaluation
        gray_folder, color_folder = get_predictions(segNet, dataFolder, device) #Temp prediction results
        segNet.train()
        temp_mIoU = cal_acc(gray_folder, os.path.join(dataFolder, 'testing/label'))
        IoU_list.append(temp_mIoU)
        if temp_mIoU > mIoU:
            mIoU = temp_mIoU
            torch.save(segNet.state_dict(), './model.pth')
            move_folders(gray_folder, color_folder, # Temp results -> final results
                         gray_folder.replace('temp_', ''),
                         color_folder.replace('temp_', ''))
print('The final mIoU is : {:.4f}.'.format(mIoU)) #The final mIoU is ~0.28
#Remember to download the results before closing the tab!
```

```
epoch 179 iter 20 loss=0.24577847123146057
epoch 179 iter 21 loss=0.21724463999271393
epoch 179 iter 22 loss=0.1547759622335434
epoch 179 iter 23 loss=0.16616399586200714
epoch 179 iter 24 loss=0.18162526190280914
epoch 179 iter 25 loss=0.18914397060871124
epoch 179 iter 26 loss=0.2247447967529297
epoch 179 iter 27 loss=0.1891363263130188
epoch 179 iter 28 loss=0.19037602841854095
epoch 179 iter 29 loss=0.18791884183883667
epoch 179 iter 30 loss=0.19215872883796692
epoch 179 iter 31 loss=0.17535221576690674
epoch 179 iter 32 loss=0.20027194917201996
epoch 179 iter 33 loss=0.2064836323261261
epoch 179 iter 34 loss=0.16802836954593658
epoch 179 iter 35 loss=0.20362138748168945
epoch 179 iter 36 loss=0.2597014605998993
epoch 179 iter 37 loss=0.17616219818592072
Begin testing
Eval result: mIoU 0.2734.
The final mIoU is : 0.2843.
```

```
from matplotlib import pyplot as plt
```

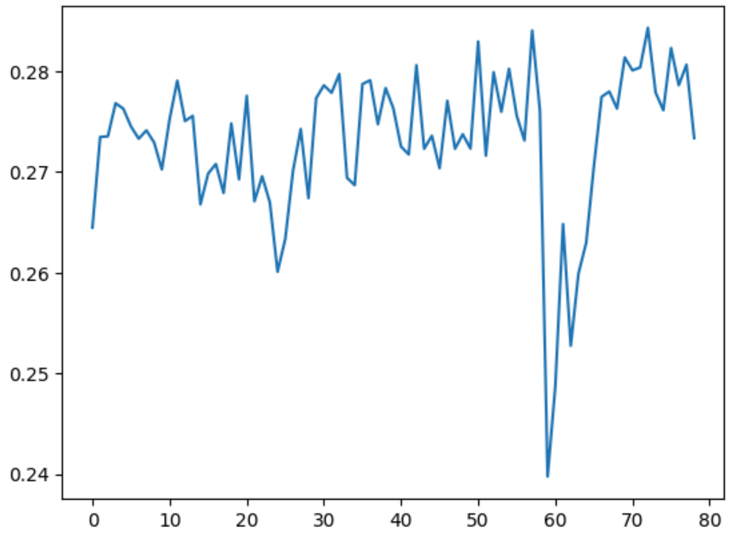
```
plt.plot(train_loss,label="train_loss")
```

```
[<matplotlib.lines.Line2D at 0x794d7b995050>]
```

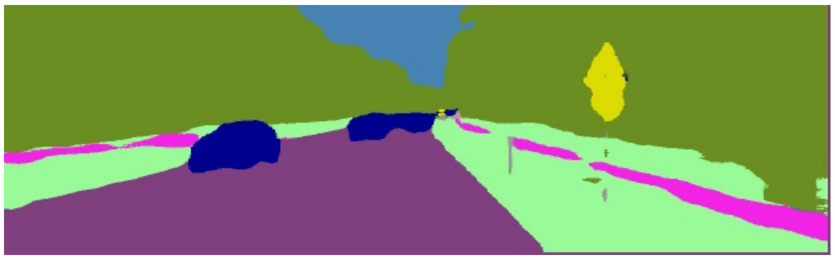


```
plt.plot(IoU_list,label="IoU")
```

```
[<matplotlib.lines.Line2D at 0x794d7e2a1150>]
```



A prediction example by using the baseline:



8. FLOPs

In deep learning, FLOPs (Floating Point Operations) quantify the total number of arithmetic operations—such as additions, multiplications, and divisions—that a model performs during a single forward pass (i.e., when making a prediction). This metric serves as an indicator of a model's computational complexity. When discussing large-scale models, FLOPs are often expressed in GFLOPs (Giga Floating Point Operations), where 1 GFLOP equals one billion operations. This unit helps in comparing the computational demands of different models.

```
# we use fvcore to calculate the FLOPs
!pip install fvcore
```

```
Collecting fvcore
  Downloading fvcore-0.1.5.post20221221.tar.gz (50 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50.2/50.2 kB 2.8 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from fvcore) (2.0.2)
Collecting yacs>=0.1.6 (from fvcore)
  Downloading yacs-0.1.8-py3-none-any.whl.metadata (639 bytes)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from fvcore) (6.0.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from fvcore) (4.67.1)
Requirement already satisfied: termcolor>=1.1 in /usr/local/lib/python3.11/dist-packages (from fvcore) (3.1.0)
Requirement already satisfied: Pillow in /usr/local/lib/python3.11/dist-packages (from fvcore) (11.2.1)
Requirement already satisfied: tabulate in /usr/local/lib/python3.11/dist-packages (from fvcore) (0.9.0)
Collecting iopath>=0.1.7 (from fvcore)
  Downloading iopath-0.1.10.tar.gz (42 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.2/42.2 kB 4.1 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: typing_extensions in /usr/local/lib/python3.11/dist-packages (from iopath>=0.1.7->fvcore)
Collecting portalocker (from iopath>=0.1.7->fvcore)
  Downloading portalocker-3.1.1-py3-none-any.whl.metadata (8.6 kB)
Downloading yacs-0.1.8-py3-none-any.whl (14 kB)
Downloading portalocker-3.1.1-py3-none-any.whl (19 kB)
Building wheels for collected packages: fvcore, iopath
  Building wheel for fvcore (setup.py) ... done
  Created wheel for fvcore: filename=fvcore-0.1.5.post20221221-py3-none-any.whl size=61397 sha256=27855a2c23a3e442d38170
  Stored in directory: /root/.cache/pip/wheels/65/71/95/3b8fde5c65c6e4a806e0867c1651dcc71a1cb2f3430e8f355f
  Building wheel for iopath (setup.py) ... done
  Created wheel for iopath: filename=iopath-0.1.10-py3-none-any.whl size=31527 sha256=b9ae3ea9215aefe13ee2087cd2d038cdab
  Stored in directory: /root/.cache/pip/wheels/ba/5e/16/6117f8fe7e9c0c161a795e10d94645ebcf301ccbd01f66d8ec
Successfully built fvcore iopath
Installing collected packages: yacs, portalocker, iopath, fvcore
Successfully installed fvcore-0.1.5.post20221221 iopath-0.1.10 portalocker-3.1.1 yacs-0.1.8
```

```
from fvcore.nn import FlopCountAnalysis
input = torch.randn(1, 3, 375, 1242) # Modifying the size (3, 375, 1242) is ***NOT*** allowed.

#Get the network and its FLOPs
model = SegNetwork(n_class=19)
flops = FlopCountAnalysis(model, input)
print(f"FLOPs: {flops.total()/1e9:.2f} GFLOPs")
```

```
WARNING:fvcore.nn.jit_analysis:Unsupported operator aten::max_pool2d encountered 4 time(s)
WARNING:fvcore.nn.jit_analysis:Unsupported operator aten::feature_dropout encountered 2 time(s)
FLOPs: 66.97 GFLOPs
```

Start coding or generate with AI.