Segment features around residential buildings in UAV images of flooded areas taken in Houston after Hurricane Harvey – Final Score: 72.95%

HOUSSAM FOUKI[1], KENZA BADDOU[2], AND ALEXANDER MAISURADZE[3]

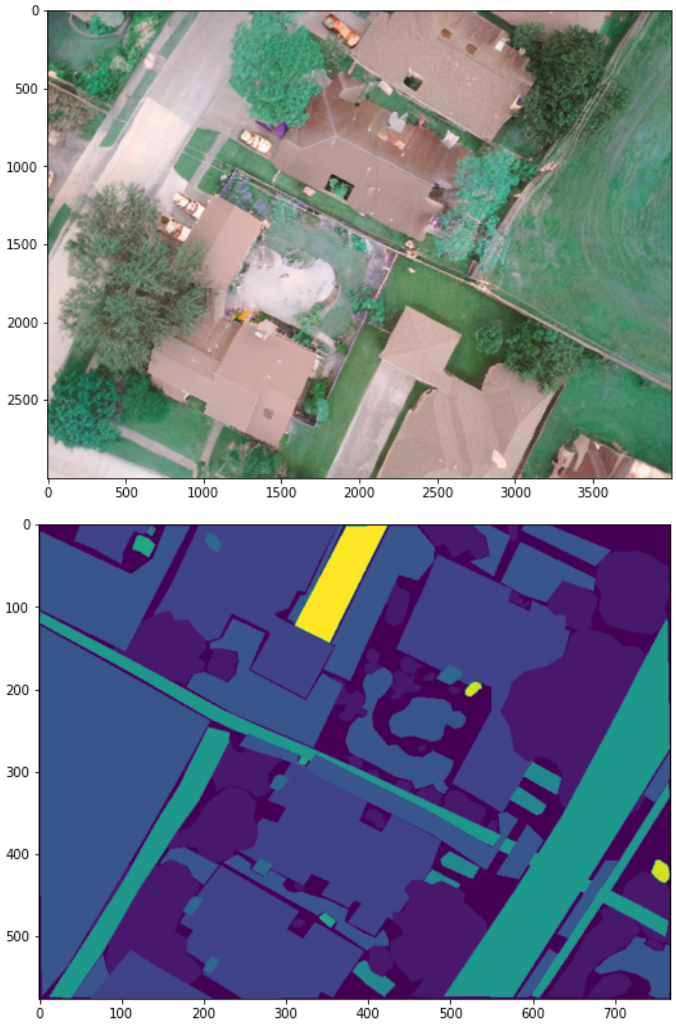
[1] *houssam.fouki@student-cs.fr*
[2] *kenza.baddou@student-cs.fr*
[3] *alexander.maisuradze@student-cs.fr*

Compiled January 20, 2023

Problem Statement – **The goal of this project is to design and implement a deep learning model that can automatically segment images. The images in question were acquired by a small unmanned aerial vehicle (sUAV) in the city of Houston, Texas, and are being used to assess the damage to residential and public properties following the impact of Hurricane Harvey. The model will be required to segment the images into 26 distinct categories, such as property roofs, swimming pools, vehicles, and grass. The overall objective is to accurately identify and classify different elements within the images to aid in the assessment of damage caused by Hurricane Harvey. More concretely, the goal of this semantic image segmentation task is to label each pixel of an image with a corresponding class of what is being represented out of 26 possible categories. The expected output itself is an image. In order to tackle this task, we have focused on a very successful architecture, U-Net, which has proven to be performing well on similar image-segmentation tasks.**

1. IMAGE PRE-PROCESSING APPROACHES

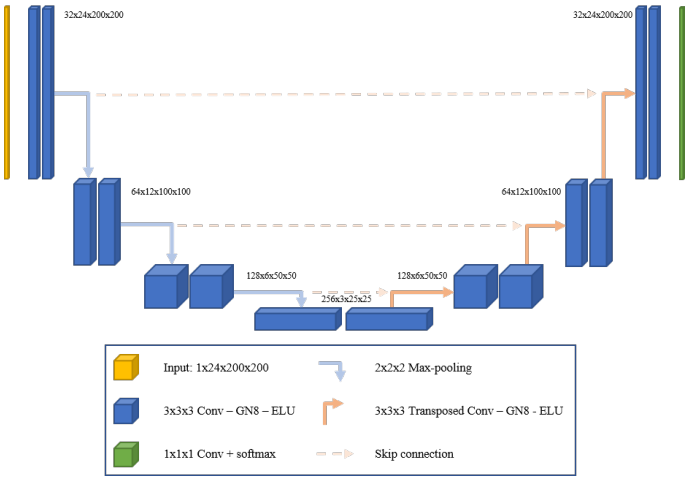
We used the torchvision.transforms module for image pre-processing and applied different transformations to the train images compared to the validation and test sets. We used image augmentation techniques to avoid overfitting and increase the number of images in the train dataset. The images were taken by UAVs, so reversing the horizontal and vertical axes did not affect the information.
For our project, we selected various image dimensions based on the GPU specifications and time limitations. Afterwards, we employed random vertical and horizontal flips with a probability of 0.5 for enhancing the images through image augmentation.



2. MODEL ARCHITECTURE

Throughout the project, we employed various types of convolutional networks for image segmentation. However, the model that yielded the best results was U-NET. U-NET is a convolutional neural network developed by Olaf Ronneberger, Philipp Fischer, and Thomas Brox in 2015. It is based on the fully convolutional network created by Long, Shelhamer, and Darrell in 2014.Initially designed for image segmentation in the biomedical field, U-NET has since been utilized for a variety of other applications, making it an ideal choice for our project.

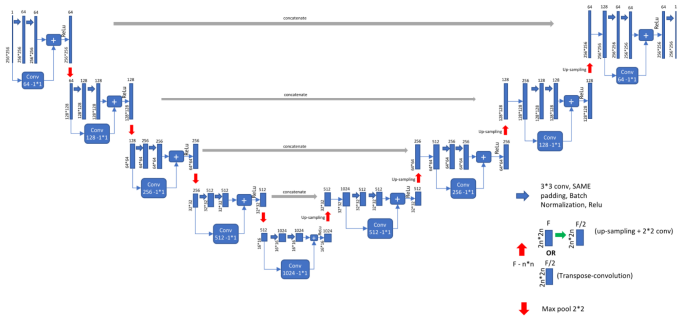
Architecture: U-NET architecture we implemented consists of two paths. The first path, known as the contraction path or encoder, is used to capture the context of the image. This path is made up of a traditional stack of convolutional and max pooling layers. The second path, known as the symmetric expanding path or decoder, enables precise localization through the use of transposed convolutions.



The U-NET architecture is built using the PyTorch Module class, which allows for the inheritance of various models and sub-models. A double_conv method is defined, which includes a sequence of convolutional layers, batch normalization, and ReLU activation functions. A double_conv_downs method, which uses PyTorch's ModuleList functionality to perform the double_conv on input channel dimensions, is also defined for the encoder procedure. Additionally, upsampling blocks are implemented using the ConvTranspose2d layer to increase the spatial dimensions of feature maps, and a max_pool_2x2 layer is used to reduce the spatial dimensions of feature maps by a factor of 2.

To improve the accuracy of location information, we incorporate skip connections into the decoder by combining the output of the transposed convolution layers with the corresponding feature maps from the Encoder at each step. In the forward process, we merge the i-th intermediate feature map from the encoder with the current output x from the upsampling block. To ensure that the spatial dimensions of encoded features and x align, we employ resizing. The combined output is then passed through the i-th decoder function.

In addition to utilizing the U-NET model, we also explored other learning algorithms. One of these was the SegNet model, which is similar to U-NET in that it is a semantic segmentation model that includes both an encoder and a decoder. The encoder and decoder were both constructed with a combination of Convolution, Batch Normalisation, ReLU activation function, and Pooling layers. Another algorithm we tried was the ResUNET, which is a variation of U-NET that incorporates residual blocks. Our decision to use this approach was driven by the fact that previous attempts to improve accuracy through increasing the number of layers or using augmentation techniques had resulted in overfitting. We also used Mask RCNN, we first cloned a repository containing some of the necessary code blocks. We then imported weights from the COCO dataset for the model and attempted to set up a custom data class. Unfortunately, the training results were not as successful as we had hoped.



3. APPENDIX

```
!mkdir Hurricane_Harvey
!mkdir Hurricane_Harvey/rasters Hurricane_Harvey/vectors
!gsutil -m cp -n -r gs://geoengine-dataset-houston-uav/rasters/
    raw Hurricane_Harvey/rasters/
!gsutil -m cp -n -r gs://geoengine-dataset-houston-uav/vectors/
    random-split-_2022_11_17-22_35_45/ Hurricane_Harvey/vectors/
!pip install segmentation-models-pytorch
```

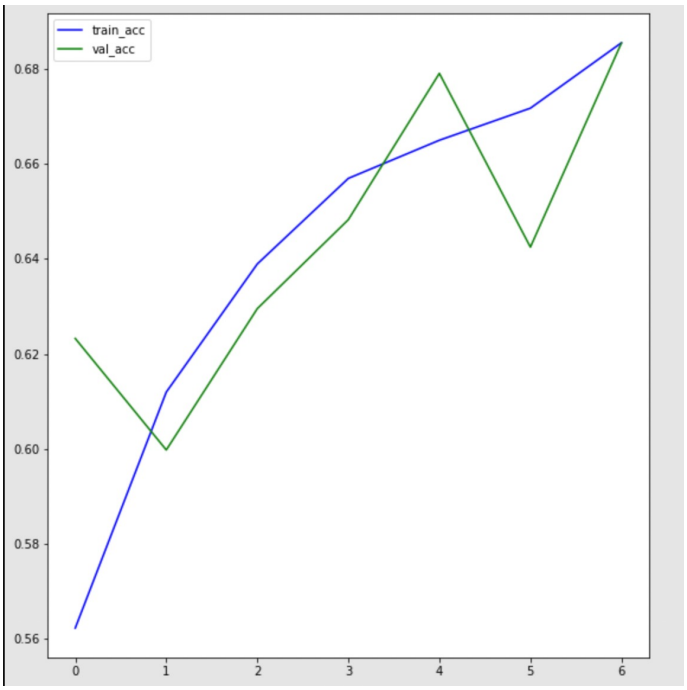


Fig. 1. Plot of accuracy in train and validation set (synthesize dataset)

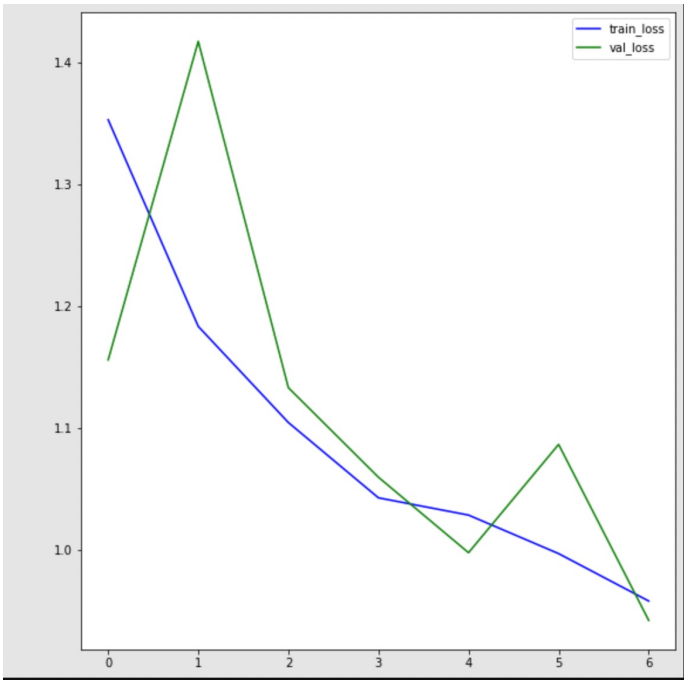


Fig. 2. Plot of loss in train and validation set (synthesize dataset)

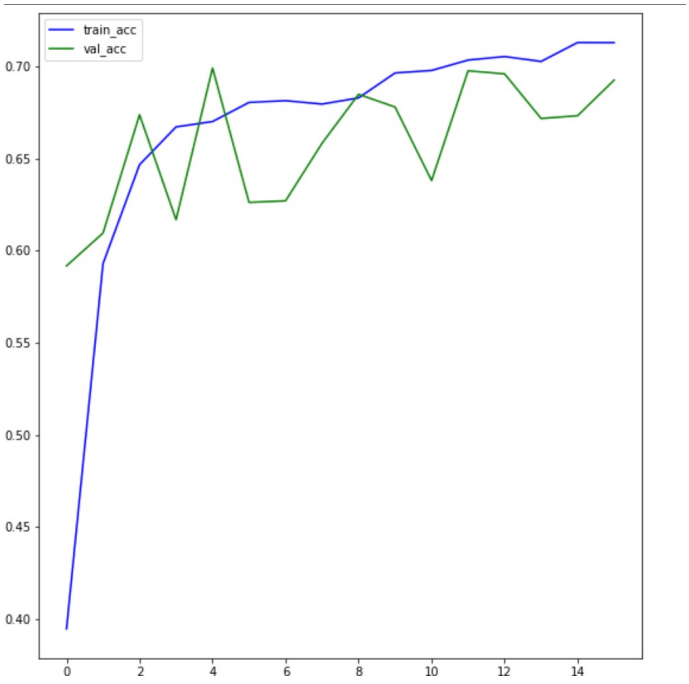


Fig. 3. Plot of accuracy in train and validation set (main dataset)

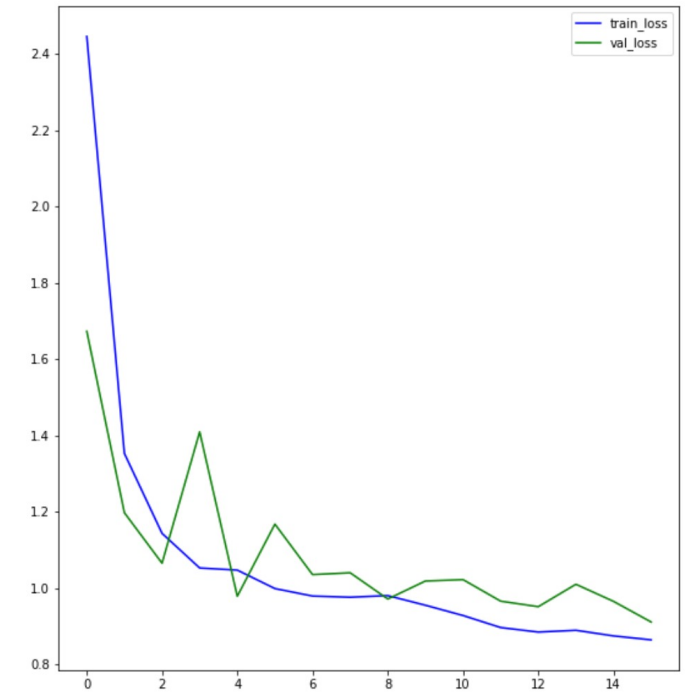


Fig. 4. Plot of loss in train and validation set (main dataset)

```
!pip install torchsummary
import os
import glob
import numpy as np
import matplotlib.pyplot as plt
import cv2
from PIL import Image
from tqdm.notebook import tqdm
from google.colab.patches import cv2_imshow
from tqdm import tqdm as progressbar
from sklearn.model_selection import train_test_split
from google.colab import drive
drive.mount('/content/drive')
import torch

def train_one_epoch(model, dataloader_train, dataloader_valid, optimizer, loss_function, epoch):
    #training phase
    model.train()
    train_loss = 0
    train_accuracy = 0
    for imgs, masks in tqdm(dataloader_train):
        optimizer.zero_grad()
        imgs = imgs.to(DEVICE)
        masks = masks.to(DEVICE)
        # print(imgs.shape, masks.shape)
        #forward pass
        outputs = model(imgs)
        #cal loss and backward
        loss = loss_function(outputs, masks.type(torch.int64))
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
        train_accuracy += pixel_accuracy(outputs, masks)
    train_loss /= len(dataloader_train)
    train_accuracy /= len(dataloader_train)

    #validating phase
    model.eval()
    valid_loss = 0
    val_accuracy = 0
    with torch.no_grad():
        for imgs, masks in tqdm(dataloader_valid):
            imgs = imgs.to(DEVICE)
            masks = masks.to(DEVICE)
            outputs = model(imgs)
            loss = loss_function(outputs, masks.type(torch.int64))
            valid_loss += loss.item()
            val_accuracy += pixel_accuracy(outputs, masks)
    valid_loss /=len(dataloader_valid)
    val_accuracy /= len(dataloader_valid)
    print(f'EPOCH: {epoch + 1} - train loss: {train_loss} - train_accuracy: {train_accuracy} - valid_loss: {valid_loss} - val_accuracy: {val_accuracy}')
    return train_loss, train_accuracy, valid_loss, val_accuracy
import albumentations.augmentations.functional as F
from torch.utils import data
import albumentations as A
from albumentations.pytorch import ToTensorV2

class HarveyDataset(data.Dataset):
    def __init__(self, image_paths, mask_paths, transform=None):
        self.image_paths = image_paths
        self.mask_paths = mask_paths
        self.transform = transform

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

    def __getitem__(self, _id):
        image = np.array(Image.open(self.image_paths[_id]))
        origin_mask = np.array(Image.open(self.mask_paths[_id]), dtype=np.int64)

        # print(f'Img: {os.path.basename(self.image_paths[_id])}\nMask: {os.path.basename(self.mask_paths[_id])}')

        origin_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        if self.transform is not None:
            transformed = self.transform(image=origin_image, mask=origin_mask)
            image = transformed["image"]
            mask = transformed["mask"]
        # return origin_image, origin_mask, image, mask
        return image, mask
def model_accuracy(output, target):

  # Transform the output to get the right format
  # output_softmax = F.softmax(output, dim=1)
  output_argmax = torch.argmax(output, dim=1)

  # Get the correct predictions as a boolean mask
  corrects = (output_argmax  == target)

  # Compute accuracy
  accuracy = corrects.sum().float() / float( target.size(0) * target.size(1) * target.size(2) )

  return accuracy * 100

def pixel_accuracy(output, mask):
    with torch.no_grad():
        output = torch.argmax(torch.nn.functional.softmax(output, dim=1), dim=1)
        correct = torch.eq(output, mask).int()
        accuracy = float(correct.sum()) / float(correct.numel())
    return accuracy
image_paths = glob.glob('/content/Hurricane_Harvey/rasters/raw/*.tif')
mask_paths = glob.glob('/content/Hurricane_Harvey/vectors/random-split-_2022_11_17-22_35_45/Masks/*.png')


print(f'Len image: {len(image_paths)}, Len mask: {len(mask_paths)}')
temp = list(map(lambda x: os.path.basename(x)[:-3], mask_paths))

train_image_paths = list(filter(lambda x: os.path.basename(x)[:-3] in temp, image_paths))
test_image_paths = list(set(image_paths) -  set(train_image_paths))
assert len(train_image_paths) == len(temp), 'Len train path should be the same to number mask'

print(f'Len train: {len(train_image_paths)}, Len test: {len(test_image_paths)}')
train_image_paths = sorted(train_image_paths)
train_mask_paths = sorted(mask_paths)
X_train, X_val, y_train, y_val = train_test_split(train_image_paths, train_mask_paths, test_size=0.1, random_state=42)

print(f'Number train: {len(X_train)}')
print(f'Number train: {len(X_val)}')
train_transform = A.Compose(
    [
        A.Resize(576, 768),
        A.HorizontalFlip(p=0.5),
        A.VerticalFlip(p=0.5),

        A.OneOf([
          A.RandomBrightnessContrast(brightness_limit=0.3, contrast_limit=0.3, p=0.5),
          A.Affine(translate_px={"x": (-20, 20), "y": (-20, 20)}),
          A.GaussNoise(p=0.5),
          A.Affine(scale=(0.8, 1.2)),
        ]),
        A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
        ToTensorV2(),
    ]
)

val_transform = A.Compose(
    [
        A.Resize(576, 768),
        A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
        ToTensorV2(),
    ]
)


train_dataset = HarveyDataset(X_train, y_train, transform=train_transform)
val_dataset = HarveyDataset(X_val, y_val, transform=val_transform)
origin_image, origin_mask, image, mask = train_dataset[np.random.randint(len(y_train))]
figure, ax = plt.subplots(nrows=1, ncols=4, figsize=(40, 20))
ax[0].imshow(origin_image)
ax[1].imshow(origin_mask)
ax[2].imshow(image)
ax[3].imshow(mask)
DEVICE = 'cuda'
BATCH_SIZE=2
import segmentation_models_pytorch as smp
model =  smp.UnetPlusPlus(
                  encoder_name='resnet101',
                  encoder_weights='imagenet',
                 #  activation='sigmoid',
                  in_channels=3,
                  classes=27,
                  encoder_depth=5,
                  decoder_channels=[1024, 512,256,128,64]
          ).to(DEVICE)
          train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True, num_workers=2)
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False, num_workers=2)
```

```
!ls '/content/drive/MyDrive/transfer_checkpoints/'
!rm '/content/drive/MyDrive/temp1_checkpoints'/new_new_model*
!rm ./checkpoints/*
max_lr = 1e-3
# epoch = 50
weight_decay = 1e-4
loss = torch.nn.CrossEntropyLoss()

optimizer = torch.optim.AdamW(model.parameters(), lr=max_lr, weight_decay=weight_decay)

max_score = 0
NUM_EPOCH = 200
hist_loss = []
for epoch in range(NUM_EPOCH):
  print(f'==================================Epoch: {epoch+1}')
  ret = train_one_epoch(model, train_loader, val_loader, optimizer, loss, epoch+1)
  hist_loss += [ret]
  if ret[1]+ret[3] > max_score:
    torch.save(model, f'./checkpoints/model3_{round(ret[1]*100)}_{round(ret[3]*100)}_{round(ret[2], 3)}.pt')
    max_score = ret[1] + ret[3]
    print(f'Save model: model3_{round(ret[1]*100)}_{round(ret[3]*100)}_{round(ret[2], 3)}.pt')
    train_losses = []
val_losses = []
for _train_loss, _val_loss in hist_loss:
  train_losses += [_train_loss]
  val_losses += [_val_loss]

plt.plot(train_losses)
plt.show()
plt.plot(val_losses)
plt.show()
!mkdir Hurricane_Harvey_Synthetic
!mkdir Hurricane_Harvey_Synthetic/rasters Hurricane_Harvey_Synthetic/vectors
!gsutil -m cp -n -r gs://geoengine-dataset-houston-uav-synthetic/rasters/raw Hurricane_Harvey_Synthetic/rasters/
!gsutil -m cp -n -r gs://geoengine-dataset-houston-uav-synthetic/vectors/random-split_2022_11_21-11_59_40/ Hurricane_Harvey_Synthetic/vectors/
# create the masks from the json document
import json
import numpy as np
import os
from PIL import Image, ImageDraw
from tqdm.notebook import tqdm

def create_masks(json_folder, output_folder):
    class_colormap = {}
    for json_file in tqdm(os.listdir(json_folder)):
        with open(json_folder + json_file) as f:
            data = json.load(f)
        try:
          image_width = data["images"][0]["width"]
          image_height = data["images"][0]["height"]
          image_name = data["images"][0]["name"]
          # print(f"image_name {image_name}")
          if f'{image_name}.png' in os.listdir(output_folder):
            print("\t image alredy saved")
            continue
          mask = Image.new('L', (image_width, image_height))
          for annotation in data["annotations"]:
              segmentation = annotation["segmentation"][0]
              segmentation = [(segmentation[i], segmentation[i+1]) for i in range(0, len(segmentation) - 2, 2)]
              label = annotation["properties"][0]["labels"][0]
              # check if this label has been seen before
              if label not in class_colormap:
                  # choose a new grayscale value for this label
                  class_colormap[label] = len(class_colormap)
              ImageDraw.Draw(mask).polygon(segmentation, fill=class_colormap[label])
          mask.save(os.path.join(output_folder, f'{image_name}.png'))
        except KeyError as e:
          print(f"Exception {e} raised for {json_file}")
          !mkdir '/content/Hurricane_Harvey_Synthetic/vectors/random-split_2022_11_21-11_59_40/Masks/'

json_folder = "/content/Hurricane_Harvey_Synthetic/vectors/random-split_2022_11_21-11_59_40/COCO/"
output_folder = "/content/Hurricane_Harvey_Synthetic/vectors/random-split_2022_11_21-11_59_40/Masks/"
create_masks(json_folder, output_folder)
image_paths = glob.glob('/content/Hurricane_Harvey_Synthetic/rasters/raw/*.tif')
mask_paths = glob.glob('/content/Hurricane_Harvey_Synthetic/vectors/random-split_2022_11_21-11_59_40/Masks/*.png')


print(f'Len image: {len(image_paths)}, Len mask: {len(mask_paths)}')
temp = list(map(lambda x: os.path.basename(x)[:-3], mask_paths))

train_image_paths = list(filter(lambda x: os.path.basename(x)[:-3] in temp, image_paths))
test_image_paths = list(set(image_paths) -  set(train_image_paths))

assert len(train_image_paths) == len(temp), 'Len train path should be the same to number mask'

print(f'Len train: {len(train_image_paths)}, Len test: {len(test_image_paths)}')
train_image_paths = sorted(train_image_paths)
train_mask_paths = sorted(mask_paths)

X_train, X_val, y_train, y_val = train_test_split(train_image_paths, train_mask_paths, test_size=0.1, random_state=42)

print(f'Number train: {len(X_train)}')
print(f'Number train: {len(X_val)}')
train_transform = A.Compose(
    [
        # A.Resize(224, 224),
        A.Resize(576, 768),
        A.HorizontalFlip(p=0.5),
        A.VerticalFlip(p=0.5),

        A.OneOf([
          A.RandomBrightnessContrast(brightness_limit=0.3, contrast_limit=0.3, p=0.5),
          A.Affine(translate_px={"x": (-20, 20), "y": (-20, 20)}),
          A.GaussNoise(p=0.5),
          A.Affine(scale=(0.8, 1.2)),
        ]),
        A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
        ToTensorV2(),
    ]
)

val_transform = A.Compose(
    [
        A.Resize(576, 768),
        # A.Resize(224, 224),
        A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
        ToTensorV2(),
    ]
)


train_dataset = HarveyDataset(X_train, y_train, transform=train_transform)
val_dataset = HarveyDataset(X_val, y_val, transform=val_transform)
origin_image, origin_mask, image, mask = train_dataset[np.random.randint(len(y_train))]
figure, ax = plt.subplots(nrows=1, ncols=4, figsize=(40, 20))
ax[0].imshow(origin_image)
ax[1].imshow(origin_mask)
ax[2].imshow(image)
ax[3].imshow(mask)
import segmentation_models_pytorch as smp
import torch
# model = smp.Unet('resnet50', encoder_weights=None, classes=27, in_channels=3 ,activation='softmax', encoder_depth=5, decoder_channels=[512,256,128,64,32]).to(DEVICE)
# model = smp.Unet('resnet101', encoder_weights='imagenet', classes=27, in_channels=3 ,activation=None, encoder_depth=5, decoder_channels=[1024, 512,256,128,64]).to(DEVICE)

DEVICE = 'cuda'
BATCH_SIZE=2

# model =  smp.UnetPlusPlus(
#                  encoder_name='resnet101',
#                  encoder_weights='imagenet',
#                 #  activation='sigmoid',
#                  in_channels=3,
#                  classes=27,
#                  encoder_depth=5,
#                  decoder_channels=[1024, 512,256,128,64]
#           ).to(DEVICE)

model = torch.load('/content/drive/MyDrive/model1_49_57_1.326.pt')
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True, num_workers=2)
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False, num_workers=2)
max_lr = 1e-3
# epoch = 50
loss = torch.nn.CrossEntropyLoss()
weight_decay = 1e-4

optimizer = torch.optim.AdamW(model.parameters(), lr=max_lr, weight_decay=weight_decay)
```

```

max_score = 0
NUM_EPOCH = 200
hist_loss = []
for epoch in range(NUM_EPOCH):
  print(f'==================================Epoch: {epoch+1}')
  ret = train_one_epoch(model, train_loader, val_loader,
      optimizer, loss, epoch+1)
  hist_loss += [ret]
  if ret[1]+ret[3] > max_score:
    torch.save(model, f'/content/drive/MyDrive/
      transfer_checkpoints/model1_{round(ret[1]*100)}_{round(ret
      [3]*100)}_{round(ret[2], 3)}.pt')
    max_score = ret[1] + ret[3]
    print(f'Save model: model1_{round(ret[1]*100)}_{round(ret
      [3]*100)}_{round(ret[2], 3)}.pt')
    train_accuries = []
val_accuries = []
train_losses = []
val_losses = []
for _t_loss, _t_acc, _v_loss, _v_acc in hist_loss:
  train_accuries += [_t_acc]
  val_accuries += [_v_acc]
  train_losses += [_t_loss]
  val_losses += [_v_loss]
  fig, ax = plt.subplots(1,2)
fig.set_size_inches(20, 10)

ax[0].plot(train_accuries, 'b', label='train_acc')
ax[0].plot(val_accuries, 'g', label='val_acc')
ax[1].plot(train_losses, 'b', label='train_loss')
ax[1].plot(val_losses, 'g', label='val_loss')

ax[0].legend()
ax[1].legend()

fig.show()
!rm results/*
# temp_model = torch.load('/content/model_34_88.pt')
temp_model = model
temp_model.eval()

with torch.no_grad():
  for _path in tqdm(test_image_paths):
    fname = os.path.basename(_path).replace('tif', 'png')
    image = np.array(Image.open(_path))
    origin_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    transformed = val_transform(image=origin_image)
    image = transformed["image"]
    h, w = origin_image.shape[:2]


    t = torch.unsqueeze(image, 0).to(DEVICE)
    output = temp_model(t)
    output = torch.argmax(torch.nn.functional.softmax(output, dim
      =1), dim=1).detach().cpu().numpy()[0]

    output = cv2.resize(output, (w,h), interpolation=cv2.
      INTER_NEAREST)

    cv2.imwrite(f'results/{fname}', output)

import tarfile
import os

tar = tarfile.open("submission.tar", "w")

for root, dir, files in os.walk('/content/results/'):
    for  file in files:
        fullpath = os.path.join(root, file)
        tar.add(fullpath, arcname=file)

tar.close()
!cp submission.tar '/content/drive/MyDrive'
len(os.listdir('results'))
model.eval()
with torch.no_grad():
  t = torch.unsqueeze(image, 0).to(DEVICE)
  output = model(t)
  output = torch.argmax(torch.nn.functional.softmax(output, dim
     =1), dim=1).detach().cpu().numpy()

figure, ax = plt.subplots(nrows=1, ncols=2, figsize=(40, 20))
ax[0].imshow(cv2.resize(origin_image, (640,480)))
ax[1].imshow(output[0])
!rm ./*.pt
train_transform = A.Compose(
    [
        # A.Resize(352, 352),
        A.Resize(576, 768),
        A.HorizontalFlip(p=0.5),
        A.VerticalFlip(p=0.5),
        A.RandomBrightnessContrast(brightness_limit=0.3,
     contrast_limit=0.3, p=0.3),
        A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229,
     0.224, 0.225)),
        ToTensorV2(),
    ]
)

val_transform = A.Compose(
    [
        A.Resize(576, 768),
        # A.Resize(352, 352),
        A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229,
     0.224, 0.225)),
        ToTensorV2(),
    ]
)
max_lr = 1e-3
# epoch = 50
weight_decay = 1e-4

optimizer = torch.optim.AdamW(model.parameters(), lr=max_lr,
     weight_decay=weight_decay)
loss = torch.nn.CrossEntropyLoss()

max_score = 0
NUM_EPOCH = 200
hist_loss = []
for epoch in range(NUM_EPOCH):
  print(f'==================================Epoch: {epoch+1}')
  X_train, X_val, y_train, y_val = train_test_split(
      train_image_paths, train_mask_paths, test_size=0.2)

  train_dataset = HarveyDataset(X_train, y_train, transform=
      train_transform)
  val_dataset = HarveyDataset(X_val, y_val, transform=
      val_transform)

  train_loader = torch.utils.data.DataLoader(train_dataset,
      batch_size=BATCH_SIZE, shuffle=True, num_workers=2)
  val_loader = torch.utils.data.DataLoader(val_dataset,
      batch_size=BATCH_SIZE, shuffle=False, num_workers=2)


  ret = train_one_epoch(model, train_loader, val_loader,
      optimizer, loss, epoch+1)
  hist_loss += [ret]
  if ret[1]+ret[3] > max_score:
    torch.save(model, f'/content/drive/MyDrive/temp1_checkpoints/
      model_{round(ret[1]*100)}_{round(ret[3]*100)}.pt')
    max_score = ret[1]+ret[3]
    print(f'Save model: model_{round(ret[1]*100)}_{round(ret
      [3]*100)}.pt')

  del train_dataset
  del val_dataset
  del train_loader
  del val_loader
  train_accuries = []
val_accuries = []
train_losses = []
val_losses = []
for _t_loss, _t_acc, _v_loss, _v_acc in hist_loss:
  train_accuries += [_t_acc]
  val_accuries += [_v_acc]
  train_losses += [_t_loss]
  val_losses += [_v_loss]
  fig, ax = plt.subplots(1,2)
fig.set_size_inches(20, 10)

ax[0].plot(train_accuries, 'b', label='train_acc')
ax[0].plot(val_accuries, 'g', label='val_acc')
ax[1].plot(train_losses, 'b', label='train_loss')
ax[1].plot(val_losses, 'g', label='val_loss')

ax[0].legend()
ax[1].legend()

fig.show()
```

Listing 1. Proposed Deep Learning model