

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

HOUSSAM FOUKI¹, KENZA BADDOU², AND ALEXANDER MAISURADZE³

¹houssam.fouki@student-cs.fr

²kenza.baddou@student-cs.fr

³alexander.maisuradze@student-cs.fr

Compiled January 20, 2023

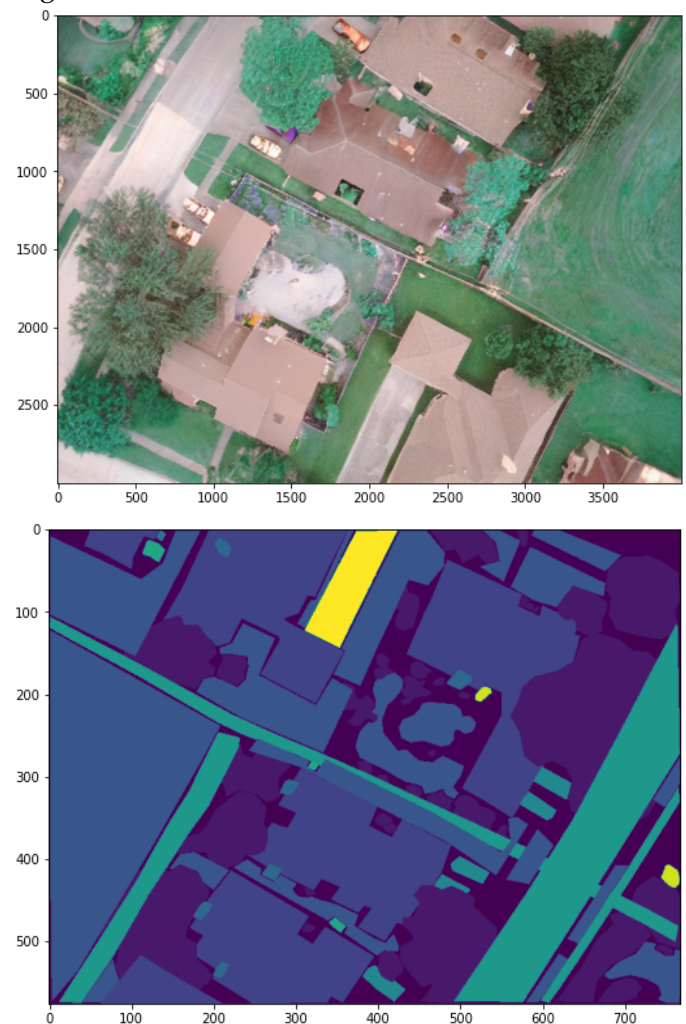
Problem Statement – The goal of this project is to design and implement a deep learning model that can automatically segment images. The images in question were acquired by a small unmanned aerial vehicle (sUAV) in the city of Houston, Texas, and are being used to assess the damage to residential and 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

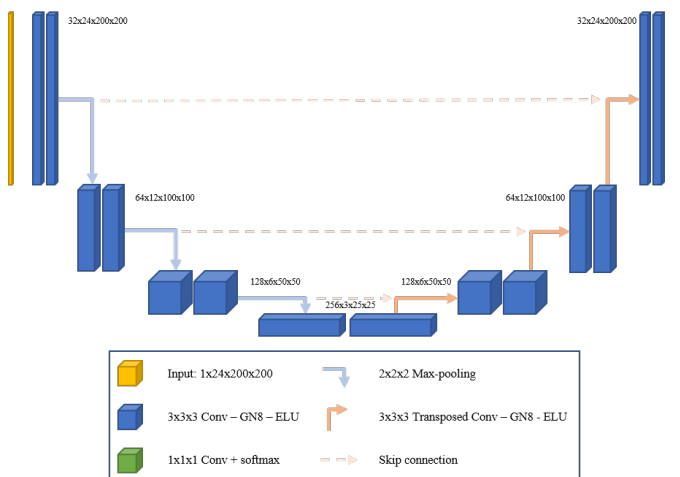
of 0.5 for enhancing the images through image augmentation.



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.

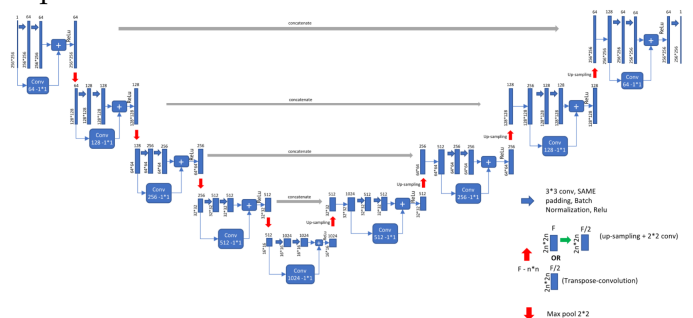
Architecture: U-NET architecture we implemented consists of two paths. The first path, known as the contraction path or encoder, is used to capture the context of the image. This path is made up of a traditional stack of convolutional and max pooling layers. The second path, known as the symmetric expanding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

```
1 mkdir Hurricane_Harvey
2 mkdir Hurricane_Harvey/rasters Hurricane_Harvey/vectors
3 gsutil -m cp -n -r gs://geoengine-dataset-houston-uav/rasters/
  raw Hurricane_Harvey/rasters/
4 gsutil -m cp -n -r gs://geoengine-dataset-houston-uav/vectors/
  random-split_-2022.11.17-22.35.45_ Hurricane_Harvey/vectors/
5 pip install segmentation-models-pytorch
```

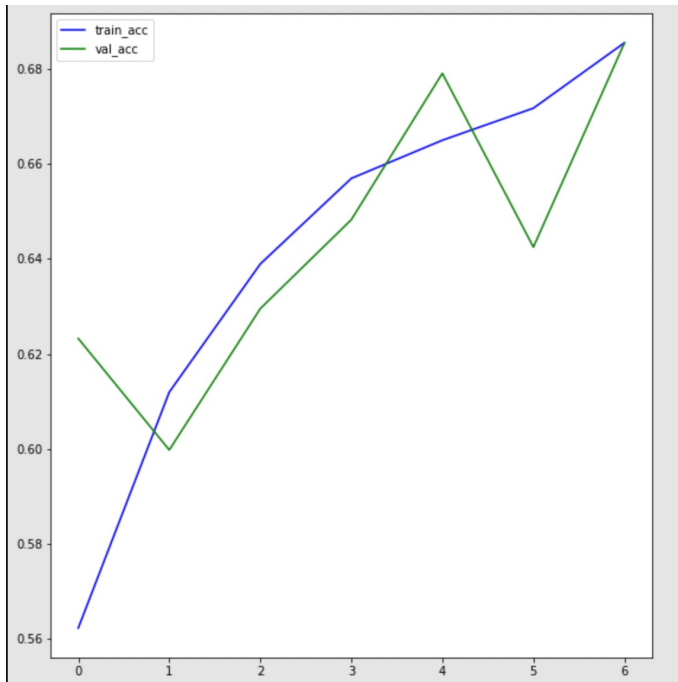


Fig. 1. Plot of accuracy in train and validation set (synthesize dataset)

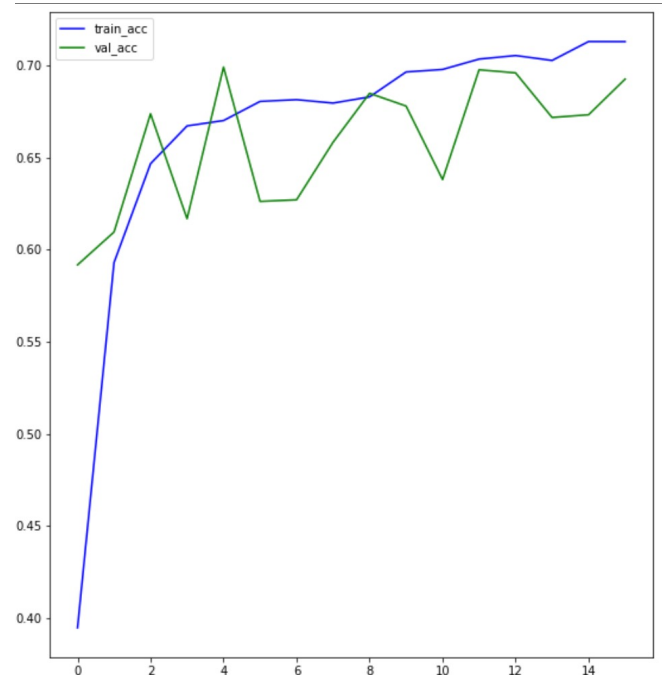


Fig. 3. Plot of accuracy in train and validation set (main dataset)

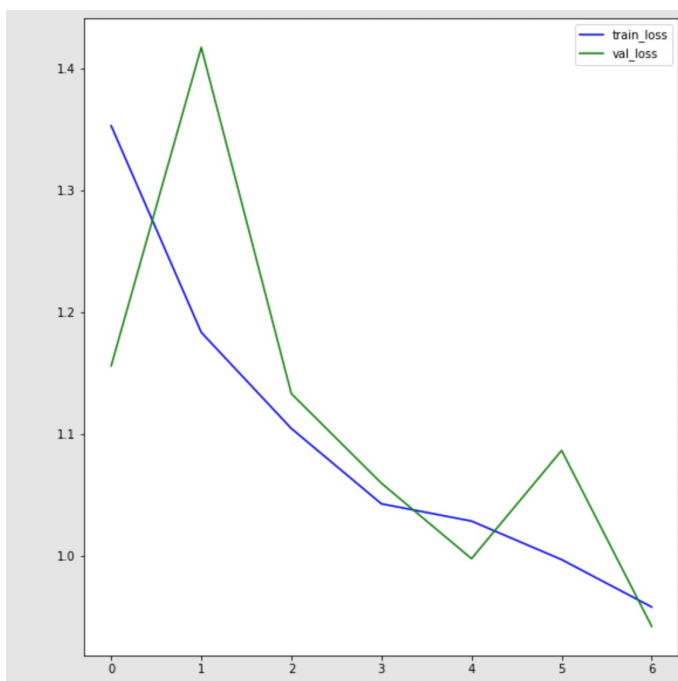


Fig. 2. Plot of loss in train and validation set (synthesize dataset)

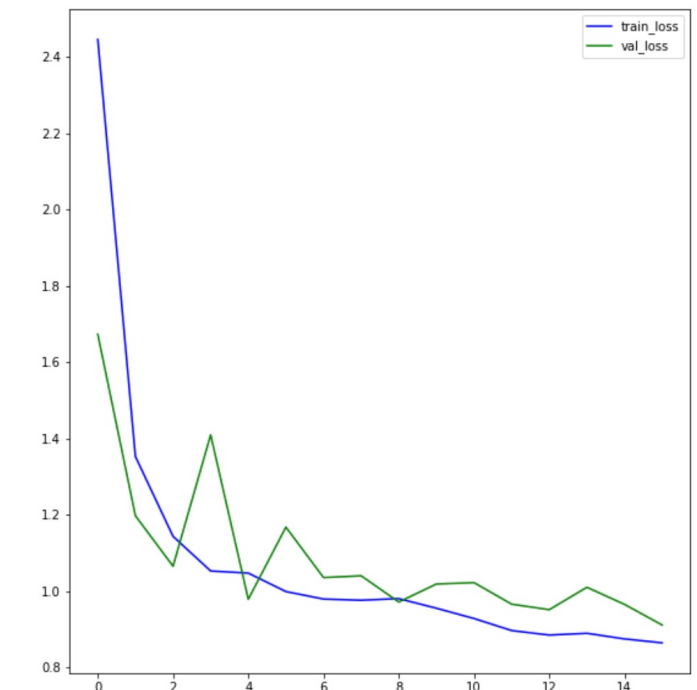


Fig. 4. Plot of loss in train and validation set (main dataset)

```

6 !pip install torchsummary
7 import os
8 import glob
9 import numpy as np
10 import matplotlib.pyplot as plt
11 import cv2
12 from PIL import Image
13 from tqdm.notebook import tqdm
14 from google.colab.patches import cv2_imshow
15 from tqdm import tqdm as progressbar
16 from sklearn.model_selection import train_test_split
17 from google.colab import drive
18 drive.mount('/content/drive')
19 import torch
20
21 def train_one_epoch(model, dataloader_train, dataloader_valid,
22                     optimizer, loss_function, epoch):
23     #training phase
24     model.train()
25     train_loss = 0
26     train_accuracy = 0
27     for imgs, masks in tqdm(dataloader_train):
28         optimizer.zero_grad()
29         imgs = imgs.to(DEVICE)
30         masks = masks.to(DEVICE)
31         # print(imgs.shape, masks.shape)
32         #forward pass
33         outputs = model(imgs)
34         #cal loss and backward
35         loss = loss_function(outputs, masks.type(torch.int64))
36         loss.backward()
37         optimizer.step()
38         train_loss += loss.item()
39         train_accuracy += pixel_accuracy(outputs, masks)
40     train_loss /= len(dataloader_train)
41     train_accuracy /= len(dataloader_train)
42
43     #validating phase
44     model.eval()
45     valid_loss = 0
46     val_accuracy = 0
47     with torch.no_grad():
48         for imgs, masks in tqdm(dataloader_valid):
49             imgs = imgs.to(DEVICE)
50             masks = masks.to(DEVICE)
51             outputs = model(imgs)
52             loss = loss_function(outputs, masks.type(torch.int64))
53             valid_loss += loss.item()
54             val_accuracy += pixel_accuracy(outputs, masks)
55     valid_loss /= len(dataloader_valid)
56     val_accuracy /= len(dataloader_valid)
57     print(f'EPOCH: {epoch + 1} - train loss: {train_loss} -
58           train_accuracy: {train_accuracy} - valid_loss: {valid_loss} -
59           val_accuracy: {val_accuracy}')
60     return train_loss, train_accuracy, valid_loss, val_accuracy
61
62 import albumentations.augmentations.functional as F
63 from torch.utils import data
64 import albumentations as A
65 from albumentations.pytorch import ToTensorV2
66
67 class HarveyDataset(data.Dataset):
68     def __init__(self, image_paths, mask_paths, transform=None):
69         self.image_paths = image_paths
70         self.mask_paths = mask_paths
71         self.transform = transform
72
73     def __len__(self):
74         return len(self.image_paths)
75
76     def __getitem__(self, _id):
77         image = np.array(Image.open(self.image_paths[_id]))
78         origin_mask = np.array(Image.open(self.mask_paths[_id]),
79                                         dtype=np.int64)
80
81         # print(f'Img: {os.path.basename(self.image_paths[_id])}\n
82               nMask: {os.path.basename(self.mask_paths[_id])}')
83
84         origin_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
85
86         if self.transform is not None:
87             transformed = self.transform(image=origin_image, mask
88                                         =origin_mask)
89             image = transformed["image"]
90             mask = transformed["mask"]
91         # return origin_image, origin_mask, image, mask
92         return image, mask
93
94 def model_accuracy(output, target):
95     # Transform the output to get the right format
96     # output_softmax = F.softmax(output, dim=1)
97     output_argmax = torch.argmax(output, dim=1)
98
99     # Get the correct predictions as a boolean mask
100     corrects = (output_argmax == target)
101
102     # Compute accuracy

```

```

96     accuracy = corrects.sum().float() / float(target.size(0) *
97         target.size(1) * target.size(2))
98
99     return accuracy * 100
100
101 def pixel_accuracy(output, mask):
102     with torch.no_grad():
103         output = torch.argmax(torch.nn.functional.softmax(output,
104             dim=1), dim=1)
105         correct = torch.eq(output, mask).int()
106         accuracy = float(correct.sum()) / float(correct.numel())
107     return accuracy
108
109 image_paths = glob.glob('/content/Hurricane_Harvey/rasters/raw/*.
110 tif')
111 mask_paths = glob.glob('/content/Hurricane_Harvey/vectors/random-
112 split-2022_11_17-22_35_45/Masks/*.png')
113
114 print(f'Len image: {len(image_paths)}, Len mask: {len(mask_paths)}')
115 temp = list(map(lambda x: os.path.basename(x)[-3:], mask_paths))
116
117 train_image_paths = list(filter(lambda x: os.path.basename(x)
118     [-3:] in temp, image_paths))
119 test_image_paths = list(set(image_paths) - set(train_image_paths))
120
121 assert len(train_image_paths) == len(temp), 'Len train path
122 should be the same to number mask'
123
124 print(f'Len train: {len(train_image_paths)}, Len test: {len(
125     test_image_paths)}')
126
127 train_image_paths = sorted(train_image_paths)
128 train_mask_paths = sorted(mask_paths)
129 X_train, X_val, y_train, y_val = train_test_split(
130     train_image_paths, train_mask_paths, test_size=0.1,
131     random_state=42)
132
133 print(f'Number train: {len(X_train)}')
134 print(f'Number train: {len(X_val)}')
135
136 train_transform = A.Compose(
137     [
138         A.Resize(576, 768),
139         A.HorizontalFlip(p=0.5),
140         A.VerticalFlip(p=0.5),
141
142         A.OneOf([
143             A.RandomBrightnessContrast(brightness_limit=0.3,
144                 contrast_limit=0.3, p=0.5),
145             A.Affine(translate_px={"x": (-20, 20), "y": (-20, 20)})
146         ],
147         p=0.5),
148         A.GaussNoise(p=0.5),
149         A.Affine(scale=(0.8, 1.2)),
150     ],
151     p=0.5)
152
153 val_transform = A.Compose(
154     [
155         A.Resize(576, 768),
156         A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229,
157             0.224, 0.225)),
158         ToTensorV2(),
159     ])
160
161 train_dataset = HarveyDataset(X_train, y_train, transform=
162     train_transform)
163 val_dataset = HarveyDataset(X_val, y_val, transform=val_transform)
164
165 origin_image, origin_mask, image, mask = train_dataset[np.random.
166     randint(len(y_train))]
167
168 figure, ax = plt.subplots(nrows=1, ncols=4, figsize=(40, 20))
169 ax[0].imshow(origin_image)
170 ax[1].imshow(origin_mask)
171 ax[2].imshow(image)
172 ax[3].imshow(mask)
173
174 DEVICE = 'cuda'
175 BATCH_SIZE=2
176
177 import segmentation_models_pytorch as smp
178 model = smp.UnetPlusPlus(
179     encoder_name='resnet101',
180     encoder_weights='imagenet',
181     # activation='sigmoid',
182     in_channels=3,
183     classes=27,
184     encoder_depth=5,
185     decoder_channels=[1024, 512, 256, 128, 64]
186 ).to(DEVICE)
187
188 train_loader = torch.utils.data.DataLoader(
189     train_dataset, batch_size=BATCH_SIZE, shuffle=True,
190     num_workers=2)
191
192 val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=
193     BATCH_SIZE, shuffle=False, num_workers=2)

```

```

173 !ls '/content/drive/MyDrive/transfer_checkpoints/'
174 !rm '/content/drive/MyDrive/tempi_checkpoints'/new_new_model*
175 max_lr = 1e-3
176 # epoch = 50
177 weight_decay = 1e-4
178 loss = torch.nn.CrossEntropyLoss()
179
180 optimizer = torch.optim.AdamW(model.parameters(), lr=max_lr,
181                                weight_decay=weight_decay)
182
183 max_score = 0
184 NUM_EPOCH = 200
185 hist_loss = []
186 for epoch in range(NUM_EPOCH):
187     print(f'=====Epoch: {epoch+1}')
```

```

188     ret = train_one_epoch(model, train_loader, val_loader,
189                           optimizer, loss, epoch+1)
189     hist_loss += [ret]
190     if ret[1]+ret[3] > max_score:
191         torch.save(model, f'./checkpoints/model3_{round(ret[1]*100)}_
192                        {round(ret[3]*100)}_{round(ret[2], 3)}.pt')
193     max_score = ret[1] + ret[3]
194     print(f'Save model: model3_{round(ret[1]*100)}_{round(ret
195           [3]*100)}_{round(ret[2], 3)}.pt')
196     train_losses = []
197     val_losses = []
198     for _train_loss, _val_loss in hist_loss:
199         train_losses += [_train_loss]
200         val_losses += [_val_loss]
201
202     plt.plot(train_losses)
203     plt.show()
204     plt.plot(val_losses)
205     plt.show()
206     !mkdir Hurricane_Harvey_Synthetic
207     !mkdir Hurricane_Harvey_Synthetic/rasters
208     Hurricane_Harvey_Synthetic/vectors
209     !gsutil -m cp -n -r gs://geoengine-dataset-houston-uav-synthetic/
210     rasters/raw Hurricane_Harvey_Synthetic/rasters/
211     !gsutil -m cp -n -r gs://geoengine-dataset-houston-uav-synthetic/
212     vectors/random-split_2022_11_21-11_59_40/
213     Hurricane_Harvey_Synthetic/vectors/
214
215 # create the masks from the json document
216 import json
217 import numpy as np
218 import os
219 from PIL import Image, ImageDraw
220 from tqdm.notebook import tqdm
221
222 def create_masks(json_folder, output_folder):
223     class_colormap = {}
224     for json_file in tqdm(os.listdir(json_folder)):
225         with open(json_folder + json_file) as f:
226             data = json.load(f)
227             try:
228                 image_width = data["images"][0]["width"]
229                 image_height = data["images"][0]["height"]
230                 image_name = data["images"][0]["name"]
231                 # print(f"image_name {image_name}")
232                 if f'{image_name}.png' in os.listdir(output_folder):
233                     print("\t image already saved")
234                     continue
235                 mask = Image.new('L', (image_width, image_height))
236                 for annotation in data["annotations"]:
237                     segmentation = annotation["segmentation"][0]
238                     segmentation = [[segmentation[i], segmentation[i
239 +1]] for i in range(0, len(segmentation) - 2, 2)]
240                     label = annotation["properties"][00]["labels"][0]
241                     # check if this label has been seen before
242                     if label not in class_colormap:
243                         # choose a new grayscale value for this label
244                         class_colormap[label] = len(class_colormap)
245                     ImageDraw.Draw(mask).polygon(segmentation, fill=
246 class_colormap[label])
247                     mask.save(os.path.join(output_folder, f'{image_name}.
248 png'))
249             except KeyError as e:
250                 print(f"Exception {e} raised for {json_file}")
251                 !mkdir '/content/Hurricane_Harvey_Synthetic/vectors/
252 random-split_2022_11_21-11_59_40/Masks/'
253
254 json_folder = "/content/Hurricane_Harvey_Synthetic/vectors/random
255 -split_2022_11_21-11_59_40/COCO/"
256 output_folder = "/content/Hurricane_Harvey_Synthetic/vectors/
257 random-split_2022_11_21-11_59_40/Masks/"
258 create_masks(json_folder, output_folder)
259 image_paths = glob.glob('/content/Hurricane_Harvey_Synthetic/
260 rasters/raw/*.tif')
261 mask_paths = glob.glob('/content/Hurricane_Harvey_Synthetic/
262 vectors/random-split_2022_11_21-11_59_40/Masks/*.png')
263
264 print(f'Len image: {len(image_paths)}, Len mask: {len(mask_paths)}
265 ')
266 temp = list(map(lambda x: os.path.basename(x)[-3:], mask_paths))
267

```

```

252 train_image_paths = list(filter(lambda x: os.path.basename(x)
253                               [: -3] in temp, image_paths))
254 test_image_paths = list(set(image_paths) - set(train_image_paths))
255
256 assert len(train_image_paths) == len(temp), 'Len train path
257        should be the same to number mask'
258
259 print(f'Len train: {len(train_image_paths)}, Len test: {len(
260     test_image_paths)}')
261 train_image_paths = sorted(train_image_paths)
262 train_mask_paths = sorted(mask_paths)
263
264 X_train, X_val, y_train, y_val = train_test_split(
265     train_image_paths, train_mask_paths, test_size=0.1,
266     random_state=42)
267
268 print(f'Number train: {len(X_train)}')
269 print(f'Number train: {len(X_val)}')
270
271 train_transform = A.Compose(
272     [
273         # A.Resize(224, 224),
274         A.Resize(576, 768),
275         A.HorizontalFlip(p=0.5),
276         A.VerticalFlip(p=0.5),
277
278         A.OneOf([
279             A.RandomBrightnessContrast(brightness_limit=0.3,
280             contrast_limit=0.3, p=0.5),
281             A.Affine(translate_px={"x": (-20, 20), "y": (-20, 20)})
282         ],
283         A.GaussNoise(p=0.5),
284         A.Affine(scale=(0.8, 1.2))),
285     ],
286     A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229,
287     0.224, 0.225)),
288     ToTensorV2(),
289 )
290
291 val_transform = A.Compose(
292     [
293         A.Resize(576, 768),
294         # A.Resize(224, 224),
295         A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229,
296     0.224, 0.225)),
297         ToTensorV2(),
298     ]
299 )
300
301 train_dataset = HarveyDataset(X_train, y_train, transform=
302     train_transform)
303 val_dataset = HarveyDataset(X_val, y_val, transform=val_transform)
304
305 origin_image, origin_mask, image, mask = train_dataset[np.random.
306     randint(len(y_train))]
307 figure, ax = plt.subplots(nrows=1, ncols=4, figsize=(40, 20))
308 ax[0].imshow(origin_image)
309 ax[1].imshow(origin_mask)
310 ax[2].imshow(image)
311 ax[3].imshow(mask)
312
313 import segmentation_models_pytorch as smp
314
315 # model = smp.Unet('resnet50', encoder_weights=None, classes=27,
316     in_channels=3, activation='softmax', encoder_depth=5,
317     decoder_channels=[512,256,128,64,32]).to(DEVICE)
318
319 # model = smp.Unet('resnet101', encoder_weights='imagenet',
320     classes=27, in_channels=3, activation=None, encoder_depth=5,
321     decoder_channels=[1024, 512,256,128,64]).to(DEVICE)
322
323 DEVICE = 'cuda'
324 BATCH_SIZE=2
325
326 # model = smp.UnetPlusPlus(
327     # encoder_name='resnet101',
328     # encoder_weights='imagenet',
329     # activation='sigmoid',
330     # in_channels=3,
331     # classes=27,
332     # encoder_depth=5,
333     # decoder_channels=[1024, 512,256,128,64]
334     ).to(DEVICE)
335
336 model = torch.load('/content/drive/MyDrive/model1_49_57_1.326.pt')
337
338 train_loader = torch.utils.data.DataLoader(train_dataset,
339     batch_size=BATCH_SIZE, shuffle=True, num_workers=2)
340 val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=
341     BATCH_SIZE, shuffle=False, num_workers=2)
342
343 max_lr = 1e-3
344 # epoch = 50
345 loss = torch.nn.CrossEntropyLoss()
346 weight_decay = 1e-4
347
348 optimizer = torch.optim.AdamW(model.parameters(), lr=max_lr,
349     weight_decay=weight_decay)

```



```

328 max_score = 0
329 NUM_EPOCH = 200
330 hist_loss = []
331 for epoch in range(NUM_EPOCH):
332     print(f'=====Epoch: {epoch+1}')
333     ret = train_one_epoch(model, train_loader, val_loader,
334                           optimizer, loss, epoch+1)
335     hist_loss += [ret]
336     if ret[1]+ret[3] > max_score:
337         torch.save(model, f'/content/drive/MyDrive/
338                           transfer_checkpoints/model1_{round(ret[1]*100)}_{round(ret
339                               [3]*100)}_{round(ret[2], 3)}.pt')
340         max_score = ret[1] + ret[3]
341         print(f'Save model: model1_{round(ret[1]*100)}_{round(ret
342             [3]*100)}_{round(ret[2], 3)}.pt')
343         train_accuries = []
344     val_accuries = []
345     train_losses = []
346     val_losses = []
347     for _t_loss, _t_acc, _v_loss, _v_acc in hist_loss:
348         train_accuries += [_t_acc]
349         val_accuries += [_v_acc]
350         train_losses += [_t_loss]
351         val_losses += [_v_loss]
352         fig, ax = plt.subplots(1,2)
353         fig.set_size_inches(20, 10)
354         ax[0].plot(train_accuries, 'b', label='train_acc')
355         ax[0].plot(val_accuries, 'g', label='val_acc')
356         ax[1].plot(train_losses, 'b', label='train_loss')
357         ax[1].plot(val_losses, 'g', label='val_loss')
358         ax[0].legend()
359         ax[1].legend()
360     fig.show()
361     !rm results/*
362     # temp_model = torch.load('/content/model_34_88.pt')
363     temp_model = model
364     temp_model.eval()
365
366     with torch.no_grad():
367         for _path in tqdm(test_image_paths):
368             fname = os.path.basename(_path).replace('tif', 'png')
369             image = np.array(Image.open(_path))
370             origin_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
371             transformed = val_transform(image=origin_image)
372             image = transformed["image"]
373             h, w = origin_image.shape[:2]
374
375             t = torch.unsqueeze(image, 0).to(DEVICE)
376             output = temp_model(t)
377             output = torch.argmax(torch.nn.functional.softmax(output, dim
378                 =1), dim=1).detach().cpu().numpy()[0]
379
380             output = cv2.resize(output, (w,h), interpolation=cv2.
381                 INTER_NEAREST)
382             cv2.imwrite(f'results/{fname}', output)
383
384     import tarfile
385     import os
386
387     tar = tarfile.open("submission.tar", "w")
388
389     for root, dir, files in os.walk('/content/results/'):
390         for file in files:
391             fullpath = os.path.join(root, file)
392             tar.add(fullpath, arcname=file)
393
394     tar.close()
395     !cp submission.tar '/content/drive/MyDrive'
396     !len(os.listdir('results'))
397     model.eval()
398     with torch.no_grad():
399         t = torch.unsqueeze(image, 0).to(DEVICE)
400         output = model(t)
401         output = torch.argmax(torch.nn.functional.softmax(output, dim
402             =1), dim=1).detach().cpu().numpy()[0]
403
404     figure, ax = plt.subplots(nrows=1, ncols=2, figsize=(40, 20))
405     ax[0].imshow(cv2.resize(origin_image, (640,480)))
406     ax[1].imshow(output[0])
407     !rm ./*.pt
408     train_transform = A.Compose(
409         [
410             # A.Resize(352, 352),
411             A.Resize(576, 768),
412             A.HorizontalFlip(p=0.5),
413             A.VerticalFlip(p=0.5),
414             A.RandomBrightnessContrast(brightness_limit=0.3,
415                 contrast_limit=0.3, p=0.3),
416             A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229,
417                 0.224, 0.225)),
418             ToTensorV2(),

```

```

416 ]
417 )
418
419 val_transform = A.Compose(
420     [
421         A.Resize(576, 768),
422         # A.Resize(352, 352),
423         A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229,
424             0.224, 0.225)),
425         ToTensorV2(),
426     ]
427 )
428 max_lr = 1e-3
429 # epoch = 50
430 weight_decay = 1e-4
431
432 optimizer = torch.optim.AdamW(model.parameters(), lr=max_lr,
433     weight_decay=weight_decay)
434 loss = torch.nn.CrossEntropyLoss()
435
436 max_score = 0
437 NUM_EPOCH = 200
438 hist_loss = []
439 for epoch in range(NUM_EPOCH):
440     print(f'=====Epoch: {epoch+1}')
441     X_train, X_val, y_train, y_val = train_test_split(
442         train_image_paths, train_mask_paths, test_size=0.2)
443
444     train_dataset = HarveyDataset(X_train, y_train, transform=
445         train_transform)
446     val_dataset = HarveyDataset(X_val, y_val, transform=
447         val_transform)
448
449     train_loader = torch.utils.data.DataLoader(train_dataset,
450         batch_size=BATCH_SIZE, shuffle=True, num_workers=2)
451     val_loader = torch.utils.data.DataLoader(val_dataset,
452         batch_size=BATCH_SIZE, shuffle=False, num_workers=2)
453
454     ret = train_one_epoch(model, train_loader, val_loader,
455                           optimizer, loss, epoch+1)
456     hist_loss += [ret]
457     if ret[1]+ret[3] > max_score:
458         torch.save(model, f'/content/drive/MyDrive/temp1_checkpoints/
459             model_{round(ret[1]*100)}_{round(ret[3]*100)}.pt')
460         max_score = ret[1]+ret[3]
461         print(f'Save model: model_{round(ret[1]*100)}_{round(ret
462             [3]*100)}.pt')
463
464     del train_dataset
465     del val_dataset
466     del train_loader
467     del val_loader
468     train_accuries = []
469     val_accuries = []
470     train_losses = []
471     val_losses = []
472     for _t_loss, _t_acc, _v_loss, _v_acc in hist_loss:
473         train_accuries += [_t_acc]
474         val_accuries += [_v_acc]
475         train_losses += [_t_loss]
476         val_losses += [_v_loss]
477         fig, ax = plt.subplots(1,2)
478         fig.set_size_inches(20, 10)
479
480         ax[0].plot(train_accuries, 'b', label='train_acc')
481         ax[0].plot(val_accuries, 'g', label='val_acc')
482         ax[1].plot(train_losses, 'b', label='train_loss')
483         ax[1].plot(val_losses, 'g', label='val_loss')
484
485         ax[0].legend()
486         ax[1].legend()
487
488     fig.show()

```

Listing 1. Proposed Deep Learning model