U-Net-architecture

March 5, 2023

[26]:
```
import torch
import torchvision
from torchvision import models
import torchvision.transforms as T

import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

print('pytorch', torch.__version__)
print('torchvision', torchvision.__version__)
```

```
pytorch 1.13.1
torchvision 0.14.1
```

[18]:
```
path_data = Path('SegTMS/train/')
path_lbl = path_data/'Labels'
path_img = path_data/'Images'
```

[44]:
```
IMG_SIZE = (256,256)

COLORS = np.array([
    (0, 0, 0),       # 0=background
    (128, 0, 0),     # 1=aeroplane
    (0, 128, 0),     # 2=bicycle
    (128, 128, 0),   # 3=bird
    (0, 0, 128),     # 4=boat
    (128, 0, 128),   # 5=bottle
    (0, 128, 128),   # 6=bus
    (128, 128, 128), # 7=car
    (255, 255, 255), # 8=cat
    (192, 0, 0),     # 9=chair
    (64, 128, 0),    # 10=cow
    (192, 128, 0),   # 11=dining table
    (64, 0, 128),    # 12=dog
    (192, 0, 128),   # 13=horse
    (64, 128, 128),  # 14=motorbike
    (192, 128, 128), # 15=person
```

```
    (0, 64, 0),       # 16=potted plant
    (128, 64, 0),     # 17=sheep
    (0, 192, 0),      # 18=sofa
    (128, 192, 0),    # 19=train
    (0, 64, 128)      # 20=tv/monitor
])
```

[45]:
```
#Download Model
deeplab = models.segmentation.deeplabv3_resnet101(pretrained=True).eval()
```

[46]:
```
#Load Image
img = Image.open('SegTMS/train/Images/img3.png')

plt.figure(figsize=(16, 16))
plt.imshow(img)
plt.axis('off')
```

[46]: (-0.5, 3839.5, 2159.5, -0.5)

[52]:
```
#Image to Tensor
trf = T.Compose([
    T.Resize(IMG_SIZE),
    T.CenterCrop(IMG_SIZE), # make square image
    T.ToTensor(),
    T.Normalize(
        mean=[0.485, 0.456, 0.406],
```

```
        std=[0.229, 0.224, 0.225]
    )
])

input_img = trf(img).unsqueeze(0)
```

[53]:
```
out = deeplab(input_img)['out']

print(out.shape)
```

```
torch.Size([1, 21, 256, 256])
```

[54]:
```
#Extract Class Map

out = torch.argmax(out.squeeze(), dim=0)
out = out.detach().cpu().numpy()

print(out.shape)
print(np.unique(out))
```

```
(256, 256)
[0]
```

[55]:
```
def seg_map(img, n_classes=21):
    rgb = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)

    for c in range(n_classes):
        idx = img == c

        rgb[idx] = COLORS[c]

    return rgb
```

[88]:
```
#Class Map to Segmentation Map

out_seg = seg_map(out)

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(16, 16))
ax[0].imshow(img)
ax[0].axis('off')
ax[1].imshow(out_seg)
ax[1].axis('off')
```

[88]:
```
(-0.5, 454.5, 255.5, -0.5)
```

[202]:
```
##--------------------Semantic Segmentation-Pytorch
 ↪unet----------------------------#
```

[281]:
```
from torch_snippets import *
from torchvision import transforms
from sklearn.model_selection import train_test_split
import cv2
import glob
from torchsummary import summary
import segmentation_models_pytorch as smp
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms as T
import torchvision
import torch.nn.functional as F
from torch.autograd import Variable
from torch.utils.data import  DataLoader
from transformers import SegformerFeatureExtractor
from transformers import SegformerForSemanticSegmentation
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

[282]:
```
tfms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) #
 ↪imagenet
])
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

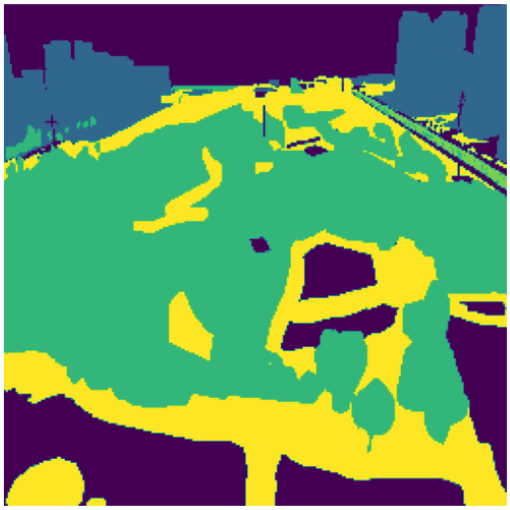
[283]:
```
#Batch generation
labels=sorted(glob.glob('SegTMS/train/Labels/*.png'))
images=sorted(glob.glob('SegTMS/train/Images/*.png'))
train_images=images[:int(0.8*(len(images)))]
train_labels=labels[:int(0.8*(len(labels)))]
test_images=images[int(0.8*(len(images))):]
```

```
test_label=labels[int(0.8*(len(labels))):]
```

[284]:
```
w=256
h=256

class SegData(Dataset):
    def __init__(self,images,labels):
        self.images=images
        self.labels=labels
    def __len__(self):
        return len(self.images)
    def __getitem__(self, ix):
        image = read(self.images[ix], 1)
        image = cv2.resize(image, (w,h))
        mask = read(self.labels[ix],1)
        mask = cv2.resize(mask, (w,h))
        mask=cv2.cvtColor(mask,cv2.COLOR_RGB2GRAY)
        return image, mask
    def choose(self): return self[randint(len(self))]
    def collate_fn(self, batch):
        ims, masks = list(zip(*batch))
        ims = torch.cat([tfms(im.copy()/255.)[None] for im in ims]).float().
 ↪to(device)
        ce_masks = torch.cat([torch.tensor(mask[None]) for mask in masks]).
 ↪long().to(device)
        return ims, ce_masks
```

[285]:
```
train_dataset=SegData(train_images,train_labels)
test_dataset=SegData(test_images,test_label)
```

[286]:
```
image,mask=train_dataset[-1]
plt.figure(figsize = (14, 10))
plt.subplot(1,2,1)
plt.axis('off')
plt.imshow(np.array(image))
plt.subplot(1,2,2)
plt.imshow(np.array(mask).squeeze())
plt.axis('off')
plt.show()
print(image.shape)
print(mask.shape)
print(mask.max())
```



```
(256, 256, 3)
```

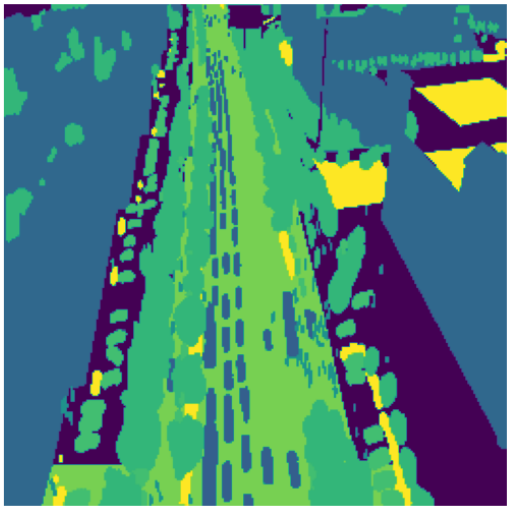
```
(256, 256)
```

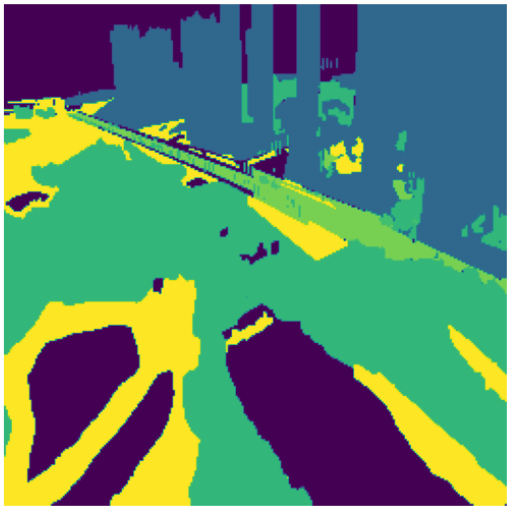
```
113
```

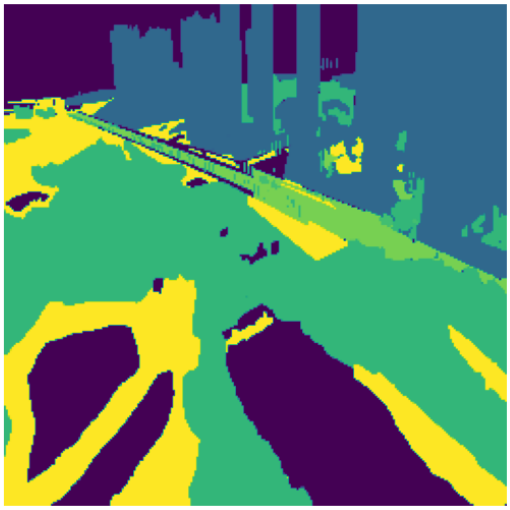
[287]:
```
nbatch_size=16
train_loader = torch.utils.data.DataLoader(train_dataset,␣
 ↪batch_size=nbatch_size, shuffle=True, num_workers=0)
test_loader=torch.utils.data.
 ↪DataLoader(test_dataset,batch_size=nbatch_size,shuffle=True, num_workers=0)
```

[288]:
```
# lets look at some samples
image, mask = train_dataset[0]
plt.figure(figsize = (14, 10))
plt.subplot(1,2,1)
plt.axis('off')
plt.imshow(np.array(image))
plt.subplot(1,2,2)
plt.imshow(np.array(mask).squeeze())
plt.axis('off')
plt.show()

image, mask = test_dataset[0]

plt.figure(figsize = (14, 10))
plt.subplot(1,2,1)
plt.axis('off')
```

```
plt.imshow(np.array(image))
plt.subplot(1,2,2)
plt.imshow(np.array(mask).squeeze())
plt.axis('off')
plt.show()

image, mask = test_dataset[0]

plt.figure(figsize = (14, 10))
plt.subplot(1,2,1)
plt.imshow(np.array(image))
plt.axis('off')
plt.subplot(1,2,2)
plt.imshow(np.array(mask).squeeze())
plt.axis('off')
plt.show()
```





[293]:
```
train_loader =train_loader
val_loader=test_loader
print('Train Size   : ', len(train_dataset))
print('Val Size     : ', len(test_dataset))
```

```
Train Size   :  160


Val Size     :  40
```

[ ]:
```
##---------------------Semantic Segmentation-TF
 ↪unet----------------------------#
```

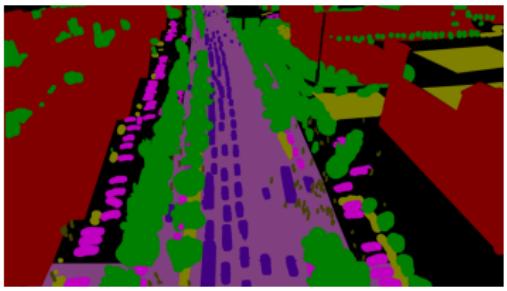
[296]:
```
import matplotlib.pyplot as plt
import imageio.v2 as imageio
import tensorflow as tf

import numpy as np

from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Conv2DTranspose
from tensorflow.keras.layers import concatenate
```

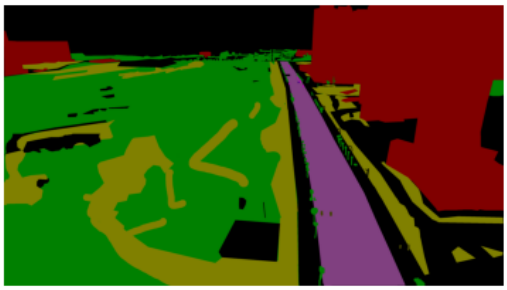
[308]:
```
inp_dir = 'SegTMS/train/Images/'
mask_dir = 'SegTMS/train/Labels/'

inp_list = os.listdir(inp_dir)
mask_list=os.listdir(mask_dir)
inp_list.sort()
mask_list.sort()
inp_img_list = [os.path.join(inp_dir,i) for i in inp_list]
mask_list = [os.path.join(mask_dir,i) for i in mask_list]
```

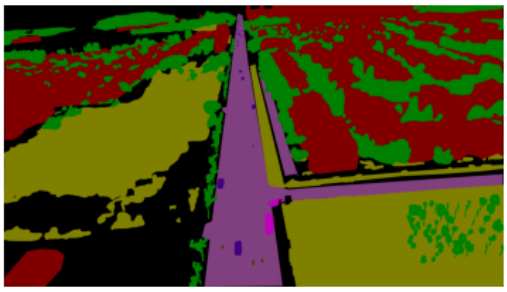
[309]:
```
def plt_image_and_mask_by_path(img_list,mask_lst,n=3):
    for i in range(3):
        img1=imageio.imread(img_list[i])
        mask1=imageio.imread(mask_lst[i])
        _,arr = plt.subplots(1,2,figsize=(10,10))
        arr[0].imshow(img1)
        arr[0].axis('off')
        arr[1].imshow(mask1)
        arr[1].axis('off')
```

[310]:
```
def plt_image_and_mask_with_dataset(image_ds,mask_ds):
    _,arr = plt.subplots(1,2,figsize=(10,10))
    arr[0].imshow(tf.keras.preprocessing.image.array_to_img(image_ds))
    arr[0].axis('off')
    arr[1].imshow(tf.keras.preprocessing.image.array_to_img(mask_ds))
    arr[1].axis('off')
```

[312]:
```
plt_image_and_mask_by_path(inp_img_list,mask_list,3)
```



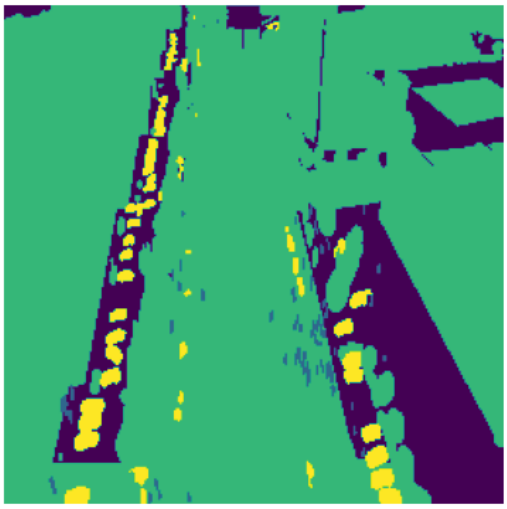


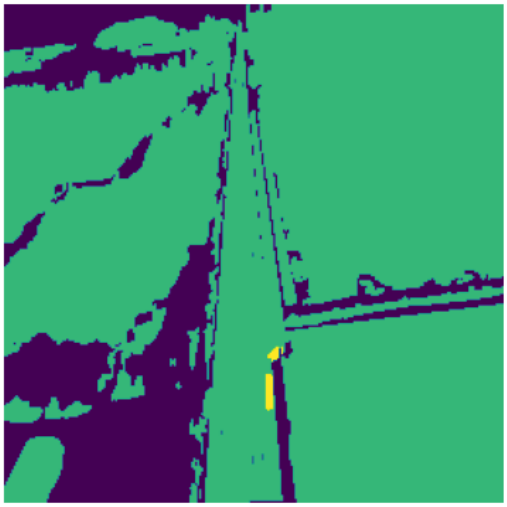
[313]:
```
# create images paths as tensors of file paths
image_names  = tf.constant(inp_img_list)
mask_names = tf.constant(mask_list)

dataset = tf.data.Dataset.from_tensor_slices((image_names,mask_names))
for image,mask in dataset.take(1):
    print(image)
    print(mask)
```

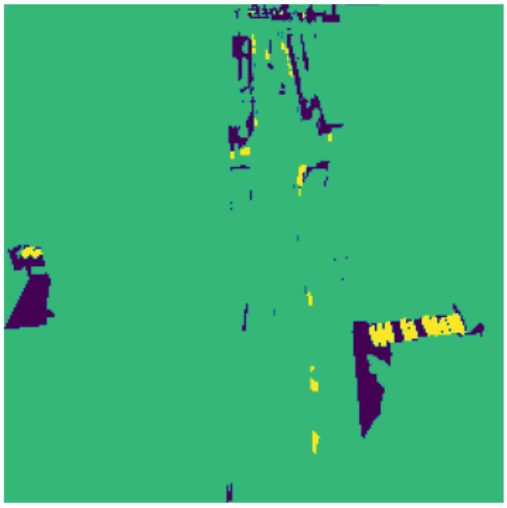
```
tf.Tensor(b'SegTMS/train/Images/img1.png', shape=(), dtype=string)


tf.Tensor(b'SegTMS/train/Labels/mask1.png', shape=(), dtype=string)
```

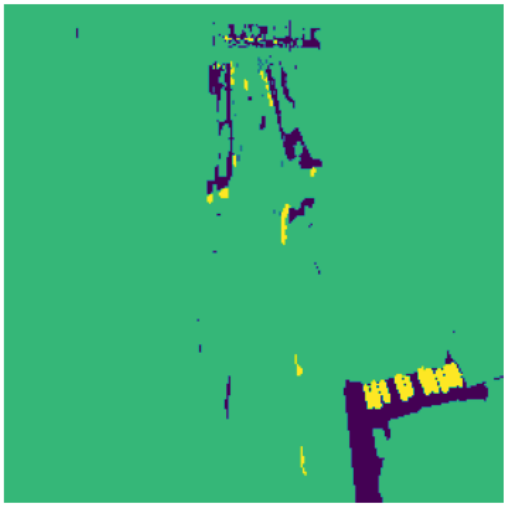
[328]:
```
def process_path(image_,mask_):
    img = tf.io.read_file(image_)
    img = tf.image.decode_png(img,channels=3)
    img = tf.image.convert_image_dtype(img,tf.float32)

    mask = tf.io.read_file(mask_)
    mask = tf.image.decode_png(mask,channels=3)
    mask = tf.math.reduce_max(mask,axis=-1,keepdims=True)
    return img,mask

def process(image,mask):
    inp_image = tf.image.resize(image,(w,h),method='nearest')
    inp_mask = tf.image.resize(mask,(w,h),method='nearest')
    return inp_image,inp_mask

image_ds = dataset.map(process_path)
processed_images_ds = image_ds.map(process)
```

[330]:
```
for image,mask in processed_images_ds.take(5):
    plt_image_and_mask_with_dataset(image,mask)
```









[331]:
```
def conv_block(inputs=None, n_filters=64, dropout_prob=0, max_pooling=True):
    conv = Conv2D(n_filters, # Number of filters
                  kernel_size=3,   # Kernel size
                  activation='relu',
                  padding='same',
                  kernel_initializer='he_normal')(inputs)
    conv = Conv2D(n_filters, # Number of filters
                  3,   # Kernel size
                  activation='relu',
                  padding='same',
                  kernel_initializer='he_normal')(conv)
```

```
    if dropout_prob > 0:
        conv = Dropout(dropout_prob)(conv)

    if max_pooling:
        next_layer = MaxPooling2D(pool_size=(2,2))(conv)

    else:
        next_layer = conv

    skip_connection = conv

    return next_layer, skip_connection
```

[332]:
```
def upsampling_block(expansive_input, contractive_input, n_filters=64):
    up = Conv2DTranspose(
                 filters=n_filters,    # number of filters
                 kernel_size=(3,3),    # Kernel size
                 strides=(2,2),
                 padding='same')(expansive_input)

    merge = concatenate([up, contractive_input], axis=3)
    conv = Conv2D(filters=n_filters,   # Number of filters
                 kernel_size=(3,3),     # Kernel size
                 activation='relu',
                 padding='same',
                 kernel_initializer='he_normal')(merge)
    conv = Conv2D(n_filters,  # Number of filters
                 kernel_size=(3,3),   # Kernel size
                 activation='relu',
                 padding='same',
                 kernel_initializer='he_normal')(conv)

    return conv
```

[333]:
```
def unet_model(input_size=(w, h, 3), n_filters=64, n_classes=256):

    inputs = Input(input_size)
    cblock1 = conv_block(inputs, n_filters)
    cblock2 = conv_block(cblock1[0], n_filters*2)
    cblock3 = conv_block(cblock2[0], n_filters*4)
    cblock4 = conv_block(cblock3[0], n_filters*8, dropout_prob=0.3)
    cblock5 = conv_block(cblock4[0], n_filters*16, dropout_prob=0.3,␣
 ↪max_pooling=False)

    ublock6 = upsampling_block(cblock5[0], cblock4[1],  n_filters*8)
    ublock7 = upsampling_block(ublock6, cblock3[1],  n_filters*4)
    ublock8 = upsampling_block(ublock7, cblock2[1],  n_filters*2)
```

```
    ublock9 = upsampling_block(ublock8, cblock1[1],  n_filters)

    conv9 = Conv2D(n_filters,
                   3,
                   activation='relu',
                   padding='same',
                   kernel_initializer='he_normal')(ublock9)

    conv10 = Conv2D(n_classes, 1, padding='same')(conv9)
    model = tf.keras.Model(inputs=inputs, outputs=conv10)

    return model
```

[334]:
```
h=256
w=256
chs = 3
unet = unet_model((h, w, chs))
```

[335]:
```
unet.summary()
```

```
Model: "model_3"
__________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==========================================================================================
 input_4 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 conv2d_60 (Conv2D)             (None, 256, 256, 64  1792
['input_4[0][0]']
                                )

 conv2d_61 (Conv2D)             (None, 256, 256, 64  36928
['conv2d_60[0][0]']
                                )

 max_pooling2d_12 (MaxPooling2D  (None, 128, 128, 64  0
['conv2d_61[0][0]']
 )                              )

 conv2d_62 (Conv2D)             (None, 128, 128, 12  73856
['max_pooling2d_12[0][0]']
                                8)

 conv2d_63 (Conv2D)             (None, 128, 128, 12  147584
['conv2d_62[0][0]']
```

```
                                8)

 max_pooling2d_13 (MaxPooling2D  (None, 64, 64, 128)  0
['conv2d_63[0][0]']
 )

 conv2d_64 (Conv2D)             (None, 64, 64, 256)  295168
['max_pooling2d_13[0][0]']

 conv2d_65 (Conv2D)             (None, 64, 64, 256)  590080
['conv2d_64[0][0]']

 max_pooling2d_14 (MaxPooling2D  (None, 32, 32, 256)  0
['conv2d_65[0][0]']
 )

 conv2d_66 (Conv2D)             (None, 32, 32, 512)  1180160
['max_pooling2d_14[0][0]']

 conv2d_67 (Conv2D)             (None, 32, 32, 512)  2359808
['conv2d_66[0][0]']

 dropout_6 (Dropout)            (None, 32, 32, 512)  0
['conv2d_67[0][0]']

 max_pooling2d_15 (MaxPooling2D  (None, 16, 16, 512)  0
['dropout_6[0][0]']
 )

 conv2d_68 (Conv2D)             (None, 16, 16, 1024  4719616
['max_pooling2d_15[0][0]']
                                )

 conv2d_69 (Conv2D)             (None, 16, 16, 1024  9438208
['conv2d_68[0][0]']
                                )

 dropout_7 (Dropout)            (None, 16, 16, 1024  0
['conv2d_69[0][0]']
                                )

 conv2d_transpose_12 (Conv2DTra  (None, 32, 32, 512)  4719104
['dropout_7[0][0]']
 nspose)

 concatenate_12 (Concatenate)   (None, 32, 32, 1024  0
['conv2d_transpose_12[0][0]',
                                )
```

```
'dropout_6[0][0]']

 conv2d_70 (Conv2D)             (None, 32, 32, 512)  4719104
['concatenate_12[0][0]']

 conv2d_71 (Conv2D)             (None, 32, 32, 512)  2359808
['conv2d_70[0][0]']

 conv2d_transpose_13 (Conv2DTra  (None, 64, 64, 256)  1179904
['conv2d_71[0][0]']
 nspose)

 concatenate_13 (Concatenate)   (None, 64, 64, 512)  0
['conv2d_transpose_13[0][0]',
'conv2d_65[0][0]']

 conv2d_72 (Conv2D)             (None, 64, 64, 256)  1179904
['concatenate_13[0][0]']

 conv2d_73 (Conv2D)             (None, 64, 64, 256)  590080
['conv2d_72[0][0]']

 conv2d_transpose_14 (Conv2DTra  (None, 128, 128, 12  295040
['conv2d_73[0][0]']
 nspose)                        8)

 concatenate_14 (Concatenate)   (None, 128, 128, 25  0
['conv2d_transpose_14[0][0]',
                                6)
'conv2d_63[0][0]']

 conv2d_74 (Conv2D)             (None, 128, 128, 12  295040
['concatenate_14[0][0]']
                                8)

 conv2d_75 (Conv2D)             (None, 128, 128, 12  147584
['conv2d_74[0][0]']
                                8)

 conv2d_transpose_15 (Conv2DTra  (None, 256, 256, 64  73792
['conv2d_75[0][0]']
 nspose)                        )

 concatenate_15 (Concatenate)   (None, 256, 256, 12  0
['conv2d_transpose_15[0][0]',
                                8)
'conv2d_61[0][0]']
```

```
 conv2d_76 (Conv2D)             (None, 256, 256, 64  73792
['concatenate_15[0][0]']
                                )

 conv2d_77 (Conv2D)             (None, 256, 256, 64  36928
['conv2d_76[0][0]']
                                )

 conv2d_78 (Conv2D)             (None, 256, 256, 64  36928
['conv2d_77[0][0]']
                                )

 conv2d_79 (Conv2D)             (None, 256, 256, 25  16640
['conv2d_78[0][0]']
                                6)

==================================================================================
==================
Total params: 34,566,848
Trainable params: 34,566,848
Non-trainable params: 0
----------------------------------------------------------------------------------
------------------
```

[336]:
```
unet.compile(optimizer='adam',
             loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
             metrics=['accuracy'])
```

[ ]:
```
EPOCHS = 10
VAL_SUBSPLITS = 5
BUFFER_SIZE = 500
BATCH_SIZE = 8
processed_images_ds.batch(BATCH_SIZE)
train_dataset = processed_images_ds.cache().shuffle(BUFFER_SIZE).
 ↪batch(BATCH_SIZE)
print(processed_images_ds.element_spec)
model_history = unet.fit(train_dataset, epochs=EPOCHS)
```

```
(
    TensorSpec(shape=(256, 256, 3), dtype=tf.float32, name=None),
    TensorSpec(shape=(256, 256, 1), dtype=tf.uint8, name=None)
)


Epoch 1/10

2023-03-05 04:26:24.133550: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:392] Filling up shuffle
```

```
buffer (this may take a while): 55 of 500
2023-03-05 04:26:34.459327: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:392] Filling up shuffle
buffer (this may take a while): 120 of 500
2023-03-05 04:26:44.536709: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:392] Filling up shuffle
buffer (this may take a while): 177 of 500
2023-03-05 04:26:48.136355: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:417] Shuffle buffer filled.

25/25 [==============================] - 809s 30s/step - loss: 1.6286 -
accuracy: 0.6886
Epoch 2/10
25/25 [==============================] - 743s 30s/step - loss: 0.6234 -
accuracy: 0.8106
Epoch 3/10
 1/25 [>…] - ETA: 13:09 - loss: 0.6287 - accuracy:
0.7670
```

[ ]:
```
unet.save('seg_survellence')
model = tf.keras.models.load_model('seg_survellence')
```

[ ]:
```
acc = model_history.history['accuracy']
loss = model_history.history['loss']

epochs_range = range(100)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.legend(loc='lower right')
plt.title('Training Accuracy')
plt.show()

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, loss, label='Training Loss')
plt.legend(loc='lower right')
plt.title('Training loss')
plt.show()
```

[ ]:
```
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.preprocessing.image.array_to_img(display_list[i]))
```

```
        plt.axis('off')
    plt.show()
```

[ ]:
```
def create_mask(pred_mask):
    pred_mask = tf.argmax(pred_mask, axis=-1)
    pred_mask = pred_mask[..., tf.newaxis]
    return pred_mask[0]
```

[ ]:
```
def show_predictions(dataset=None, num=1):
    if dataset:
        for image, mask in dataset.take(num):
            pred_mask = unet.predict(image)
            display([image[0], mask[0], create_mask(pred_mask)])
```

[ ]:
```
show_predictions(train_dataset, 6)
```

[ ]:

[ ]: