# Semantic Segmentation Competition (40%)

For this competition, we will use a small autonomous driving dataset. The dataset contains 150 training images and 50 testing images. It can be download from https://drive.google.com/drive/folders/1UyGysAa9S7Jw-BRgouwNoSNgIZqEx7EX?usp=sharing or https://cloudstor.aarnet.edu.au/plus/s/qtk9MB74r0q1cVh

We provide a baseline by the following steps:

- 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 trick/tweak(s) 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, Attention Block, and other varients.
4. Use of a new loss function.

Your code should be modified from the provided baseline. A pdf report is required to explain the tricks you employed, and the imporvements they achieved.

## Marking Rules:

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

Final mark = acc_mark + efficiency mark + report mark + bonus mark

### Acc_mark 15:

We will rank all the submission results based on their test accuracy. The top 30% of the students will get full marks.

| Accuracy | Mark |
|---|---|
| Top 30% in the class | 15 |
| 30%-50% | 11 |
| 50%-80% | 7 |
| 80%-90% | 3 |
| 90%-100% | 1 |
| Not implemented | 0 |

## Efficiency mark 5:

Efficiency is evaluated by the computational costs (flops: https://en.wikipedia.org/wiki/FLOPS). Please report the computational costs for your final model. We provide an evaluation code to calculate flops.

| Efficiency | Mark |
|---|---|
| Top 30% in the class | 5 |
| 30%-50% | 4 |
| 50%-80% | 3 |
| 80%-90% | 2 |
| 90%-100% | 2 |
| Not implemented | 0 |

## Report mark 20:

1. Introduction and your understanding to the baseline model: 2 points

2. Employed more than three tricks with ablation studies to improve the accuracy: 6 points

Clearly explain the reference, motivation and design choice for each trick/tweak(s). Providing the experimental results in tables. Example table:

| Trick1 | Trick2 | Trick3 | Accuracy |
|---|---|---|---|
| N | N | N | 28.44% |
| Y | N | N | 30.15% |
| Y | Y | N | 31.39% |
| Y | Y | Y | 31.88% |

Observation and discussion based on the experiment results.

1. Expaination of the methods on reducing the computational cost and/or improve the trade-off between accuracy and efficiency: 4 points
2. Explaination of the code implementation： 3 points
3. Visulization results: segmentation examples (color images) on test set: 3 points
4. Open ended: Limitations, conclusions, failure cases analysis...: 2 points

## Bonus mark:

1. Top three results: 2 points
2. Fancy designs: 2 points

# 1. Download data and set configs

In [168...

```
#########################################################################
### Subject: Computer Vision
### Year: 2022
### Student Name: Chunyu Zhang, Haonan Zhang
### Student ID: a1751743, a1742769
### Comptetion Name: Semantic Segmentation Competition
### Final Results: Final mIoU is 0.3188
### ACC: 31.88%          FLOPs: 158.80G
#########################################################################
```

In [169...

```
#1.1 Download the dataset.
!wget https://cloudstor.aarnet.edu.au/plus/s/qtk9MB74r0q1cVh/download
!unzip download
!rm -rf download
```

In [170…

```
#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 matplotlib.pyplot as plt
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 driv
dataFolder = './seg_data'

# To access Google Colab GPU; Go To: Edit >>> Notebook Settings >>> Hardw
device = torch.device('cuda') if torch.cuda.is_available() else torch.dev
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 = 5 #can be adjusted
epochs = 160 #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 trai
```

```
device: cuda
```

# 2. Define a dataloader to load data

In [171…

```
#The class to load images and labels
class ExpDataSet(Dataset):
    def __init__(self, dataFolder):
        self.image_path = os.listdir(os.path.join(dataFolder, "training/i
        self.label_path = os.listdir(os.path.join(dataFolder, "training/i
        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/ima
            self.label_path[idx] = os.path.join(dataFolder, "training/lab
        # --------------------Transformation functions----------------
        #-------------Tips: data augmentation can be used (for example fl

        #data augmentation
        self.transformImg = tf.Compose(
            [
                tf.ToPILImage(),
                tf.Resize((288, 972)),
                tf.ToTensor(),
                tf.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)
             ]
        )
        self.transformLabel = tf.Compose(
            [
                tf.ToPILImage(),
                tf.Resize((288, 972), 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, n
```

```
load info for 150 images
```

# 3. Define a convolutional neural network

In [172…

```
#Define the semantic segmentation network. Tips: a new network can be use
class SegNetwork(nn.Module):
    def __init__(self, n_class=19):
        super(SegNetwork, self).__init__()
        #stage 1
        self.conv1_1 = nn.Conv2d(3, 8*8, 3, padding=1)     #make network
        self.relu1_1 = nn.ReLU(inplace=True)
        self.conv1_2 = nn.Conv2d(8*8, 8*8, 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(8*8, 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, 16*16, 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(16*16, 28*28, 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(28*28, 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, stri
        self.upsample_cov2 = nn.ConvTranspose2d(n_class, n_class, 4, stri
        self.upsample_cov3 = nn.ConvTranspose2d(n_class, n_class, 4, stri

    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_corne
        return x

#Get the predefined network
```

```
segNet = SegNetwork(n_class=19).to(device)
```

# 4. Define a loss function and optimizer

In [173…

```
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

In [174…

```
#------------------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(
        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.IMRE
        target = cv2.imread(os.path.join(gt_folder, image_name), cv2.IMRE
        intersection, union, target = intersectionAndUnion(pred, target,
        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

In [175…

```
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_
        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(colo
        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
    gray_folder, color_folder = './temp_grey', './temp_color'
    listImages, gt_folder = os.listdir(os.path.join(dataFolder, "testing/
    colors_path  = os.path.join(dataFolder, "colors.txt") #colors for vis
    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), (0.22
    for idx in range(0, len(listImages)):
        img = cv2.imread(os.path.join(dataFolder, "testing/image", listIm
        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 contai
```

# 7. Train the network

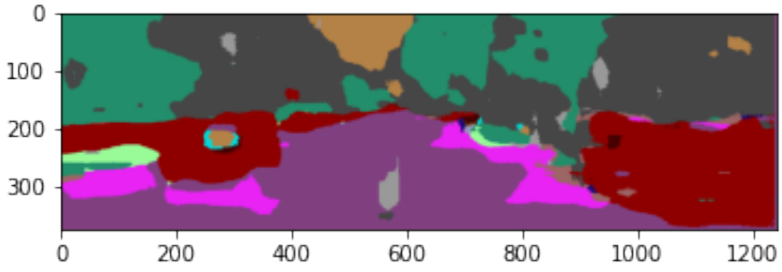
In [ ]:

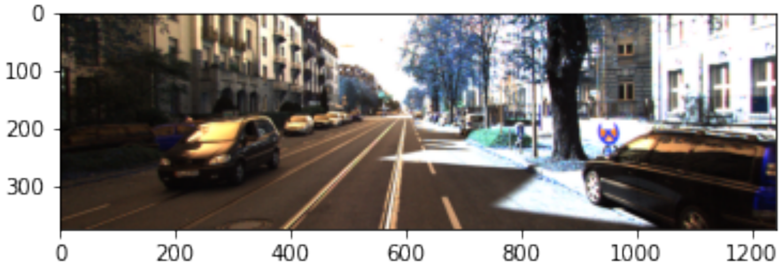
```
#The training will take ~1 h.
mIoU = 0.0
evl_each = True #Perform evaluation after each epoch. You can define the
lost = []
lost1 = 0
for epoch in range(epochs):
    for iter, (imgs, labels) in enumerate(train_loader):
        pred = segNet(imgs.to(device))
        segNet.zero_grad()
        loss = criterion(pred, labels.long().to(device)) #calculate the l
        loss.backward()
        optimizer.step()
        print('epoch {} iter {} loss={}'.format(epoch, iter, loss.data.cp
        lost1 += loss.data.cpu().numpy()
    lost1 = lost1/(iter+1)
    lost.append(lost1)
#-----Evaluation------
    if evl_each and epoch> 90:
        segNet.eval()  # The eval() must be called before evaluation
        gray_folder, color_folder = get_predictions(segNet, dataFolder, d
        segNet.train()
        temp_mIoU = cal_acc(gray_folder, os.path.join(dataFolder, 'testin
        if temp_mIoU > mIoU:
            mIoU = temp_mIoU
            torch.save(segNet.state_dict(), './model.pth')
            move_folders(gray_folder, color_folder, # Temp results -> fin
                         gray_folder.replace('temp_', ''),
                         color_folder.replace('temp_', ''))
print('The final mIoU is : {:.4f}.'.format(mIoU)) #The final mIoU is ~0.2
#Remember to download the results before closing the tab!
```

In [180…

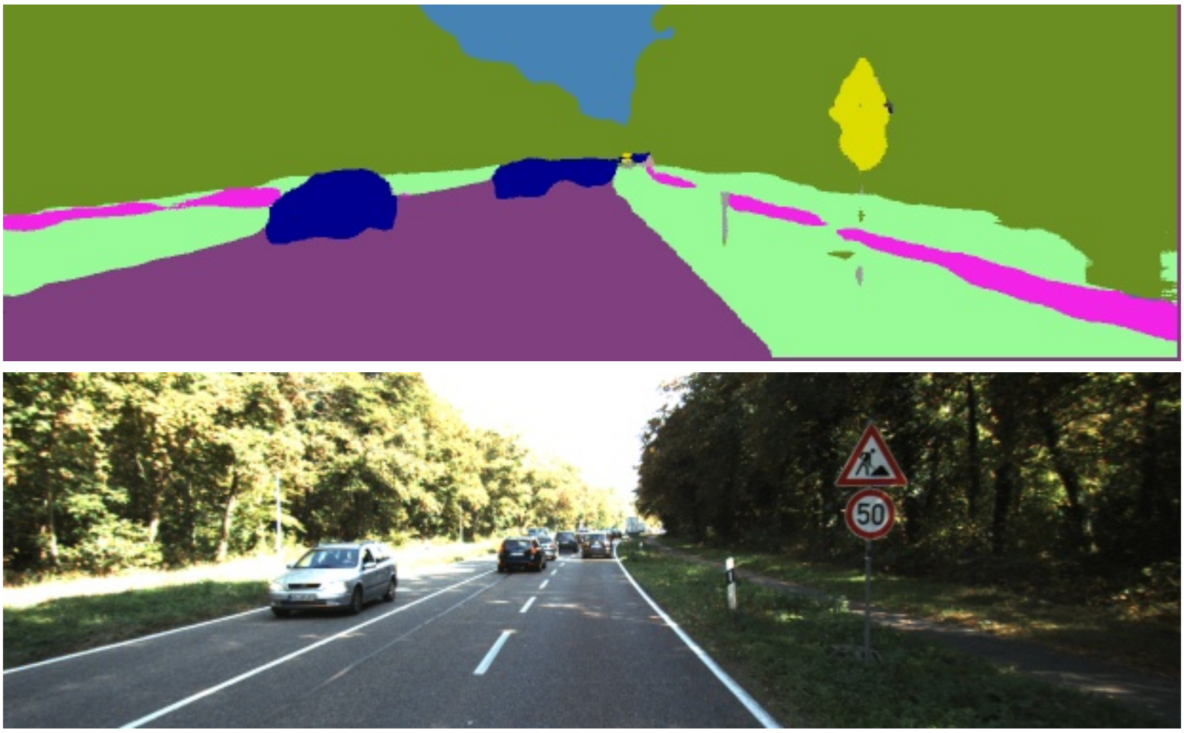
```
def prediction_show():
  img1 = cv2.imread("/content/color/000000_10.png")
  img2 = cv2.imread("/content/seg_data/testing/image/000000_10.png")
  plt.imshow(img1)
  plt.show()
  plt.imshow(img2)
  plt.show()

prediction_show()
```





A prediction example by using the baseline:



# 8. FLOPs

In [ ]:

```
#The code from https://cloudstor.aarnet.edu.au/plus/s/PcSc67ZncTSQP0E c
#Download the code.
!wget -c https://cloudstor.aarnet.edu.au/plus/s/hXo1dK9SZqiEVn9/downloa
!mv download FLOPs_counter.py
```

In [182…

```
 from FLOPs_counter import print_model_parm_flops
 input = torch.randn(1, 3, 375, 1242) # Modifying the size (3, 375, 124

 #Get the network and its FLOPs
 model = SegNetwork(n_class=19)
 print_model_parm_flops(model, input, detail=False)
```

```
+ Number of FLOPs: 158.80G
```