

# image-segmentation

December 22, 2023

Nom Prénom : BOUABD Chaimaa

```
[ ]: $pip install git+https://github.com/tensorflow/examples.git
```

```
Collecting git+https://github.com/tensorflow/examples.git
  Cloning https://github.com/tensorflow/examples.git to /tmp/pip-req-build-52varbyp
    Running command git clone --filter=blob:none --quiet
https://github.com/tensorflow/examples.git /tmp/pip-req-build-52varbyp
      Resolved https://github.com/tensorflow/examples.git to commit
0b14ce1c88537b94772cc99d995d936417be6f5d
      Preparing metadata (setup.py) ... done
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-
packages (from tensorflow-
examples==0.1699471818.63262868223049538191520590201343318150499692381) (1.4.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from tensorflow-
examples==0.1699471818.63262868223049538191520590201343318150499692381) (1.16.0)
Building wheels for collected packages: tensorflow-examples
  Building wheel for tensorflow-examples (setup.py) ... done
  Created wheel for tensorflow-examples: filename=tensorflow_examples-0.16994718
18.63262868223049538191520590201343318150499692381-py3-none-any.whl size=301553
sha256=1df718ef9495598eb9bebb0c1db952b3d67cd18eb26e6ddb0036939ef5a8012b
  Stored in directory: /tmp/pip-ephem-wheel-cache-
aezi6eix/wheels/72/5f/d0/7fe769eaa229bf20101d11a357eb23c83c481bee2d7f710599
Successfully built tensorflow-examples
Installing collected packages: tensorflow-examples
Successfully installed tensorflow-
examples-0.1699471818.63262868223049538191520590201343318150499692381
```

```
[ ]: import tensorflow as tf
```

```
import tensorflow_datasets as tfds
```

```
[ ]: from tensorflow_examples.models.pix2pix import pix2pix
```

```
from IPython.display import clear_output
import matplotlib.pyplot as plt
```

```
[ ]: ataset, info = tfds.load('oxford_iiit_pet:3.*.*', with_info=True)

Downloading and preparing dataset 773.52 MiB (download: 773.52 MiB, generated:
774.69 MiB, total: 1.51 GiB) to
/root/tensorflow_datasets/oxford_iiit_pet/3.2.0...

Dl Completed...: 0 url [00:00, ? url/s]
Dl Size...: 0 MiB [00:00, ? MiB/s]
Extraction completed...: 0 file [00:00, ? file/s]

[ ]: def normalize(input_image, input_mask):
    input_image = tf.cast(input_image, tf.float32) / 255.0
    input_mask -= 1
    return input_image, input_mask
def load_image(datapoint):
    input_image = tf.image.resize(datapoint['image'], (128, 128))
    input_mask = tf.image.resize(datapoint['segmentation_mask'], (128, 128))

    input_image, input_mask = normalize(input_image, input_mask)

    return input_image, input_mask

[ ]: TRAIN_LENGTH = info.splits['train'].num_examples
BATCH_SIZE = 64
BUFFER_SIZE = 1000
STEPS_PER_EPOCH = TRAIN_LENGTH // BATCH_SIZE
train_images = ataset['train'].map(load_image, num_parallel_calls=tf.data.
    AUTOTUNE)
test_images = ataset['test'].map(load_image, num_parallel_calls=tf.data.
    AUTOTUNE)

[ ]: class Augment(tf.keras.layers.Layer):
    def __init__(self, seed=42):
        super().__init__()
        # both use the same seed, so they'll make the same random changes.
        self.augment_inputs = tf.keras.layers.RandomFlip(mode="horizontal", ↴
            seed=seed)
        self.augment_labels = tf.keras.layers.RandomFlip(mode="horizontal", ↴
            seed=seed)

    def call(self, inputs, labels):
        inputs = self.augment_inputs(inputs)
        labels = self.augment_labels(labels)
        return inputs, labels
```

```
[ ]: train_batches = (
    train_images
    .cache()
    .shuffle(BUFFER_SIZE)
    .batch(BATCH_SIZE)
    .repeat()
    .map(Augment())
    .prefetch(buffer_size=tf.data.AUTOTUNE))

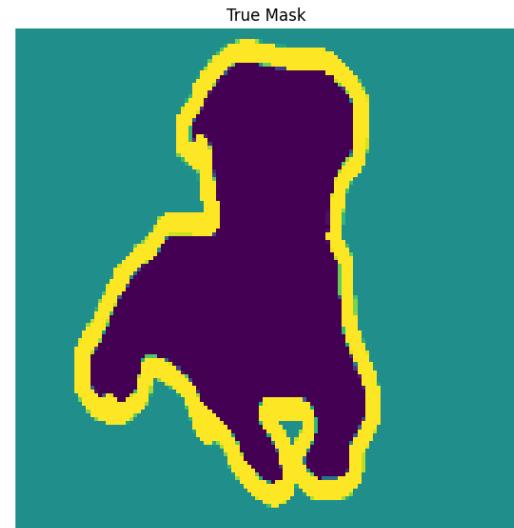
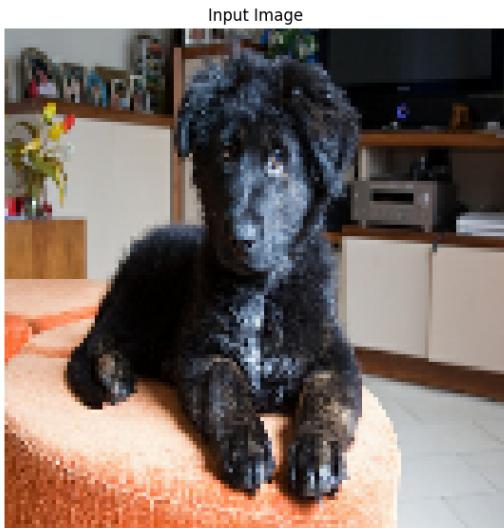
test_batches = test_images.batch(BATCH_SIZE)
```

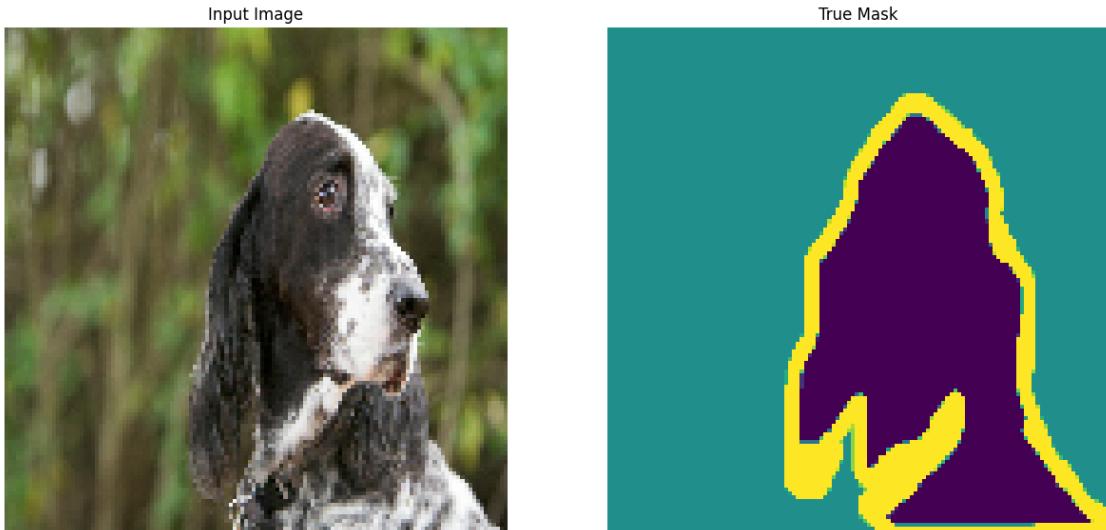
```
[ ]: def display(display_list):
    plt.figure(figsize=(15, 15))

    title = ['Input Image', 'True Mask', 'Predicted Mask']

    for i in range(len(display_list)):
        plt.subplot(1, len(display_list), i+1)
        plt.title(title[i])
        plt.imshow(tf.keras.utils.array_to_img(display_list[i]))
        plt.axis('off')
    plt.show()

for images, masks in train_batches.take(2):
    sample_image, sample_mask = images[0], masks[0]
    display([sample_image, sample_mask])
```





```
[ ]: base_model = tf.keras.applications.MobileNetV2(input_shape=[128, 128, 3],  
    ↪include_top=False)

# Use the activations of these layers  

layer_names = [  

    'block_1_expand_relu',      # 64x64  

    'block_3_expand_relu',      # 32x32  

    'block_6_expand_relu',      # 16x16  

    'block_13_expand_relu',     # 8x8  

    'block_16_project',        # 4x4
]  

base_model_outputs = [base_model.get_layer(name).output for name in layer_names]

# Create the feature extraction model  

down_stack = tf.keras.Model(inputs=base_model.input, outputs=base_model_outputs)

down_stack.trainable = False
```

Downloading data from [https://storage.googleapis.com/tensorflow/keras-applications/mobilenet\\_v2/mobilenet\\_v2\\_weights\\_tf\\_dim\\_ordering\\_tf\\_kernels\\_1.0\\_128\\_no\\_top.h5](https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_128_no_top.h5)  
9406464/9406464 [=====] - 2s 0us/step

[ ]:

[ ]:

[ ]: pip install segmentation-models-pytorch

```
Collecting segmentation-models-pytorch
  Downloading segmentation_models_pytorch-0.3.3-py3-none-any.whl (106 kB)
    106.7/106.7

kB 2.1 MB/s eta 0:00:00
Requirement already satisfied: torchvision>=0.5.0 in
/usr/local/lib/python3.10/dist-packages (from segmentation-models-pytorch)
(0.16.0+cu121)
Collecting pretrainedmodels==0.7.4 (from segmentation-models-pytorch)
  Downloading pretrainedmodels-0.7.4.tar.gz (58 kB)
    58.8/58.8 kB

6.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting efficientnet-pytorch==0.7.1 (from segmentation-models-pytorch)
  Downloading efficientnet_pytorch-0.7.1.tar.gz (21 kB)
  Preparing metadata (setup.py) ... done
Collecting timm==0.9.2 (from segmentation-models-pytorch)
  Downloading timm-0.9.2-py3-none-any.whl (2.2 MB)
    2.2/2.2 MB

11.9 MB/s eta 0:00:00
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from segmentation-models-pytorch) (4.66.1)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages
(from segmentation-models-pytorch) (9.4.0)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages
(from efficientnet-pytorch==0.7.1->segmentation-models-pytorch) (2.1.0+cu121)
Collecting munch (from pretrainedmodels==0.7.4->segmentation-models-pytorch)
  Downloading munch-4.0.0-py2.py3-none-any.whl (9.9 kB)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages
(from timm==0.9.2->segmentation-models-pytorch) (6.0.1)
Requirement already satisfied: huggingface-hub in
/usr/local/lib/python3.10/dist-packages (from timm==0.9.2->segmentation-models-
pytorch) (0.19.4)
Requirement already satisfied: safetensors in /usr/local/lib/python3.10/dist-
packages (from timm==0.9.2->segmentation-models-pytorch) (0.4.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from torchvision>=0.5.0->segmentation-models-pytorch) (1.23.5)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from torchvision>=0.5.0->segmentation-models-pytorch) (2.31.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from torch->efficientnet-pytorch==0.7.1->segmentation-models-pytorch)
(3.13.1)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.10/dist-packages (from torch->efficientnet-
pytorch==0.7.1->segmentation-models-pytorch) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages
(from torch->efficientnet-pytorch==0.7.1->segmentation-models-pytorch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-
```

```
packages (from torch->efficientnet-pytorch==0.7.1->segmentation-models-pytorch)
(3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch->efficientnet-pytorch==0.7.1->segmentation-models-pytorch) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch->efficientnet-pytorch==0.7.1->segmentation-models-pytorch)
(2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-
packages (from torch->efficientnet-pytorch==0.7.1->segmentation-models-pytorch)
(2.1.0)
Requirement already satisfied: packaging>=20.9 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub->timm==0.9.2->segmentation-models-pytorch) (23.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from
requests->torchvision>=0.5.0->segmentation-models-pytorch) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->torchvision>=0.5.0->segmentation-models-pytorch) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from
requests->torchvision>=0.5.0->segmentation-models-pytorch) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from
requests->torchvision>=0.5.0->segmentation-models-pytorch) (2023.11.17)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch->efficientnet-
pytorch==0.7.1->segmentation-models-pytorch) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-
packages (from sympy->torch->efficientnet-pytorch==0.7.1->segmentation-models-
pytorch) (1.3.0)
Building wheels for collected packages: efficientnet-pytorch, pretrainedmodels
  Building wheel for efficientnet-pytorch (setup.py) ... done
    Created wheel for efficientnet-pytorch:
filename=efficientnet_pytorch-0.7.1-py3-none-any.whl size=16428
sha256=8ad688299a16021abc9bef26c432f7bb93256b2adc804fb566996cb2cc8c679e
  Stored in directory: /root/.cache/pip/wheels/03/3f/e9/911b1bc46869644912bda90a
56bcf7b960f20b5187feea3baf
  Building wheel for pretrainedmodels (setup.py) ... done
    Created wheel for pretrainedmodels: filename=pretrainedmodels-0.7.4-py3-none-
any.whl size=60945
sha256=a09334a8cb94df61ad7d681cb34b67d6f53a387f19c5654e21bba89517a52cad
  Stored in directory: /root/.cache/pip/wheels/35/cb/a5/8f534c60142835bfc889f9a4
82e4a67e0b817032d9c6883b64
Successfully built efficientnet-pytorch pretrainedmodels
Installing collected packages: munch, efficientnet-pytorch, timm,
pretrainedmodels, segmentation-models-pytorch
Successfully installed efficientnet-pytorch-0.7.1 munch-4.0.0
pretrainedmodels-0.7.4 segmentation-models-pytorch-0.3.3 timm-0.9.2
```

```
[ ]: ##### IMPORT PACKAGES #####
%load_ext tensorboard
import os
import sys
import cv2
import random
import numpy as np
from pathlib import Path
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from PIL import Image
from google.colab.patches import cv2_imshow
import sklearn
from sklearn import model_selection
from sklearn import metrics

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torch.utils.data import DataLoader
import torch.nn as nn
from collections import defaultdict
import torchvision
from torchvision import transforms
import torch.nn.functional as F
from tqdm.auto import tqdm
from torch.utils.tensorboard import SummaryWriter
import copy
import time

import segmentation_models_pytorch

import albumentations as A
#from albumentations.pytorch import ToTensorV2
from albumentations.pytorch.transforms import ToTensor
```

```
[ ]: from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
[ ]: !ls /content/drive/MyDrive/LP_MI/FloodNet_Challenge/Train/Labeled
```

Flooded Non-Flooded

```
[ ]: !ls /content/drive/MyDrive/LP_MI/FloodNet_Challenge/Train/Labeled/Flooded
```

```
image mask
[ ]: RESIZE=(512,512)
temp_root = "/content/drive/MyDrive/LP_MI/FloodNet_Challenge"
local_root = "/content/drive/MyDrive/LP_MI/FloodNet_Challenge"
def resize_and_save(path, resize=RESIZE, samples='all'):
    if len(os.listdir(os.path.join(local_root, path))) == 0:
        print(f"{path} --> Saving...\n")
    if samples == 'all':
        samples = len(os.listdir(os.path.join(temp_root, path)))
    for img_name in tqdm(os.listdir(os.path.join(temp_root, path))[:samples]):
        img = cv2.imread(os.path.join(temp_root, path, img_name))
        img = cv2.resize(img, RESIZE)
        cv2.imwrite(os.path.join(local_root, path, img_name), img)
    else:
        print(f"{path} --> images are already saved")
```

```
[ ]: !ls
```

```
drive sample_data
[ ]: resize_and_save("Train/Labeled/Flooded/image")
resize_and_save("Train/Labeled/Non-Flooded/image")
resize_and_save("Train/Labeled/Flooded/mask")
resize_and_save("Train/Labeled/Non-Flooded/mask")
resize_and_save("Train/Unlabeled/image")
resize_and_save("Test/image")
resize_and_save("Train/Unlabeled/image", samples=100)
```

```
Train/Labeled/Flooded/image --> images are already saved
Train/Labeled/Non-Flooded/image --> images are already saved
Train/Labeled/Flooded/mask --> images are already saved
Train/Labeled/Non-Flooded/mask --> images are already saved
Train/Unlabeled/image --> images are already saved
Test/image --> images are already saved
Train/Unlabeled/image --> images are already saved
```

```
[ ]: flood_dir = "Train/Labeled/Flooded"
non_flood_dir = "Train/Labeled/Non-Flooded"

f_img_dir = os.path.join(local_root, flood_dir, "image")
f_mask_dir = os.path.join(local_root, flood_dir, "mask")
n_img_dir = os.path.join(local_root, non_flood_dir, "image")
n_mask_dir = os.path.join(local_root, non_flood_dir, "mask")

f_img = len(os.listdir(f_img_dir))
f_mask = len(os.listdir(f_mask_dir))
n_img = len(os.listdir(n_img_dir))
```

```

n_mask = len(os.listdir(n_mask_dir))
print(f"Flooded images: {f_img} Flooded masks: {f_mask}")
print(f"Non-Flooded images: {n_img} Non-Flooded masks: {n_mask}")

```

Flooded images: 51 Flooded masks: 51  
 Non-Flooded images: 347 Non-Flooded masks: 347

```

[ ]: sample_f_img = cv2.imread(os.path.join(f_img_dir, sorted(os.
   .listdir(f_img_dir))[0]))
sample_f_mask = cv2.imread(os.path.join(f_mask_dir, sorted(os.
   .listdir(f_mask_dir))[0]))
sample_n_img = cv2.imread(os.path.join(n_img_dir, sorted(os.
   .listdir(n_img_dir))[0]))
sample_n_mask = cv2.imread(os.path.join(n_mask_dir, sorted(os.
   .listdir(n_mask_dir))[0)))

print(np.min(sample_f_mask),
np.max(sample_f_mask),
np.min(sample_n_mask),
np.max(sample_n_mask))

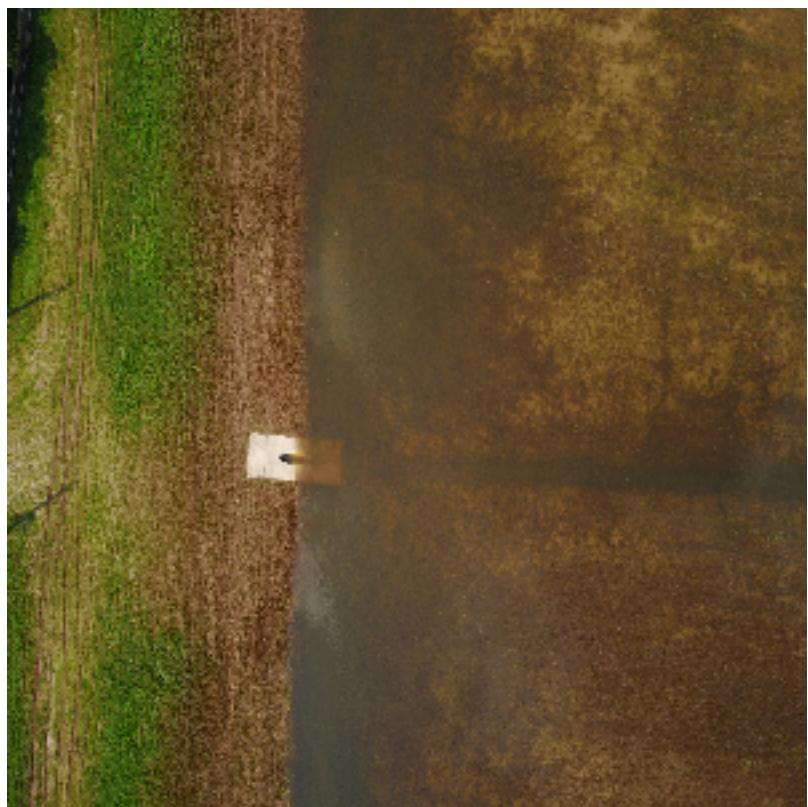
sample_f_mask = sample_f_mask * (255/9)
sample_n_mask = sample_n_mask * (255/9)

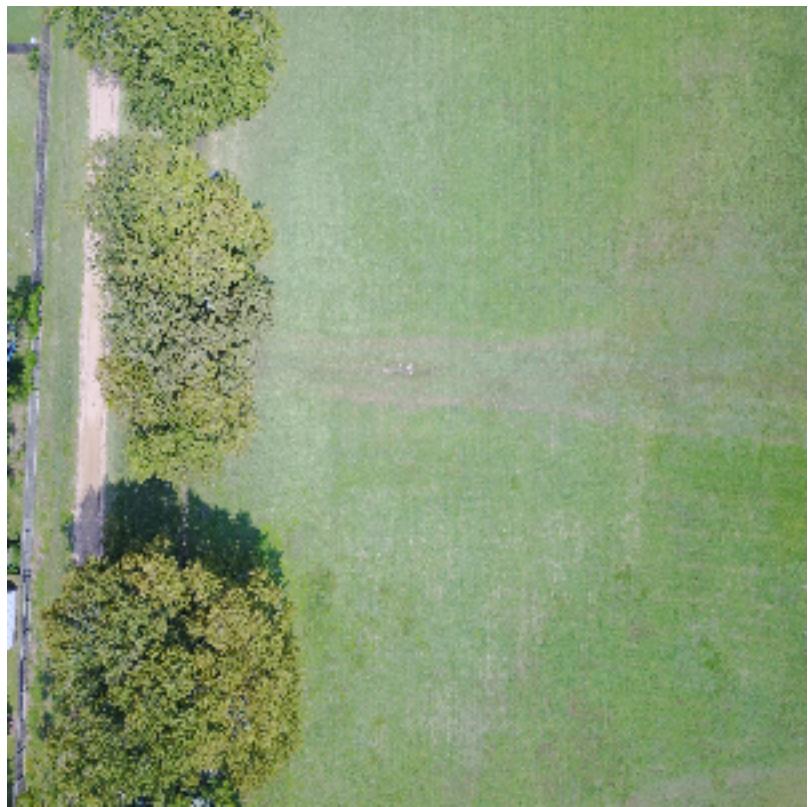
print(np.min(sample_f_mask),
np.max(sample_f_mask),
np.min(sample_n_mask),
np.max(sample_n_mask))

cv2_imshow(cv2.resize(sample_f_img, dsize=(300, 300)))
cv2_imshow(cv2.resize(sample_f_mask, dsize=(300, 300)))
cv2_imshow(cv2.resize(sample_n_img, dsize=(300, 300)))
cv2_imshow(cv2.resize(sample_n_mask, dsize=(300, 300)))

```

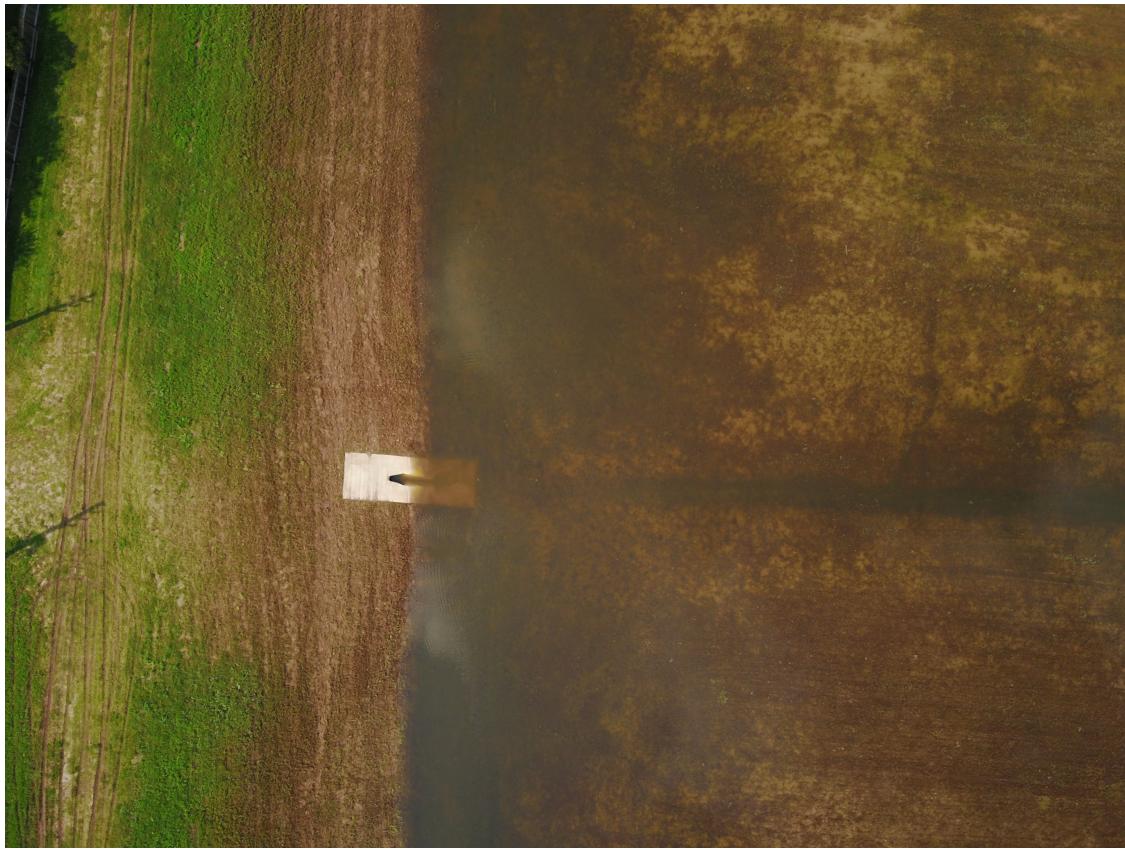
5 5 2 9  
 141.66666666666666 141.66666666666666 56.66666666666664 255.0







```
[ ]: kernel = np.ones((2,2),np.uint8)
cv2_imshow(cv2.erode(cv2.dilate(cv2.bilateralFilter(sample_f_img, 5, 75, 75),kernel, iterations=2), kernel, iterations=1))
```



[ ]:

10165.jpg	6527.jpg	6798.jpg	7078.jpg	7340.jpg	7719.jpg	7966.jpg	8216.jpg
8470.jpg	8960.jpg						
10166.jpg	6528.jpg	6799.jpg	7079.jpg	7344.jpg	7720.jpg	7967.jpg	8217.jpg
8472.jpg	8962.jpg						
10168.jpg	6530.jpg	6800.jpg	7081.jpg	7345.jpg	7721.jpg	7971.jpg	8221.jpg
8473.jpg	8963.jpg						
10170.jpg	6531.jpg	6802.jpg	7082.jpg	7353.jpg	7722.jpg	7972.jpg	8222.jpg
8475.jpg	8965.jpg						
10171.jpg	6533.jpg	6803.jpg	7083.jpg	7354.jpg	7724.jpg	7973.jpg	8223.jpg
8477.jpg	8966.jpg						
10172.jpg	6534.jpg	6804.jpg	7086.jpg	7356.jpg	7725.jpg	7974.jpg	8224.jpg
8479.jpg	8967.jpg						
10175.jpg	6535.jpg	6805.jpg	7087.jpg	7357.jpg	7726.jpg	7975.jpg	8225.jpg
8480.jpg	8968.jpg						
10176.jpg	6537.jpg	6806.jpg	7088.jpg	7362.jpg	7727.jpg	7977.jpg	8226.jpg
8481.jpg	8969.jpg						
10179.jpg	6538.jpg	6807.jpg	7090.jpg	7363.jpg	7728.jpg	7979.jpg	8227.jpg
8482.jpg	8970.jpg						
10180.jpg	6540.jpg	6808.jpg	7091.jpg	7364.jpg	7729.jpg	7980.jpg	8228.jpg

8483.jpg	8971.jpg							
10181.jpg	6541.jpg	6810.jpg	7094.jpg	7366.jpg	7731.jpg	7981.jpg	8229.jpg	
8484.jpg	8973.jpg							
10182.jpg	6542.jpg	6812.jpg	7096.jpg	7372.jpg	7735.jpg	7982.jpg	8230.jpg	
8485.jpg	8974.jpg							
10184.jpg	6543.jpg	6813.jpg	7097.jpg	7374.jpg	7736.jpg	7987.jpg	8231.jpg	
8487.jpg	8975.jpg							
10300.jpg	6544.jpg	6814.jpg	7098.jpg	7375.jpg	7737.jpg	7989.jpg	8232.jpg	
8488.jpg	8977.jpg							
10566.jpg	6546.jpg	6818.jpg	7100.jpg	7377.jpg	7739.jpg	7996.jpg	8234.jpg	
8490.jpg	8978.jpg							
10687.jpg	6548.jpg	6819.jpg	7102.jpg	7405.jpg	7746.jpg	7998.jpg	8235.jpg	
8491.jpg	8980.jpg							
10806.jpg	6549.jpg	6820.jpg	7103.jpg	7410.jpg	7748.jpg	7999.jpg	8236.jpg	
8492.jpg	8982.jpg							
10810.jpg	6552.jpg	6824.jpg	7107.jpg	7412.jpg	7749.jpg	8000.jpg	8240.jpg	
8493.jpg	8984.jpg							
10811.jpg	6555.jpg	6826.jpg	7108.jpg	7414.jpg	7750.jpg	8001.jpg	8241.jpg	
8496.jpg	8986.jpg							
10817.jpg	6556.jpg	6829.jpg	7109.jpg	7418.jpg	7751.jpg	8002.jpg	8242.jpg	
8497.jpg	8987.jpg							
10818.jpg	6559.jpg	6830.jpg	7110.jpg	7422.jpg	7752.jpg	8003.jpg	8243.jpg	
8498.jpg	8994.jpg							
10819.jpg	6560.jpg	6832.jpg	7115.jpg	7426.jpg	7753.jpg	8005.jpg	8244.jpg	
8500.jpg	8995.jpg							
10820.jpg	6563.jpg	6833.jpg	7118.jpg	7427.jpg	7754.jpg	8008.jpg	8245.jpg	
8501.jpg	8996.jpg							
10821.jpg	6564.jpg	6836.jpg	7121.jpg	7428.jpg	7756.jpg	8009.jpg	8247.jpg	
8502.jpg	8998.jpg							
10825.jpg	6565.jpg	6838.jpg	7122.jpg	7429.jpg	7760.jpg	8010.jpg	8248.jpg	
8504.jpg	9000.jpg							
10826.jpg	6566.jpg	6839.jpg	7123.jpg	7430.jpg	7761.jpg	8011.jpg	8249.jpg	
8506.jpg	9003.jpg							
10827.jpg	6569.jpg	6842.jpg	7124.jpg	7434.jpg	7764.jpg	8013.jpg	8251.jpg	
8507.jpg	9007.jpg							
10828.jpg	6573.jpg	6843.jpg	7125.jpg	7436.jpg	7765.jpg	8014.jpg	8252.jpg	
8508.jpg	9008.jpg							
10834.jpg	6574.jpg	6846.jpg	7127.jpg	7438.jpg	7766.jpg	8015.jpg	8253.jpg	
8510.jpg	9010.jpg							
10835.jpg	6580.jpg	6849.jpg	7128.jpg	7439.jpg	7767.jpg	8016.jpg	8255.jpg	
8512.jpg	9011.jpg							
10836.jpg	6584.jpg	6851.jpg	7130.jpg	7440.jpg	7768.jpg	8020.jpg	8256.jpg	
8515.jpg	9015.jpg							
10837.jpg	6585.jpg	6852.jpg	7131.jpg	7445.jpg	7770.jpg	8024.jpg	8260.jpg	
8516.jpg	9017.jpg							
10840.jpg	6586.jpg	6853.jpg	7132.jpg	7448.jpg	7774.jpg	8029.jpg	8261.jpg	
8517.jpg	9019.jpg							
10841.jpg	6588.jpg	6854.jpg	7136.jpg	7449.jpg	7775.jpg	8030.jpg	8264.jpg	

8520.jpg	9022.jpg							
11723.jpg	6589.jpg	6855.jpg	7142.jpg	7451.jpg	7776.jpg	8032.jpg	8265.jpg	
8521.jpg	9023.jpg							
6279.jpg	6591.jpg	6857.jpg	7144.jpg	7452.jpg	7777.jpg	8033.jpg	8267.jpg	
8522.jpg	9025.jpg							
6287.jpg	6592.jpg	6858.jpg	7146.jpg	7453.jpg	7778.jpg	8034.jpg	8269.jpg	
8524.jpg	9026.jpg							
6332.jpg	6595.jpg	6859.jpg	7147.jpg	7454.jpg	7782.jpg	8037.jpg	8271.jpg	
8526.jpg	9027.jpg							
6334.jpg	6597.jpg	6860.jpg	7151.jpg	7458.jpg	7783.jpg	8038.jpg	8272.jpg	
8527.jpg	9028.jpg							
6335.jpg	6600.jpg	6890.jpg	7156.jpg	7459.jpg	7784.jpg	8039.jpg	8274.jpg	
8528.jpg	9029.jpg							
6338.jpg	6601.jpg	6891.jpg	7157.jpg	7460.jpg	7786.jpg	8042.jpg	8275.jpg	
8529.jpg	9030.jpg							
6340.jpg	6603.jpg	6899.jpg	7160.jpg	7462.jpg	7787.jpg	8043.jpg	8276.jpg	
8531.jpg	9033.jpg							
6341.jpg	6608.jpg	6900.jpg	7161.jpg	7463.jpg	7788.jpg	8044.jpg	8277.jpg	
8535.jpg	9034.jpg							
6344.jpg	6610.jpg	6901.jpg	7162.jpg	7465.jpg	7789.jpg	8045.jpg	8278.jpg	
8536.jpg	9036.jpg							
6346.jpg	6613.jpg	6902.jpg	7163.jpg	7466.jpg	7791.jpg	8046.jpg	8280.jpg	
8538.jpg	9039.jpg							
6347.jpg	6614.jpg	6904.jpg	7164.jpg	7467.jpg	7792.jpg	8047.jpg	8281.jpg	
8539.jpg	9040.jpg							
6348.jpg	6615.jpg	6908.jpg	7165.jpg	7469.jpg	7795.jpg	8049.jpg	8282.jpg	
8540.jpg	9043.jpg							
6350.jpg	6616.jpg	6909.jpg	7167.jpg	7472.jpg	7796.jpg	8051.jpg	8284.jpg	
8545.jpg	9047.jpg							
6351.jpg	6617.jpg	6911.jpg	7168.jpg	7473.jpg	7798.jpg	8052.jpg	8286.jpg	
8546.jpg	9049.jpg							
6352.jpg	6618.jpg	6913.jpg	7170.jpg	7474.jpg	7800.jpg	8053.jpg	8288.jpg	
8547.jpg	9050.jpg							
6354.jpg	6620.jpg	6914.jpg	7171.jpg	7475.jpg	7801.jpg	8055.jpg	8294.jpg	
8548.jpg	9052.jpg							
6358.jpg	6622.jpg	6918.jpg	7174.jpg	7477.jpg	7802.jpg	8056.jpg	8297.jpg	
8550.jpg	9053.jpg							
6359.jpg	6623.jpg	6919.jpg	7175.jpg	7481.jpg	7806.jpg	8057.jpg	8298.jpg	
8551.jpg	9059.jpg							
6360.jpg	6625.jpg	6922.jpg	7176.jpg	7485.jpg	7807.jpg	8059.jpg	8299.jpg	
8552.jpg	9061.jpg							
6361.jpg	6627.jpg	6924.jpg	7177.jpg	7487.jpg	7808.jpg	8060.jpg	8304.jpg	
8553.jpg	9062.jpg							
6363.jpg	6628.jpg	6926.jpg	7178.jpg	7488.jpg	7809.jpg	8061.jpg	8305.jpg	
8555.jpg	9063.jpg							
6364.jpg	6633.jpg	6928.jpg	7179.jpg	7489.jpg	7810.jpg	8063.jpg	8306.jpg	
8556.jpg	9064.jpg							
6365.jpg	6635.jpg	6929.jpg	7181.jpg	7520.jpg	7814.jpg	8065.jpg	8311.jpg	

8560.jpg	9065.jpg							
6367.jpg	6639.jpg	6930.jpg	7182.jpg	7521.jpg	7815.jpg	8068.jpg	8312.jpg	
8586.jpg	9066.jpg							
6368.jpg	6640.jpg	6931.jpg	7183.jpg	7522.jpg	7816.jpg	8070.jpg	8313.jpg	
8587.jpg	9069.jpg							
6369.jpg	6643.jpg	6933.jpg	7184.jpg	7523.jpg	7818.jpg	8072.jpg	8315.jpg	
8588.jpg	9070.jpg							
6370.jpg	6644.jpg	6934.jpg	7185.jpg	7524.jpg	7820.jpg	8074.jpg	8317.jpg	
8589.jpg	9071.jpg							
6372.jpg	6645.jpg	6935.jpg	7186.jpg	7525.jpg	7825.jpg	8075.jpg	8318.jpg	
8590.jpg	9072.jpg							
6373.jpg	6646.jpg	6936.jpg	7187.jpg	7526.jpg	7827.jpg	8077.jpg	8319.jpg	
8593.jpg	9073.jpg							
6374.jpg	6647.jpg	6938.jpg	7188.jpg	7542.jpg	7828.jpg	8080.jpg	8320.jpg	
8594.jpg	9074.jpg							
6375.jpg	6648.jpg	6939.jpg	7189.jpg	7545.jpg	7829.jpg	8081.jpg	8321.jpg	
8597.jpg	9075.jpg							
6376.jpg	6649.jpg	6940.jpg	7192.jpg	7546.jpg	7830.jpg	8082.jpg	8322.jpg	
8598.jpg	9076.jpg							
6379.jpg	6651.jpg	6941.jpg	7193.jpg	7547.jpg	7831.jpg	8083.jpg	8326.jpg	
8599.jpg	9077.jpg							
6381.jpg	6652.jpg	6943.jpg	7194.jpg	7548.jpg	7832.jpg	8085.jpg	8330.jpg	
8600.jpg	9078.jpg							
6384.jpg	6654.jpg	6945.jpg	7195.jpg	7552.jpg	7833.jpg	8086.jpg	8334.jpg	
8603.jpg	9079.jpg							
6385.jpg	6655.jpg	6946.jpg	7199.jpg	7556.jpg	7834.jpg	8087.jpg	8335.jpg	
8604.jpg	9080.jpg							
6386.jpg	6656.jpg	6947.jpg	7201.jpg	7557.jpg	7837.jpg	8089.jpg	8336.jpg	
8605.jpg	9081.jpg							
6387.jpg	6657.jpg	6953.jpg	7202.jpg	7559.jpg	7841.jpg	8090.jpg	8342.jpg	
8615.jpg	9083.jpg							
6388.jpg	6658.jpg	6955.jpg	7205.jpg	7560.jpg	7842.jpg	8091.jpg	8344.jpg	
8638.jpg	9084.jpg							
6393.jpg	6660.jpg	6956.jpg	7209.jpg	7566.jpg	7843.jpg	8093.jpg	8345.jpg	
8658.jpg	9085.jpg							
6395.jpg	6662.jpg	6957.jpg	7210.jpg	7567.jpg	7844.jpg	8094.jpg	8347.jpg	
8659.jpg	9086.jpg							
6397.jpg	6664.jpg	6960.jpg	7211.jpg	7568.jpg	7845.jpg	8095.jpg	8348.jpg	
8770.jpg	9087.jpg							
6398.jpg	6666.jpg	6964.jpg	7212.jpg	7571.jpg	7847.jpg	8096.jpg	8350.jpg	
8774.jpg	9088.jpg							
6399.jpg	6667.jpg	6965.jpg	7214.jpg	7573.jpg	7848.jpg	8098.jpg	8351.jpg	
8775.jpg	9089.jpg							
6402.jpg	6668.jpg	6966.jpg	7219.jpg	7574.jpg	7849.jpg	8099.jpg	8352.jpg	
8777.jpg	9090.jpg							
6403.jpg	6670.jpg	6967.jpg	7221.jpg	7575.jpg	7852.jpg	8101.jpg	8354.jpg	
8783.jpg	9091.jpg							
6404.jpg	6673.jpg	6968.jpg	7222.jpg	7579.jpg	7855.jpg	8102.jpg	8355.jpg	

8785.jpg	9093.jpg							
6407.jpg	6675.jpg	6969.jpg	7225.jpg	7580.jpg	7857.jpg	8106.jpg	8357.jpg	
8786.jpg	9094.jpg							
6408.jpg	6676.jpg	6970.jpg	7226.jpg	7582.jpg	7858.jpg	8107.jpg	8359.jpg	
8791.jpg	9095.jpg							
6409.jpg	6678.jpg	6971.jpg	7227.jpg	7585.jpg	7859.jpg	8108.jpg	8361.jpg	
8794.jpg	9098.jpg							
6411.jpg	6681.jpg	6972.jpg	7229.jpg	7587.jpg	7861.jpg	8109.jpg	8366.jpg	
8796.jpg	9099.jpg							
6413.jpg	6682.jpg	6974.jpg	7230.jpg	7589.jpg	7862.jpg	8110.jpg	8368.jpg	
8798.jpg	9100.jpg							
6414.jpg	6685.jpg	6976.jpg	7235.jpg	7594.jpg	7864.jpg	8116.jpg	8369.jpg	
8799.jpg	9101.jpg							
6415.jpg	6686.jpg	6978.jpg	7236.jpg	7596.jpg	7865.jpg	8117.jpg	8370.jpg	
8800.jpg	9103.jpg							
6416.jpg	6687.jpg	6981.jpg	7238.jpg	7597.jpg	7867.jpg	8118.jpg	8371.jpg	
8801.jpg	9104.jpg							
6421.jpg	6689.jpg	6982.jpg	7239.jpg	7600.jpg	7868.jpg	8119.jpg	8374.jpg	
8802.jpg	9105.jpg							
6422.jpg	6692.jpg	6985.jpg	7240.jpg	7601.jpg	7869.jpg	8121.jpg	8375.jpg	
8803.jpg	9106.jpg							
6424.jpg	6693.jpg	6986.jpg	7242.jpg	7602.jpg	7871.jpg	8122.jpg	8379.jpg	
8804.jpg	9108.jpg							
6426.jpg	6701.jpg	6992.jpg	7243.jpg	7603.jpg	7872.jpg	8129.jpg	8380.jpg	
8805.jpg	9110.jpg							
6427.jpg	6703.jpg	6993.jpg	7244.jpg	7606.jpg	7873.jpg	8130.jpg	8381.jpg	
8807.jpg	9115.jpg							
6430.jpg	6704.jpg	6994.jpg	7245.jpg	7609.jpg	7875.jpg	8131.jpg	8382.jpg	
8809.jpg	9116.jpg							
6431.jpg	6706.jpg	6995.jpg	7247.jpg	7612.jpg	7878.jpg	8132.jpg	8385.jpg	
8816.jpg	9144.jpg							
6433.jpg	6707.jpg	6996.jpg	7251.jpg	7614.jpg	7879.jpg	8133.jpg	8386.jpg	
8817.jpg	9162.jpg							
6435.jpg	6708.jpg	6997.jpg	7252.jpg	7641.jpg	7882.jpg	8134.jpg	8387.jpg	
8818.jpg	9278.jpg							
6437.jpg	6710.jpg	6998.jpg	7256.jpg	7643.jpg	7883.jpg	8135.jpg	8388.jpg	
8820.jpg	9287.jpg							
6438.jpg	6711.jpg	6999.jpg	7257.jpg	7645.jpg	7884.jpg	8137.jpg	8389.jpg	
8821.jpg	9343.jpg							
6442.jpg	6712.jpg	7000.jpg	7258.jpg	7646.jpg	7890.jpg	8138.jpg	8390.jpg	
8822.jpg	9345.jpg							
6443.jpg	6713.jpg	7003.jpg	7261.jpg	7648.jpg	7892.jpg	8139.jpg	8392.jpg	
8823.jpg	9346.jpg							
6444.jpg	6714.jpg	7006.jpg	7263.jpg	7650.jpg	7893.jpg	8141.jpg	8393.jpg	
8833.jpg	9349.jpg							
6446.jpg	6715.jpg	7007.jpg	7264.jpg	7653.jpg	7894.jpg	8142.jpg	8394.jpg	
8859.jpg	9351.jpg							
6447.jpg	6716.jpg	7008.jpg	7265.jpg	7654.jpg	7895.jpg	8143.jpg	8398.jpg	

8879.jpg	9353.jpg							
6450.jpg	6719.jpg	7009.jpg	7266.jpg	7655.jpg	7896.jpg	8145.jpg	8400.jpg	
8886.jpg	9365.jpg							
6451.jpg	6727.jpg	7010.jpg	7267.jpg	7659.jpg	7901.jpg	8146.jpg	8403.jpg	
8895.jpg	9366.jpg							
6453.jpg	6728.jpg	7011.jpg	7270.jpg	7661.jpg	7902.jpg	8148.jpg	8404.jpg	
8900.jpg	9459.jpg							
6454.jpg	6731.jpg	7014.jpg	7271.jpg	7662.jpg	7905.jpg	8149.jpg	8405.jpg	
8901.jpg	9511.jpg							
6455.jpg	6732.jpg	7015.jpg	7272.jpg	7664.jpg	7907.jpg	8150.jpg	8406.jpg	
8904.jpg	9513.jpg							
6456.jpg	6734.jpg	7017.jpg	7273.jpg	7665.jpg	7908.jpg	8151.jpg	8407.jpg	
8905.jpg	9514.jpg							
6457.jpg	6735.jpg	7018.jpg	7278.jpg	7671.jpg	7909.jpg	8152.jpg	8408.jpg	
8906.jpg	9538.jpg							
6458.jpg	6736.jpg	7021.jpg	7282.jpg	7672.jpg	7910.jpg	8153.jpg	8409.jpg	
8907.jpg	9546.jpg							
6459.jpg	6737.jpg	7022.jpg	7285.jpg	7673.jpg	7914.jpg	8157.jpg	8410.jpg	
8909.jpg	9558.jpg							
6460.jpg	6738.jpg	7024.jpg	7287.jpg	7674.jpg	7916.jpg	8158.jpg	8411.jpg	
8910.jpg	9718.jpg							
6461.jpg	6739.jpg	7025.jpg	7288.jpg	7675.jpg	7917.jpg	8159.jpg	8412.jpg	
8911.jpg	9722.jpg							
6462.jpg	6742.jpg	7026.jpg	7289.jpg	7677.jpg	7918.jpg	8161.jpg	8413.jpg	
8914.jpg	9723.jpg							
6463.jpg	6743.jpg	7027.jpg	7290.jpg	7678.jpg	7919.jpg	8164.jpg	8415.jpg	
8915.jpg	9724.jpg							
6464.jpg	6745.jpg	7028.jpg	7292.jpg	7680.jpg	7920.jpg	8165.jpg	8416.jpg	
8916.jpg	9725.jpg							
6465.jpg	6746.jpg	7033.jpg	7293.jpg	7681.jpg	7922.jpg	8166.jpg	8417.jpg	
8917.jpg	9795.jpg							
6466.jpg	6747.jpg	7036.jpg	7298.jpg	7682.jpg	7923.jpg	8167.jpg	8418.jpg	
8918.jpg	9796.jpg							
6469.jpg	6748.jpg	7037.jpg	7299.jpg	7685.jpg	7924.jpg	8168.jpg	8419.jpg	
8921.jpg	9797.jpg							
6477.jpg	6749.jpg	7038.jpg	7304.jpg	7686.jpg	7925.jpg	8172.jpg	8420.jpg	
8922.jpg	9874.jpg							
6478.jpg	6750.jpg	7039.jpg	7305.jpg	7688.jpg	7929.jpg	8173.jpg	8422.jpg	
8923.jpg	9875.jpg							
6481.jpg	6754.jpg	7043.jpg	7306.jpg	7691.jpg	7931.jpg	8174.jpg	8423.jpg	
8924.jpg	9876.jpg							
6484.jpg	6755.jpg	7044.jpg	7310.jpg	7692.jpg	7932.jpg	8175.jpg	8424.jpg	
8925.jpg	9877.jpg							
6486.jpg	6756.jpg	7045.jpg	7311.jpg	7693.jpg	7933.jpg	8176.jpg	8425.jpg	
8927.jpg	9879.jpg							
6492.jpg	6760.jpg	7046.jpg	7312.jpg	7696.jpg	7934.jpg	8177.jpg	8426.jpg	
8928.jpg	9880.jpg							
6496.jpg	6762.jpg	7047.jpg	7313.jpg	7697.jpg	7939.jpg	8182.jpg	8427.jpg	

8929.jpg	9884.jpg							
6498.jpg	6763.jpg	7049.jpg	7314.jpg	7700.jpg	7944.jpg	8184.jpg	8429.jpg	
8931.jpg	9885.jpg							
6499.jpg	6764.jpg	7050.jpg	7316.jpg	7702.jpg	7945.jpg	8185.jpg	8430.jpg	
8932.jpg	9887.jpg							
6501.jpg	6770.jpg	7051.jpg	7317.jpg	7703.jpg	7947.jpg	8190.jpg	8431.jpg	
8933.jpg	9890.jpg							
6504.jpg	6772.jpg	7053.jpg	7318.jpg	7704.jpg	7948.jpg	8192.jpg	8432.jpg	
8935.jpg	9892.jpg							
6505.jpg	6773.jpg	7055.jpg	7319.jpg	7705.jpg	7949.jpg	8193.jpg	8433.jpg	
8938.jpg	9894.jpg							
6506.jpg	6776.jpg	7059.jpg	7321.jpg	7706.jpg	7950.jpg	8196.jpg	8436.jpg	
8940.jpg	9895.jpg							
6509.jpg	6779.jpg	7061.jpg	7322.jpg	7707.jpg	7951.jpg	8197.jpg	8437.jpg	
8943.jpg	9896.jpg							
6510.jpg	6783.jpg	7064.jpg	7324.jpg	7709.jpg	7952.jpg	8200.jpg	8438.jpg	
8947.jpg	9897.jpg							
6512.jpg	6787.jpg	7065.jpg	7325.jpg	7711.jpg	7953.jpg	8202.jpg	8442.jpg	
8951.jpg	9898.jpg							
6513.jpg	6788.jpg	7070.jpg	7328.jpg	7712.jpg	7954.jpg	8205.jpg	8443.jpg	
8952.jpg	9925.jpg							
6516.jpg	6789.jpg	7071.jpg	7330.jpg	7713.jpg	7958.jpg	8206.jpg	8445.jpg	
8953.jpg								
6520.jpg	6790.jpg	7072.jpg	7331.jpg	7714.jpg	7959.jpg	8207.jpg	8463.jpg	
8955.jpg								
6523.jpg	6792.jpg	7073.jpg	7332.jpg	7715.jpg	7962.jpg	8208.jpg	8464.jpg	
8956.jpg								
6525.jpg	6793.jpg	7075.jpg	7335.jpg	7717.jpg	7963.jpg	8210.jpg	8465.jpg	
8957.jpg								
6526.jpg	6797.jpg	7077.jpg	7336.jpg	7718.jpg	7965.jpg	8212.jpg	8469.jpg	
8958.jpg								

```
[ ]: RESIZE=(512,512)
temp_root = "/content/drive/MyDrive/LP_MI/Resize_img"
local_root = "/content/drive/MyDrive/LP_MI/FloodNet-Supervised_v1.0"
def resize_and_save(path, resize=RESIZE, samples='all'):
    if len(os.listdir(os.path.join(local_root, path))) == 0:
        print(f"{path} --> Saving...\n")
        if samples == 'all':
            samples = len(os.listdir(os.path.join(temp_root, path)))
        for img_name in tqdm(os.listdir(os.path.join(temp_root, path))[:samples]):
            img = cv2.imread(os.path.join(temp_root, path, img_name))
            img = cv2.resize(img, RESIZE)
            cv2.imwrite(os.path.join(local_root, path, img_name), img)
    else:
        print(f"{path} --> images are already saved")
```

```
[ ]: !ls
```

```
drive sample_data
```

```
[ ]: resize_and_save("train/train-org-img")
resize_and_save("train/train-label-img")
resize_and_save("val/val-label-img")
resize_and_save("val/val-org-img")
resize_and_save("test/test-label-img")
resize_and_save("test/test-org-img")
```

train/train-org-img --> images are already saved  
 train/train-label-img --> images are already saved  
 val/val-label-img --> images are already saved  
 val/val-org-img --> images are already saved  
 test/test-label-img --> images are already saved  
 test/test-org-img --> images are already saved

```
[ ]: flood_dir = "train/train-org-img"
non_flood_dir = "val/val-org-img"

f_img_dir = os.path.join(local_root, flood_dir)
f_mask_dir = os.path.join(local_root, flood_dir,)
n_img_dir = os.path.join(local_root, non_flood_dir)
n_mask_dir = os.path.join(local_root, non_flood_dir)

f_img = len(os.listdir(f_img_dir))
f_mask = len(os.listdir(f_mask_dir))
n_img = len(os.listdir(n_img_dir))
n_mask = len(os.listdir(n_mask_dir))
print(f"Flooded images: {f_img} Flooded masks: {f_mask}")
print(f"Non-Flooded images: {n_img} Non-Flooded masks: {n_mask}")
```

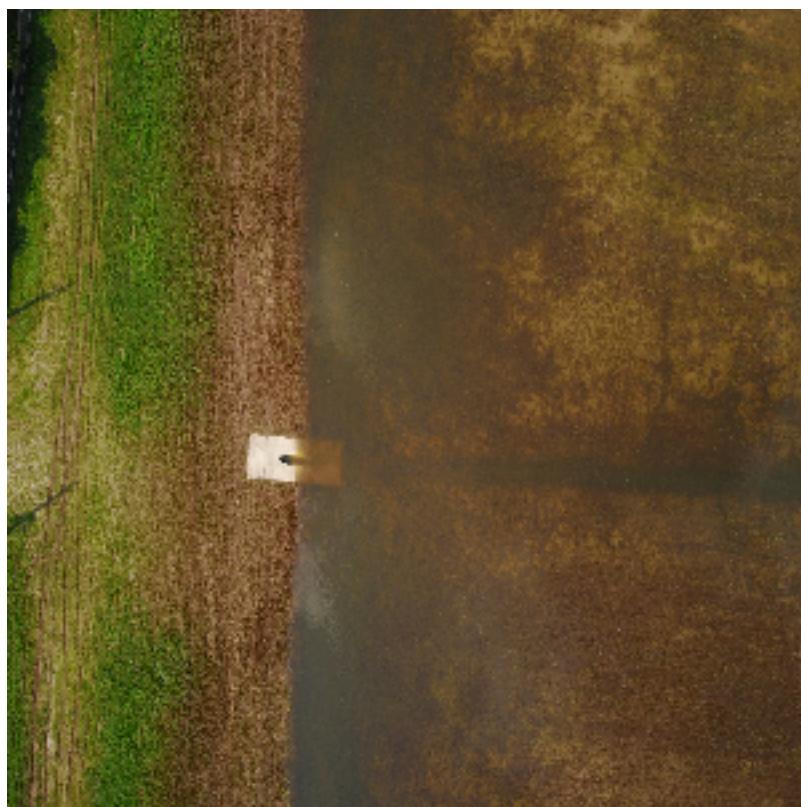
Flooded images: 1445 Flooded masks: 1445  
 Non-Flooded images: 450 Non-Flooded masks: 450

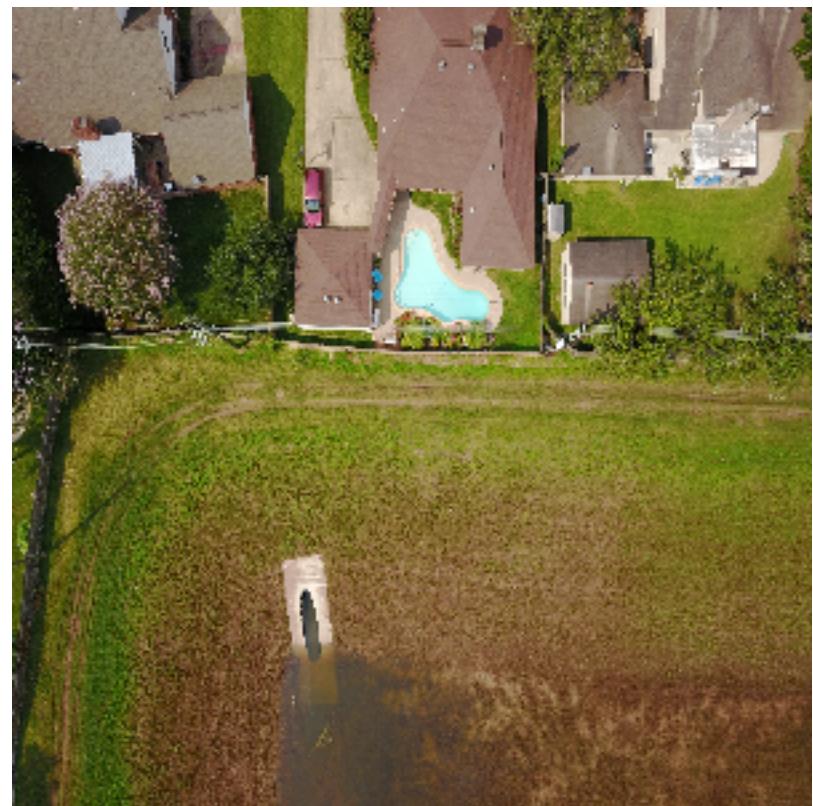
```
[ ]: sample_f_img = cv2.imread(os.path.join(f_img_dir, sorted(os.
   .listdir(f_img_dir))[0]))
sample_f_mask = cv2.imread(os.path.join(f_mask_dir, sorted(os.
   .listdir(f_mask_dir))[0]))
sample_n_img = cv2.imread(os.path.join(n_img_dir, sorted(os.
   .listdir(n_img_dir))[0]))
sample_n_mask = cv2.imread(os.path.join(n_mask_dir, sorted(os.
   .listdir(n_mask_dir))[0)))

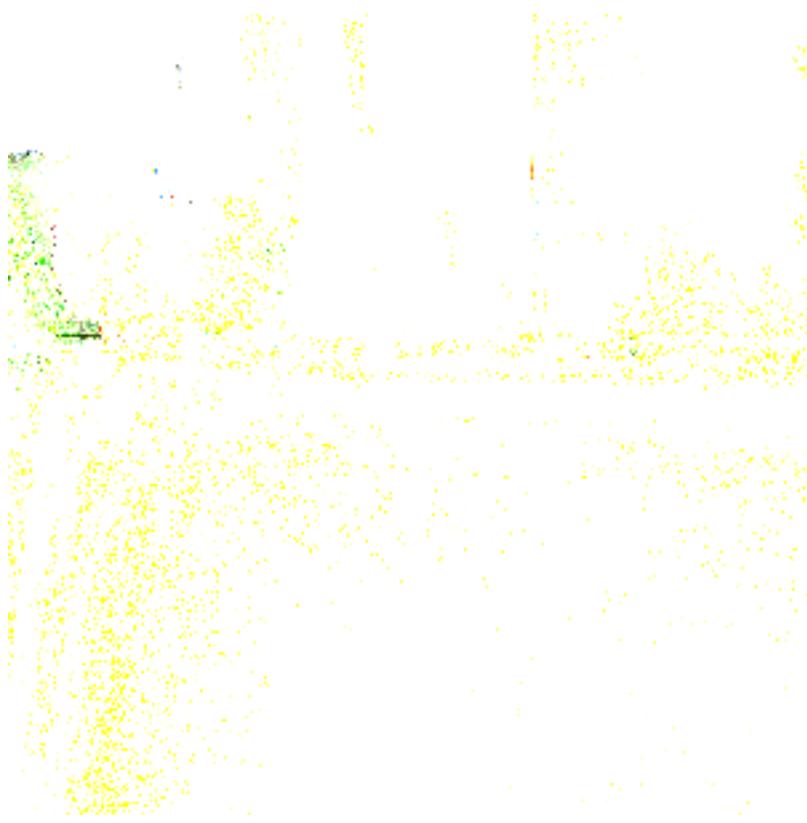
print(np.min(sample_f_mask),
np.max(sample_f_mask),
```

```
np.min(sample_n_mask),  
np.max(sample_n_mask))  
  
sample_f_mask = sample_f_mask * (255/9)  
sample_n_mask = sample_n_mask * (255/9)  
  
print(np.min(sample_f_mask),  
np.max(sample_f_mask),  
np.min(sample_n_mask),  
np.max(sample_n_mask))  
  
cv2_imshow(cv2.resize(sample_f_img, dsize=(300, 300)))  
cv2_imshow(cv2.resize(sample_f_mask, dsize=(300, 300)))  
cv2_imshow(cv2.resize(sample_n_img, dsize=(300, 300)))  
cv2_imshow(cv2.resize(sample_n_mask, dsize=(300, 300)))
```

0 255 0 255  
0.0 7225.0 0.0 7225.0







```
[ ]: kernel = np.ones((2,2),np.uint8)
cv2.imshow(cv2.erode(cv2.dilate(cv2.bilateralFilter(sample_f_img, 5, 75, 75), kernel, iterations=2), kernel, iterations=1))
```

```
[ ]: CLASSES={'Background':0,'Building-flooded':1,'Building-non-flooded':
˓→2,'Road-flooded':3,'Road-non-flooded':4,
'Water':5,'Tree':6,'Vehicle':7,'Pool':8,'Grass':9}
IMG_DIM= 512
```

```
[ ]: # CONFIG
TENSORBOARD_DIR = "/content/drive/MyDrive/LP_MI/runs/"
MODEL_DIR = "/content/drive/MyDrive/LP_MI/models/"
UNLABELLED_SPLIT = 100 # first 100 unlabelled examples will be used
## RAM storage error
os.makedirs(TENSORBOARD_DIR, exist_ok=True)
os.makedirs(MODEL_DIR, exist_ok=True)

##### UNET CONFIG #####
# ENCODER_DEPTH=5
```

```

# DECODER_CHANNELS=(256, 128, 64, 32, 16)
# # BATCH_SIZE= [8, 16, 32, 64, 128]
# # LR = [1, 1e-2, 1e-4, 1e-6]
# BATCH_SIZE = [8]
# LR = [1e-2]
# EPOCHS= 50
# ENCODER_NAME= 'resnet34'

# ##### DEEPLAB V3+ CONFIG #####
ENCODER_DEPTH=5
DECODER_CHANNELS=256
BATCH_SIZE= [8]
EPOCHS= 100
LR = [1e-3]
ENCODER_NAME= 'efficientnet-b3'

```

```

[ ]: from albumentations.pytorch import ToTensorV2

dirs = """
flood_dir
non_flood_dir

f_img_dir
f_mask_dir
n_img_dir
n_mask_dir
"""

x_f = [os.path.join(f_img_dir, file) for file in sorted(os.listdir(f_img_dir))]
x_n = [os.path.join(n_img_dir, file) for file in sorted(os.listdir(n_img_dir))]

x = x_f + x_n

x_train, x_test = model_selection.train_test_split(x, test_size=0.3, □
    ↪shuffle=True)

train_transform1 = A.Compose([
    A.Resize(IMG_DIM, IMG_DIM),
    A.ShiftScaleRotate(shift_limit=0.2, scale_limit=0.2, rotate_limit=30, □
        ↪p=0.5),
    # A.RGBShift(r_shift_limit=25, g_shift_limit=25, b_shift_limit=25, p=0.5),
    A.RandomBrightnessContrast(brightness_limit=0.3, contrast_limit=0.3, □
        ↪p=0.5),
    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))
])

train_transform2=A.Compose([ToTensorV2()])

```

```

val_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

```

```

[ ]: class SegDataset:
    def __init__(self, x_paths, trans1, trans2, img_dim, unlabelled=False):
        self.x_paths = x_paths
        self.unlabelled = unlabelled
        if not self.unlabelled:
            self.y_paths = [x.replace("image", "mask").replace(".jpg", "_lab.
˓→png") for x in self.x_paths]
        else:
            self.y_paths = None
        self.img_dim = img_dim
        self.trans1 = trans1
        self.trans2 = trans2

    def __len__(self):
        return len(self.x_paths)

    def get_newMask(self, mask, classes, dim=IMG_DIM):
        mask = torch.as_tensor(mask[:, :, 0], dtype=torch.int64)
        return torch.moveaxis(torch.nn.functional.one_hot(mask, num_classes=len(classes)), -1, 0)

    def __getitem__(self, index):
        image = cv2.imread(self.x_paths[index])

        kernel = np.ones((2, 2), np.uint8)
        image = cv2.bilateralFilter(image, 5, 75, 75)
        image = cv2.erode(cv2.dilate(image, kernel, iterations=2), kernel, iterations=1)
        if not self.unlabelled:
            mask = cv2.imread(self.y_paths[index])
        else:
            mask = np.random.rand(512, 512, 3)
        if self.trans1:
            transformed1 = self.trans1(image=image, mask=mask)
            image = transformed1["image"]
            mask = self.get_newMask(transformed1["mask"], CLASSES)
        if self.trans2:
            transformed2 = self.trans2(image=image, mask=mask)
            image = transformed2["image"]
        return image, mask

```

```
[ ]: def dice_loss(pred, target, smooth = 1e-5):
    pred = pred.contiguous()
    target = target.contiguous()

    intersection = (pred * target).sum(dim=2).sum(dim=2)

    loss = (1 - ((2. * intersection + smooth) / (pred.sum(dim=2).sum(dim=2) + target.sum(dim=2).sum(dim=2) + smooth)))

    return loss.mean()
```

```
[ ]: def calc_loss(pred, target, metrics, bce_weight=0.5, unlabelled=False):
    bce = F.binary_cross_entropy_with_logits(pred, target.to(torch.float32))
    pred = F.sigmoid(pred)
    dice = dice_loss(pred, target)

    target_np=target.data.cpu().numpy()
    pred_np=pred.data.cpu().numpy()
    MIoU= np.mean(sklearn.metrics.jaccard_score(np.argmax(target_np, axis=1).flatten(), np.argmax(pred_np, axis=1).flatten(), average=None))
    loss = bce * bce_weight + dice * (1 - bce_weight)
    if not unlabelled:
        metrics['bce'] += bce.data.cpu().numpy() * target.size(0)
        metrics['dice'] += dice.data.cpu().numpy() * target.size(0)
        metrics['loss'] += loss.data.cpu().numpy() * target.size(0)
        metrics['MIoU'] += MIoU * target.size(0)

    return loss
```

```
[ ]: def print_metrics(metrics, epoch_samples, phase):
    outputs = []
    for k in metrics.keys():
        outputs.append("{}: {:.4f}".format(k, metrics[k] / epoch_samples))
    print("{}: {}".format(phase, ", ".join(outputs)))
```

```
[ ]:
```

## Train Function

```
[ ]: training_set = SegDataset(x_train, train_transform1, train_transform2,
                             img_dim=IMG_DIM)
testing_set = SegDataset(x_test, train_transform1, train_transform2,
                        img_dim=IMG_DIM)
image_datasets = {'train': training_set, 'valid': testing_set}
```

```
[ ]: u_dir = "/content/drive/MyDrive/LP_MI/FloodNet-Supervised_v1.0/train/
      train-org-img"
```

```

unlabelled_paths = [os.path.join(u_dir, file) for file in os.listdir(u_dir)]
unlabelled_set = SegDataset(unlabelled_paths[0:-500],  

    ↪trans1=train_transform1, trans2=train_transform2, img_dim=IMG_DIM,  

    ↪unlabelled=True)
image_datasets["unlabelled"] = unlabelled_set

[ ]: def train_model(unique_name, num_epochs=EPOCHS, start_alpha_from=15,  

    ↪reach_max_alpha_in=655, max_alpha=0.5):
    for lr in LR:
        for bs in BATCH_SIZE:
            # if(lr == 1 and (bs == 8 or bs == 16)):
            #     continue
            print("__"*80)
            print("__"*80)
            print(f"name: {unique_name} LR: {lr} BS: {bs}")
            print("__"*80)
            print("__"*80)

    alphas = np.linspace(0, max_alpha, reach_max_alpha_in-start_alpha_from)

    os.makedirs(os.path.join(TENSORBOARD_DIR, f'{unique_name}-{lr}-{bs}'),  

    ↪exist_ok=True)
    writer = SummaryWriter(log_dir=os.path.join(TENSORBOARD_DIR,  

    ↪f'{unique_name}-{lr}-{bs}') )

    best_loss = 1e10
    best_epoch = 0
    best_miou = 0

    train_data_loader = torch.utils.data.DataLoader(training_set,  

    ↪batch_size=bs, num_workers=0)
    test_data_loader = torch.utils.data.DataLoader(testing_set,  

    ↪batch_size=bs, num_workers=0)
    ulbl_data_loader = torch.utils.data.DataLoader(unlabelled_set,  

    ↪batch_size=bs, num_workers=0)
    dataloaders = {'train': train_data_loader, 'valid': test_data_loader,  

    ↪"unlabelled" : ulbl_data_loader}

    # model = segmentation_models_pytorch.Unet(encoder_name=ENCODER_NAME,  

    ↪encoder_depth=ENCODER_DEPTH,
    #                                     decoder_channels=DECODER_CHANNELS,  

    ↪classes=len(CLASSES))
    model = segmentation_models_pytorch.  

    ↪DeepLabV3Plus(encoder_name=ENCODER_NAME, encoder_depth=ENCODER_DEPTH,
                    decoder_channels=DECODER_CHANNELS,  

    ↪classes=len(CLASSES))

```

```

model = model.to(device)
model.load_state_dict(torch.load(os.path.join(MODEL_DIR, u
↪"abhi_sudo_full-pretrained_preproc_1-deeplabv3+-ep_7-0.001-8.pt")))
best_model_wts = copy.deepcopy(model.state_dict())

# optimizer = optim.Adam(model.parameters(), lr=lr) ## IF YOU CHANGE ↴
↪THIS, CHANGE THE UNIQUE ABOVE
optimizer = optim.SGD(model.parameters(), lr=0.01)
# scheduler = lr_scheduler.MultiStepLR(optimizer, milestones=[7, 20, 80], u
↪gamma=0.1)
scheduler = None

for epoch in range(num_epochs):
    print('_' * 80)
    print('Epoch {} / {}'.format(epoch, num_epochs - 1))
    print('_' * 80)

    if epoch < start_alpha_from:
        alpha = 0
    elif epoch - start_alpha_from >= len(alphas):
        alpha = alphas[-1]
    else:
        alpha = alphas[max(0, epoch - start_alpha_from)]

    since = time.time()

    for phase in ['train', 'unlabelled', 'valid']:
        print(phase)
        print("_" * 20)
        if alpha == 0 and phase == 'unlabelled':
            continue
        if phase in ['train', 'unlabelled']:
            model.train()
        else:
            model.eval()

        metrics = defaultdict(float)
        epoch_samples = 0

        for batch_no, (inputs, labels) in u
↪enumerate(tqdm(dataloaders[phase])):
            inputs = inputs.to(device)
            if phase in ['train', 'valid']:
                labels = labels.to(device)

            # zero the parameter gradients
            optimizer.zero_grad()

```

```

# forward
with torch.set_grad_enabled(phase in ['train', 'unlabelled']):
    outputs = model(inputs)
    # loss = calc_loss(outputs, labels, metrics)

    if phase in ["train", "valid"]:
        loss = calc_loss(outputs, labels, metrics)
    else:
        loss = alpha * calc_loss(outputs, □
    ↪pseudo_labels[batch_no].to(device), metrics, unlabelled=True)
        # backward + optimize only if in training phase
        if phase in ['train', 'unlabelled']:
            loss.backward()
            optimizer.step()

    # statistics
    epoch_samples += inputs.size(0)

if scheduler is not None and phase == "train":
    scheduler.step()

print_metrics(metrics, epoch_samples, phase)
epoch_loss = metrics['loss'] / epoch_samples
epoch_miou = metrics['MIoU'] / epoch_samples

## tensorboard writer
if phase == "train":
    writer.add_scalar(f'Loss/{phase}', epoch_loss, epoch)
    writer.add_scalar(f'MIoU/{phase}', epoch_miou, epoch)
    writer.add_scalar(f'Alpha/{phase}', alpha, epoch)
    # writer.add_scalars(f'Loss/{phase}', {'loss':epoch_loss, □
    ↪'alpha':alpha}, epoch)
        # writer.add_scalars(f'MIoU/{phase}', {'miou':epoch_miou, □
    ↪'alpha':alpha}, epoch)

    if phase == "valid": # older implementation had phase == 'val'
        writer.add_scalar(f'Loss/val', epoch_loss, epoch)
        writer.add_scalar(f'MIoU/val', epoch_miou, epoch)
        writer.add_scalar(f'Alpha/val', alpha, epoch)
        # writer.add_scalars(f'Loss/val', {'loss':epoch_loss, 'alpha': □
    ↪alpha}, epoch)
            # writer.add_scalars(f'MIoU/val', {'miou':epoch_miou, 'alpha': □
    ↪alpha}, epoch)

## generate pseudo labels
if phase == 'train' and epoch >= start_alpha_from-1:

```

```

        pseudo_labels = []
        model.eval()
        for inputs, _ in tqdm(dataloaders['unlabelled'], □
        ↪desc="Predicting pseudo labels"):
            inputs = inputs.to(device)
            with torch.no_grad():
                outputs = model(inputs)
                pseudo_labels.append(outputs.detach().cpu())

        # deep copy the model
        if phase == 'valid' and epoch_miou > best_miou:
            best_miou = epoch_miou
            best_epoch = epoch
            best_model_wts = copy.deepcopy(model.state_dict())
            print(f'Best miou: {best_miou:.4f} Epoch: {epoch}')

        if epoch % 5 == 0:
            PATH = os.path.join(MODEL_DIR, □
            ↪f'{unique_name}-ep_{best_epoch}-{lr}-{bs}.pt')
            torch.save(best_model_wts, PATH)
            PATH = os.path.join(MODEL_DIR, □
            ↪f'{unique_name}-ep_{epoch}-{lr}-{bs}.pt')
            torch.save(model.state_dict(), PATH)

        time_elapsed = time.time() - since
        print('{:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60))

        print('Best val loss: {:.4f}'.format(best_loss))
        writer.close()
        # load best model weights
        # model.load_state_dict(best_model_wts)
        PATH = os.path.join(MODEL_DIR, f'{unique_name}-ep_{best_epoch}-{lr}-{bs}.pt')
        torch.save(best_model_wts, PATH)

    return model

```

[ ]:

## Training

```
[ ]: device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
os.chdir("/content/")
```

cuda:0

```
[ ]: !ls /content/drive/MyDrive/LP_MI/runs
```

```
[ ]:
```