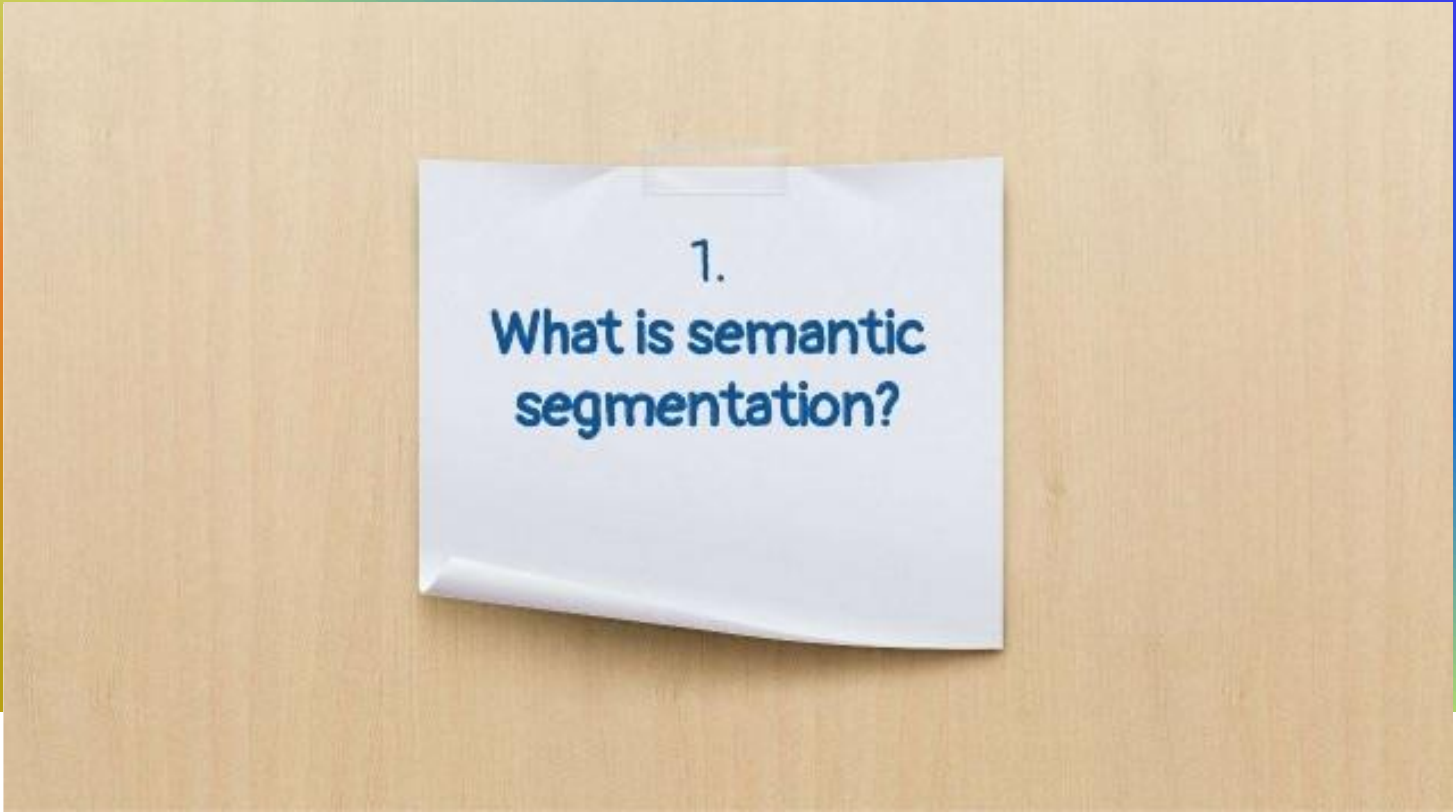
Deep Learning based Semantic Segmentation using Drone Image Dataset

Anam Nawaz Khan

MCL AD20206809

Table of Contents:

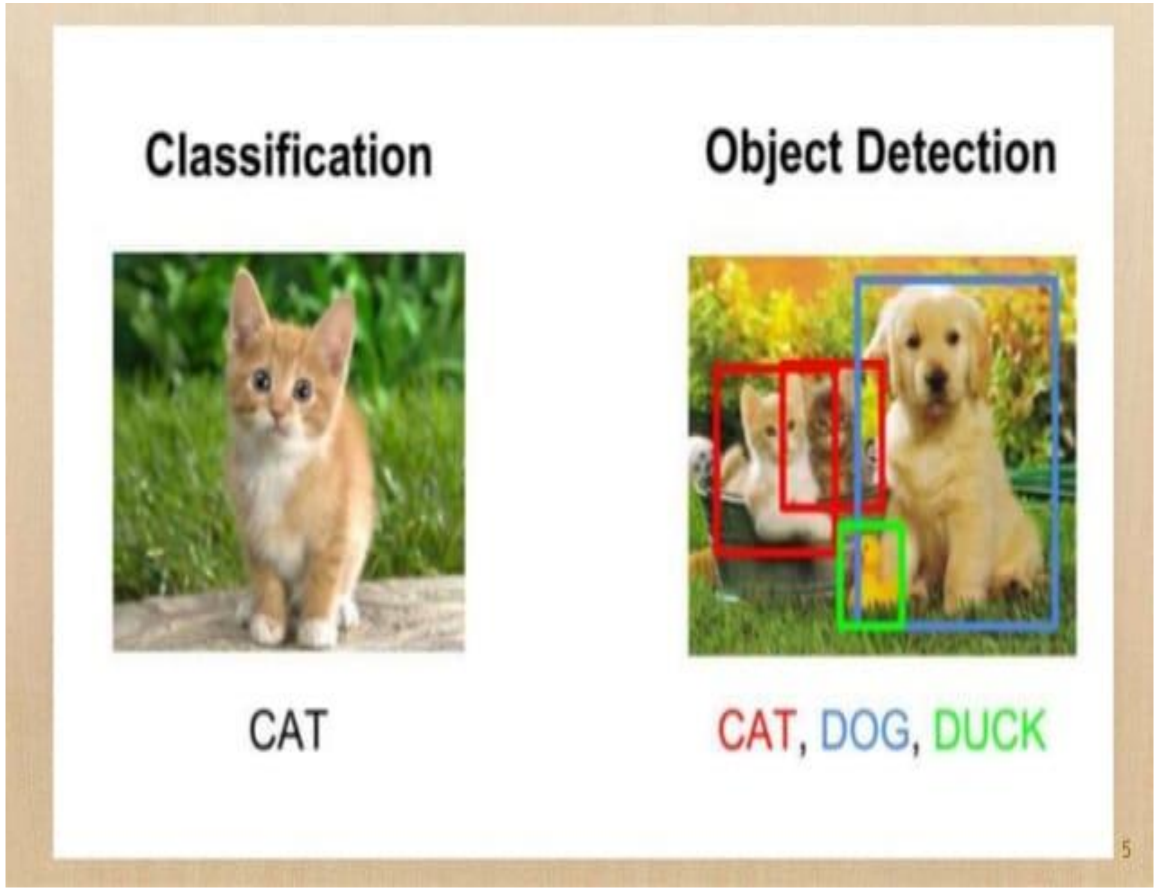


1.

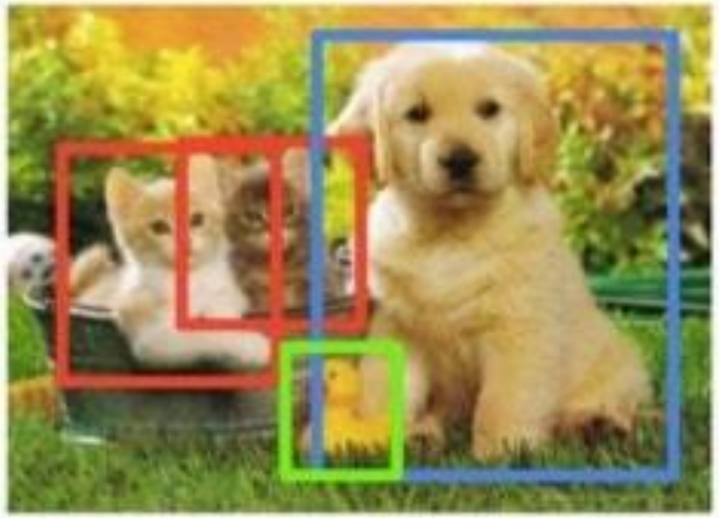
What is semantic segmentation?

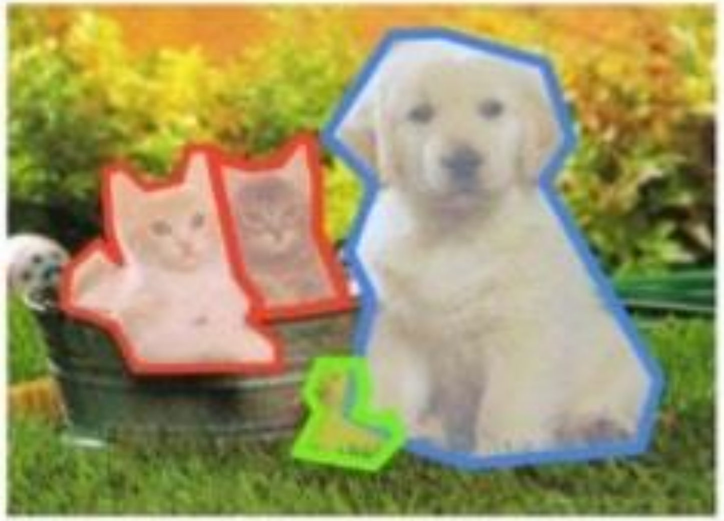
Limitations

Still a bit rough since we are only drawing bounding boxes and don't get an accurate idea of object shape



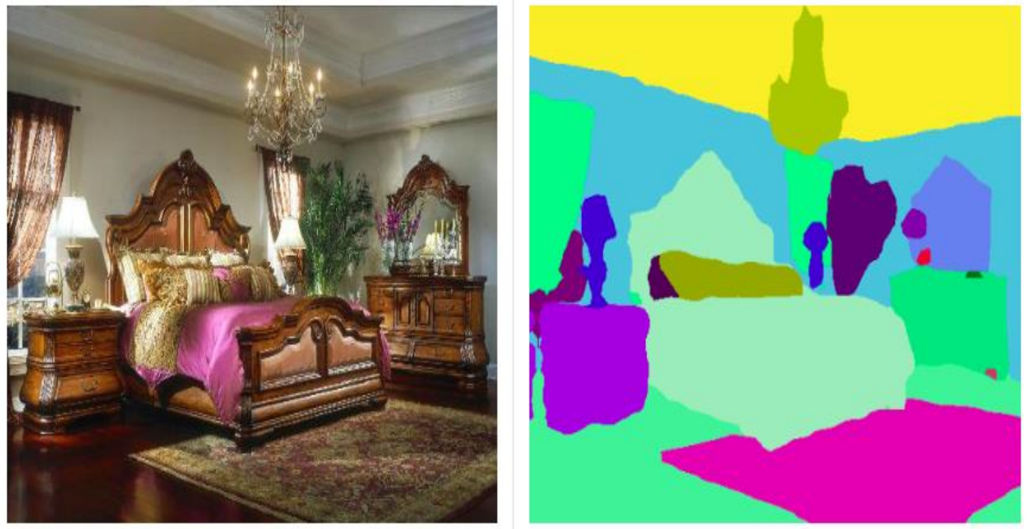
What if!?

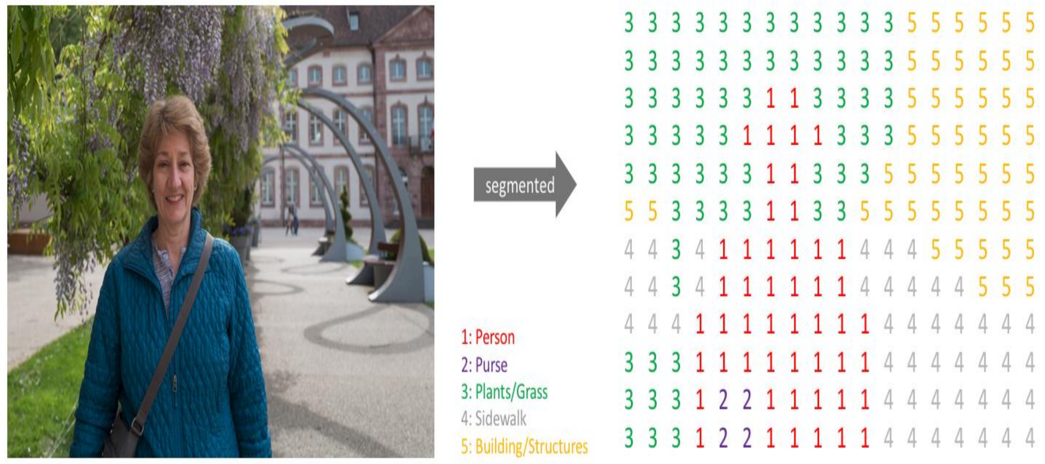




What is semantic segmentation

+ Semantic image segmentation is the task of classifying each pixel in an image from a predefined set of classes. In the following example, different entities are classified.
+ In the above example, the pixels belonging to the bed are classified in the class "bed", the pixels corresponding to the walls are labeled as "wall", etc.
+ In particular, our goal is to take an image of size W x H x 3 and generate a W x H matrix containing the predicted class ID's corresponding to all the pixels.
+ Therefore In order to perform semantic segmentation, a higher level understanding of the image is required.
+ The algorithm should figure out the objects present and also the pixels which correspond to the object.
+ Semantic segmentation is one of the essential tasks for complete scene understanding.





3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	1	1	1	1	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	5	5	5	5	5	5	5
5	5	3	3	3	3	1	1	3	3	5	5	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	4	4	5	5	5
4	4	4	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4

Input

Semantic Labels

2.
Applications of Semantic Segmentation

Semantic Segmentation Applications

There are several applications for which semantic segmentation is very useful.

Medical images

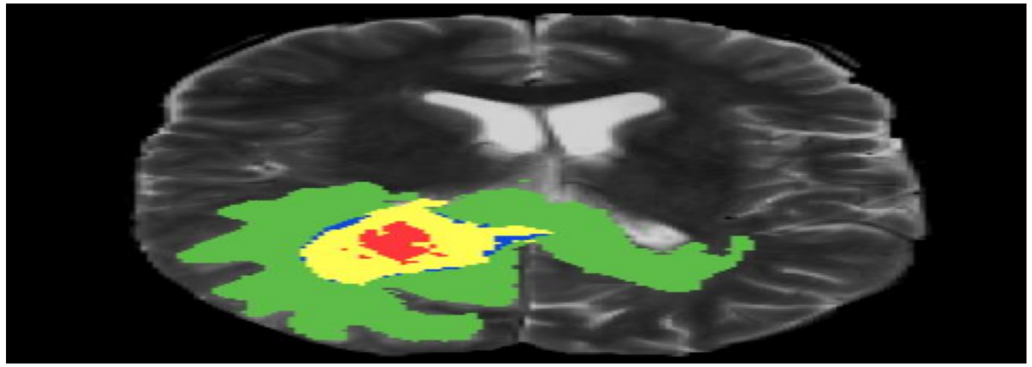
- Automated segmentation of body scans can help doctors to perform diagnostic tests. For example, models can be trained to segment tumor.

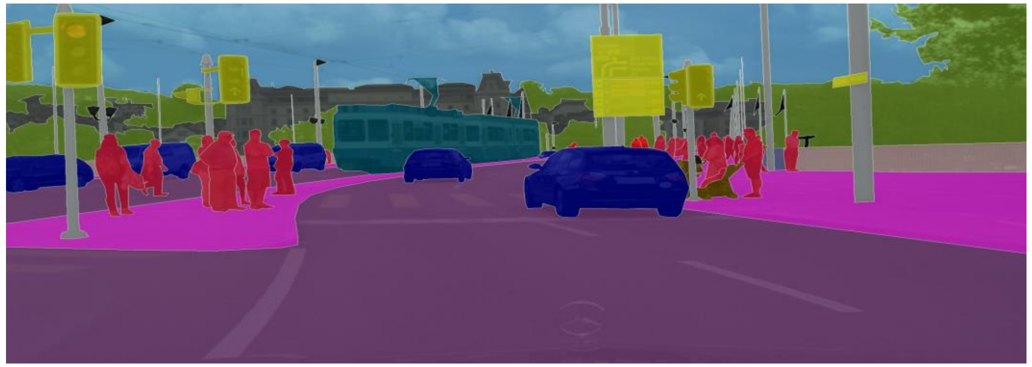
Autonomous vehicles

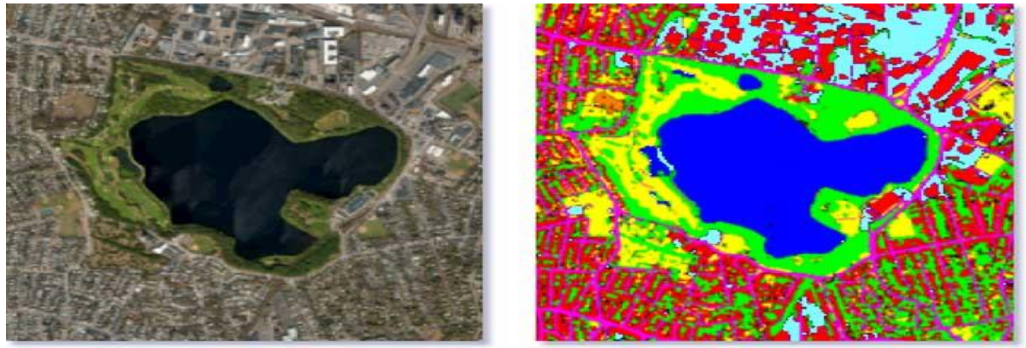
- Autonomous vehicles such as self-driving cars and drones can benefit from automated segmentation. For example, self-driving cars can detect drivable regions.

Satellite image analysis

- Aerial images can be used to segment different types of land. Automated land mapping can also be done.

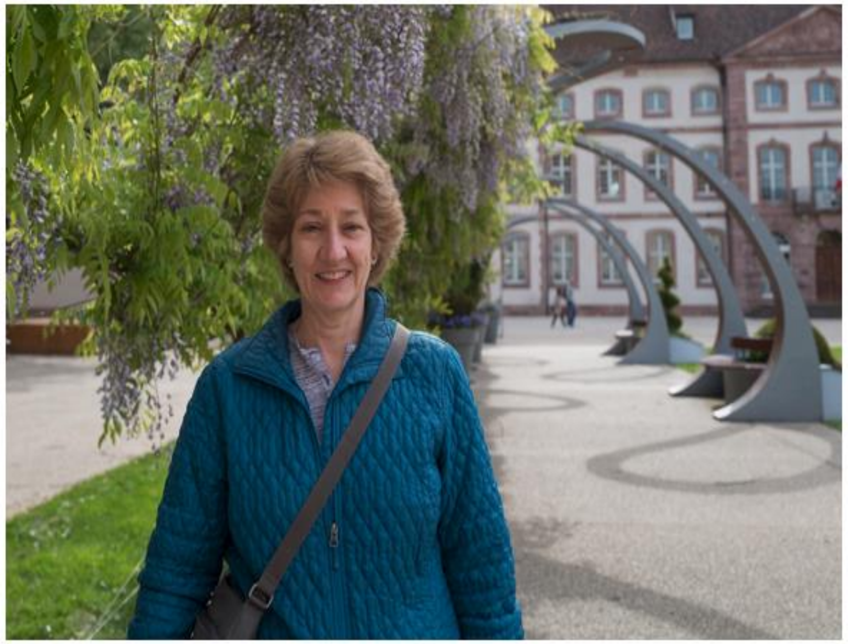

Tumor segmentation of brain MRI scan


Segmentation of a road scene


Segmentation of a satellite image

3. Representing the Task

Our goal is to take either a RGB color image or a gray-scale image and output a segmented map where each pixel contain a class label represented as integer



segmented

1: Person
2: Purse
3: Plants/Grass
4: Sidewalk
5: Building/Structures

3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	1	1	1	1	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	5	5	5	5	5	5	5
5	5	3	3	3	3	1	1	3	3	5	5	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	4	4	5	5	5
4	4	4	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4

Input

Semantic Labels

3. Constructing an Architecture

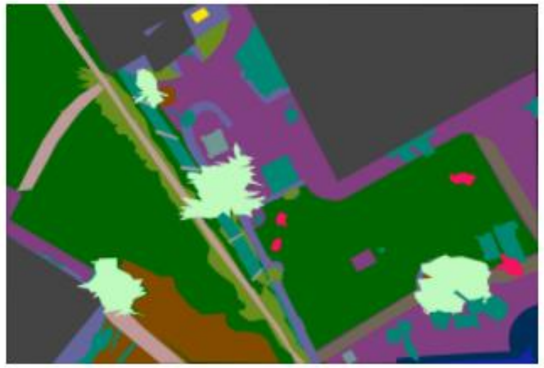
Aerial Semantic Segmentation Drone Dataset

Dataset Overview

+ The Semantic Drone Dataset focuses on semantic understanding of urban scenes for increasing the safety of autonomous drone flight and landing procedures.
+ The imagery depicts more than 20 houses from bird's eye view acquired at an altitude of 5 to 30 meters above ground.
+ A high resolution camera was used to acquire images at a size of 6000x4000px (24Mpx).
+ The training set contains 400 publicly available images and the test set is made up of 200 private images.
+ We prepared pixel-accurate annotation for the same training and test set. The complexity of the dataset is limited to 20 classes as listed in the following table.

Table 1: Semantic classes of the Drone Dataset

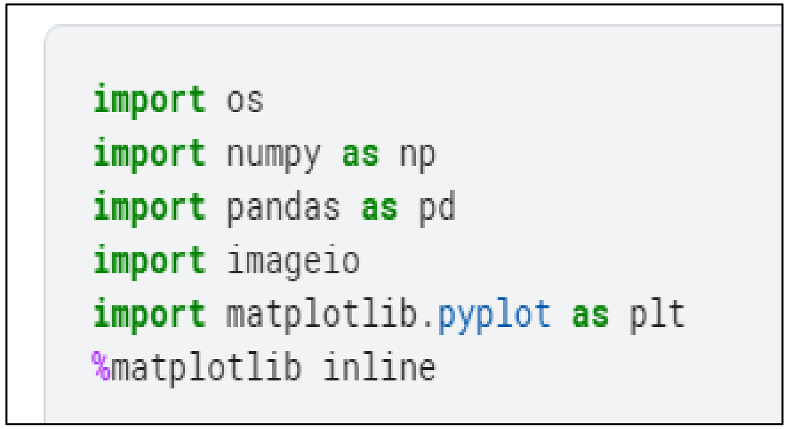
tree	rocks	dog	fence
gras	water	car	fence-pole
other vegetation	paved area	bicycle	window
dirt	pool	roof	door
gravel	person	wall	obstacle



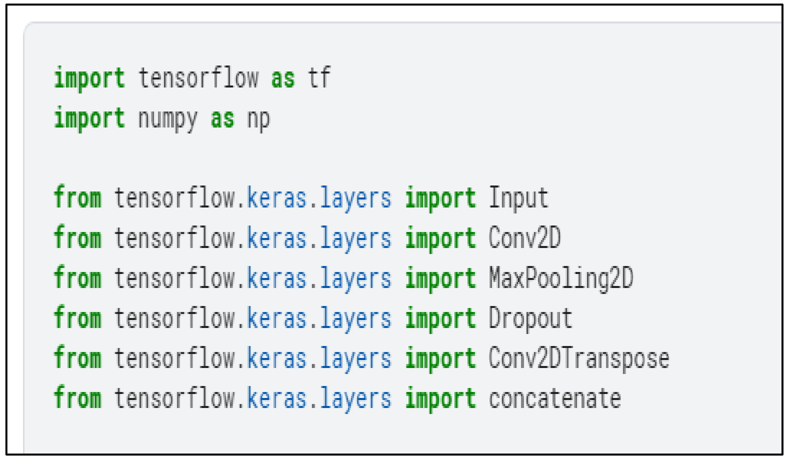
Libraries

Importing the Necessary Libraries

+ NumPy is a Python library used for working with arrays.
+ Pandas is a predominantly used python data analysis library.
+ Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy.
+ Imageio is a Python library that provides an easy interface to read and write a wide range of image data
+ Input() is used to instantiate a Keras tensor. A Keras tensor is a symbolic tensor-like object, which we augment with certain attributes that allow us to build a Keras model by knowing the inputs and outputs of the model.
+ Keras Conv2D is a 2D Convolution Layer, this layer creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs.
+ MaxPooling2D layer Down-samples the input along its spatial dimensions (height and width) by taking the maximum value over an input window (of size defined by pool_size) for each channel of the input.
+ Dropout: A Simple Way to Prevent Neural Networks from Overfitting
+ Conv2DTranspose Using Conv2DTranspose will up-sample its input but the key difference is the model should learn what is the best up-sampling for the job. conv2d_transpose() simply transposes the weights and flips them by 180 degrees. Then it applies the standard conv2d(). "Transposes" practically means that it changes the order of the "columns" in the weights tensor.
+ Concatenate is a Layer that concatenates a list of inputs. It takes as input a list of tensors, all of the same shape except for the concatenation axis, and returns a single tensor that is the concatenation of all inputs.



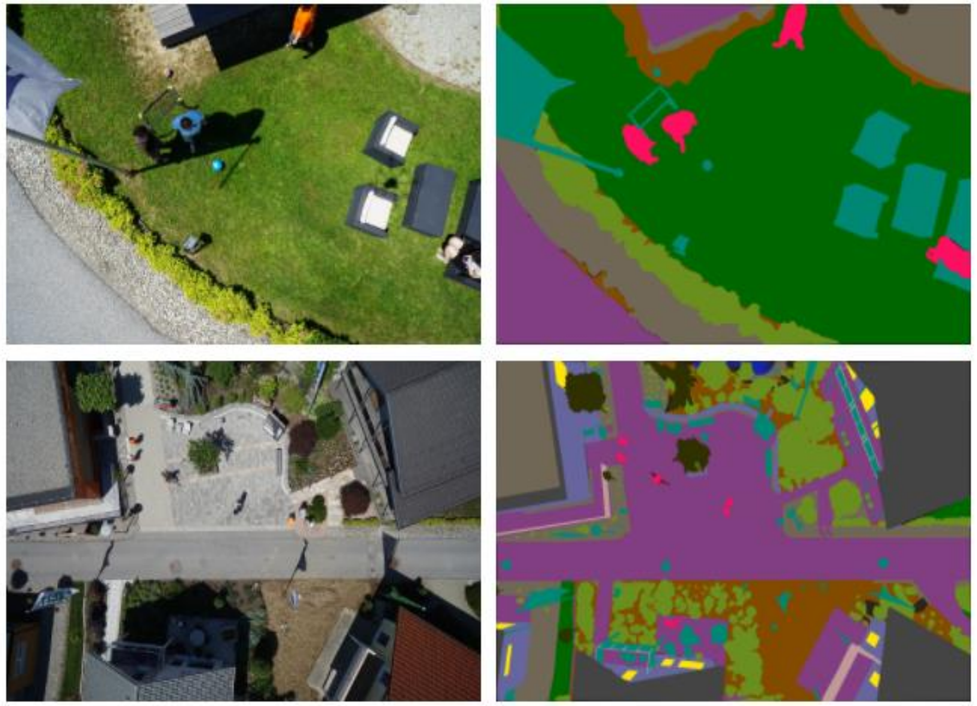
```
import os
import numpy as np
import pandas as pd
import imageio
import matplotlib.pyplot as plt
%matplotlib inline
```

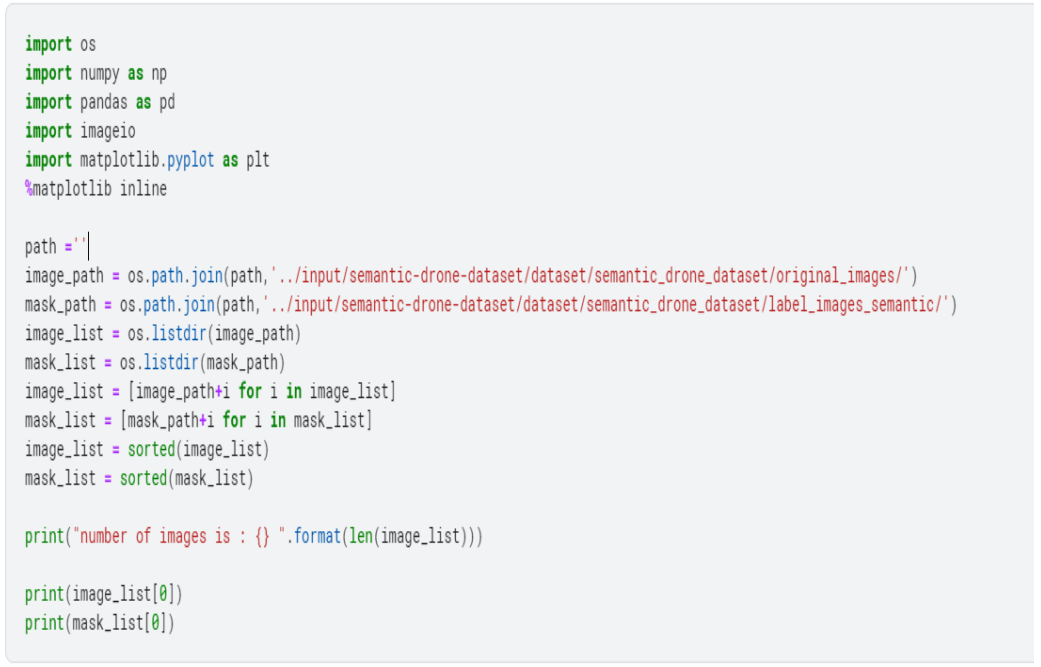


```
import tensorflow as tf
import numpy as np

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
```

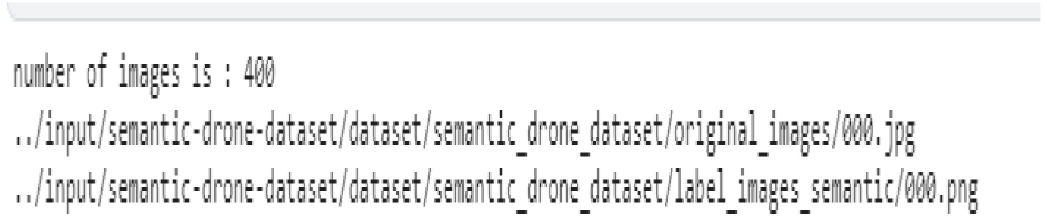
Load and Split The Data

- The training set contains 400 publicly available images and the test set is made up of 200 private images.





```python
import os
import numpy as np
import pandas as pd
import imageio
import matplotlib.pyplot as plt
%matplotlib inline

path =''
image_path = os.path.join(path,'../input/semantic-drone-dataset/dataset/semantic_drone_dataset/original_images/')
mask_path = os.path.join(path,'../input/semantic-drone-dataset/dataset/semantic_drone_dataset/label_images_semantic/')
image_list = os.listdir(image_path)
mask_list = os.listdir(mask_path)
image_list = [image_path+i for i in image_list]
mask_list = [mask_path+i for i in mask_list]
image_list = sorted(image_list)
mask_list = sorted(mask_list)

print("number of images is : {} ".format(len(image_list)))

print(image_list[0])
print(mask_list[0])
```

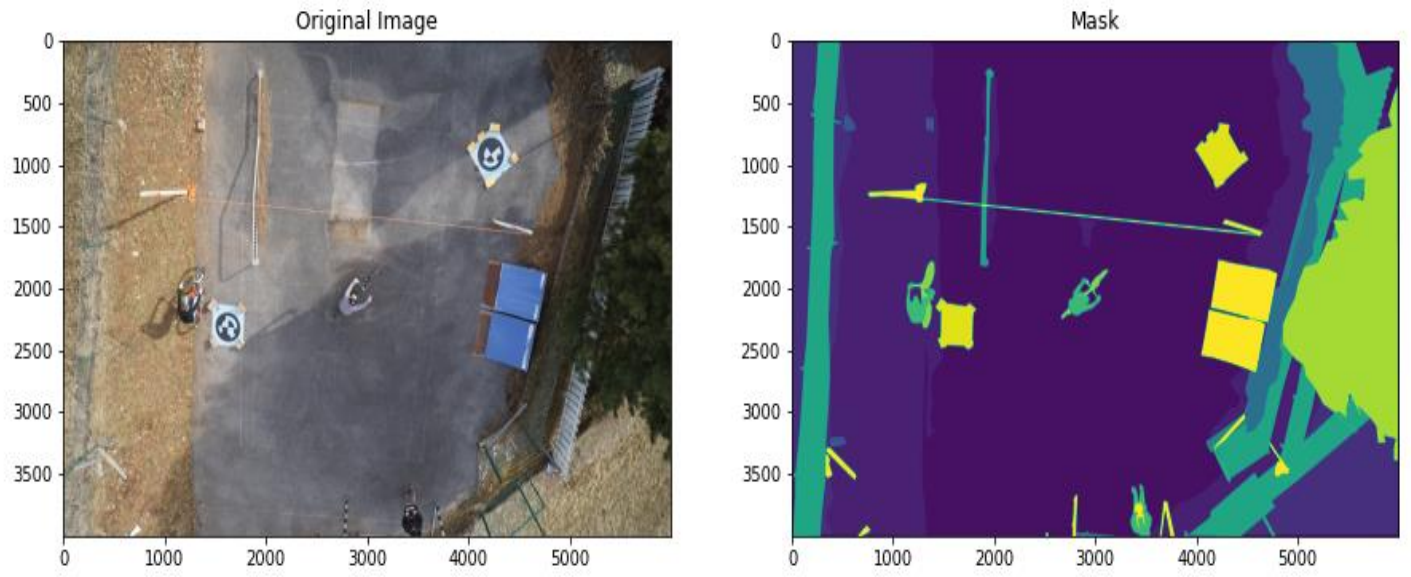


```
number of images is : 400
../input/semantic-drone-dataset/dataset/semantic_drone_dataset/original_images/000.jpg
../input/semantic-drone-dataset/dataset/semantic_drone_dataset/label_images_semantic/000.png
```

Exploration and overview of images

Drone's images semantic segmentation overview

```
n = 10 # you can chose any index
img  = imageio.imread(image_list[n])
print(img.shape)
mask = imageio.imread(mask_list[n])
print(mask.shape)

# now let's plot
fig ,arr  = plt.subplots(1,2,figsize=(15,10))
arr[0].imshow(img)
arr[0].set_title('Original Image')
arr[1].imshow(mask)
arr[1].set_title('Mask')
```



Preprocessing our data

- Tf.io.read_file

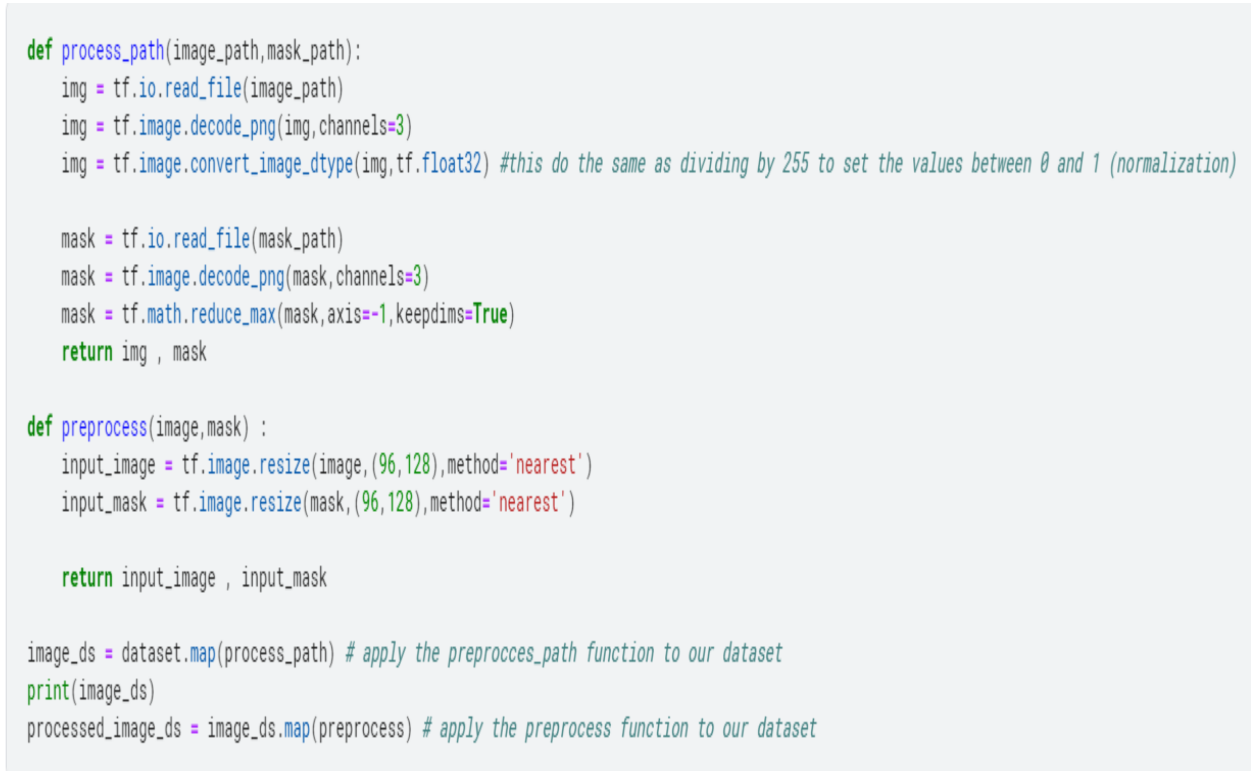
  Returns the contents of the file
- Tf.img.decode_png

  Decodes an encoded image to a unit8 or uint16 tensor
- Tf.image.convert_image_dtype

  Supported datatype of the image
- Tf.reduce_max

  Reduce the dimensions along the dimensions given in axis
- Tf.image.resize

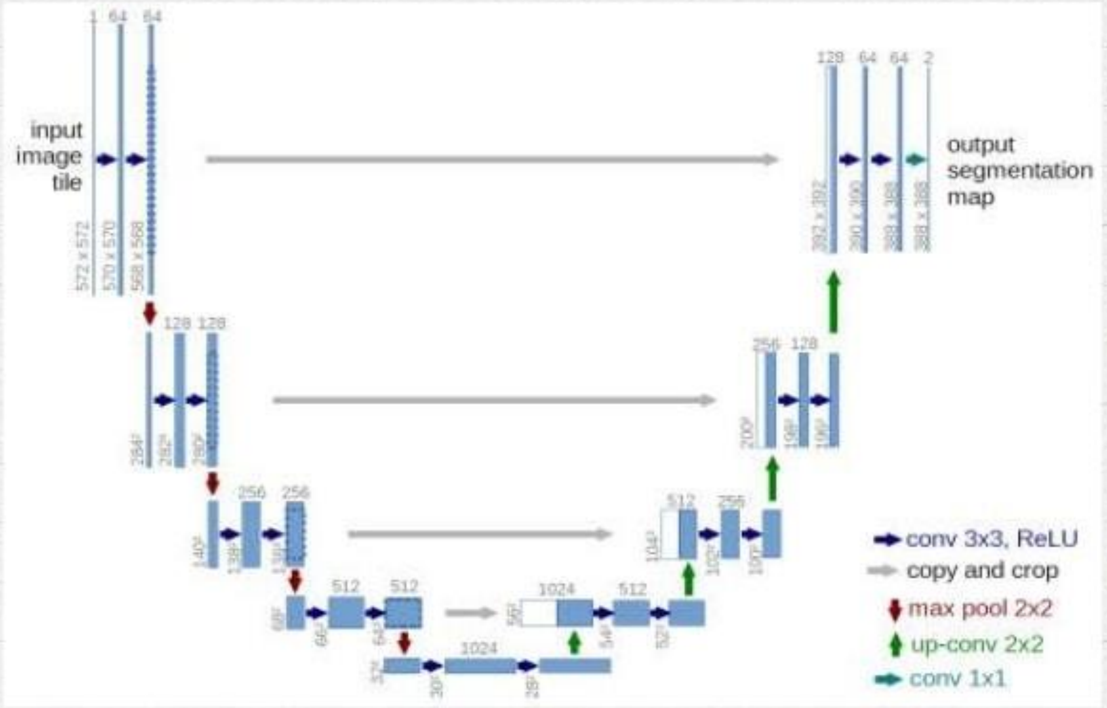
  Resize the images using a specified method



```python
def process_path(image_path,mask_path):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_png(img,channels=3)
    img = tf.image.convert_image_dtype(img,tf.float32) #this do the same as dividing by 255 to set the values between 0 and 1 (normalization)

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

def preprocess(image,mask) :
    input_image = tf.image.resize(image,(96,128),method='nearest')
    input_mask = tf.image.resize(mask,(96,128),method='nearest')

    return input_image , input_mask

image_ds = dataset.map(process_path) # apply the preprocces_path function to our dataset
print(image_ds)
processed_image_ds = image_ds.map(preprocess) # apply the preprocess function to our dataset
```

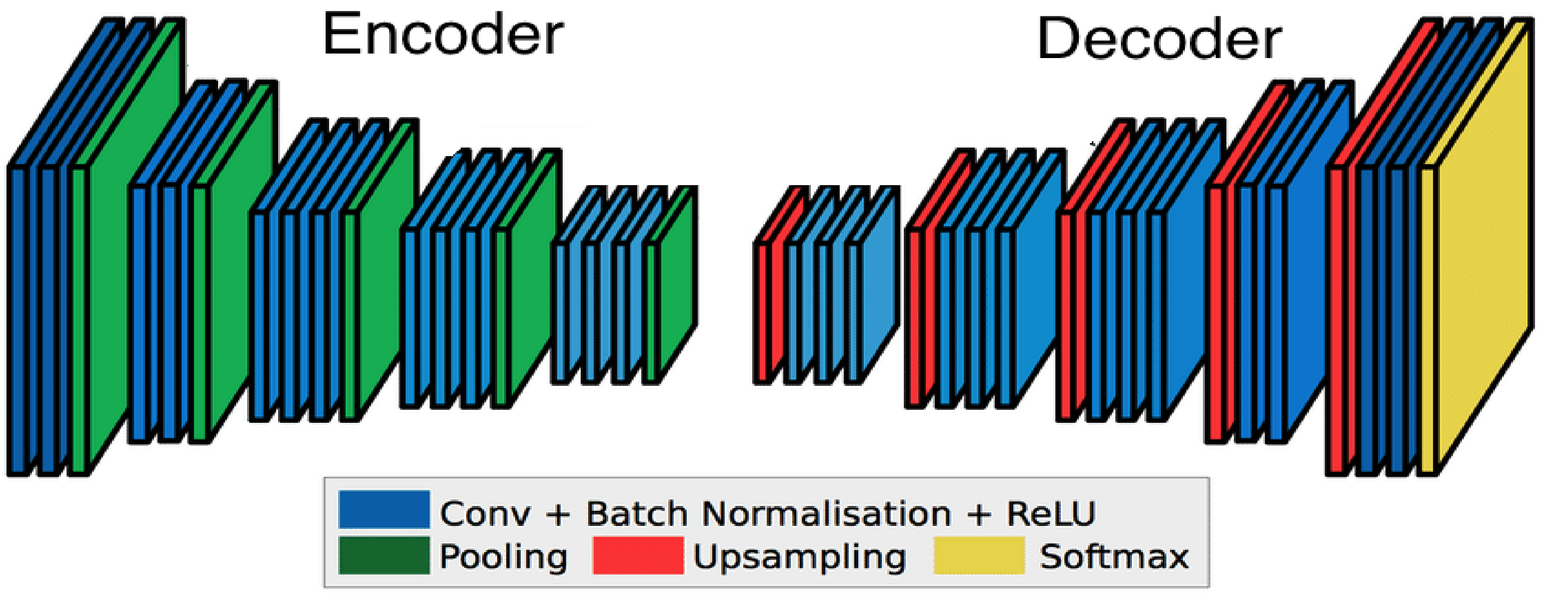
U-Net Architecture..

Consists of a contracting path to capture context and a symmetric expa nding path that enables precise localization.



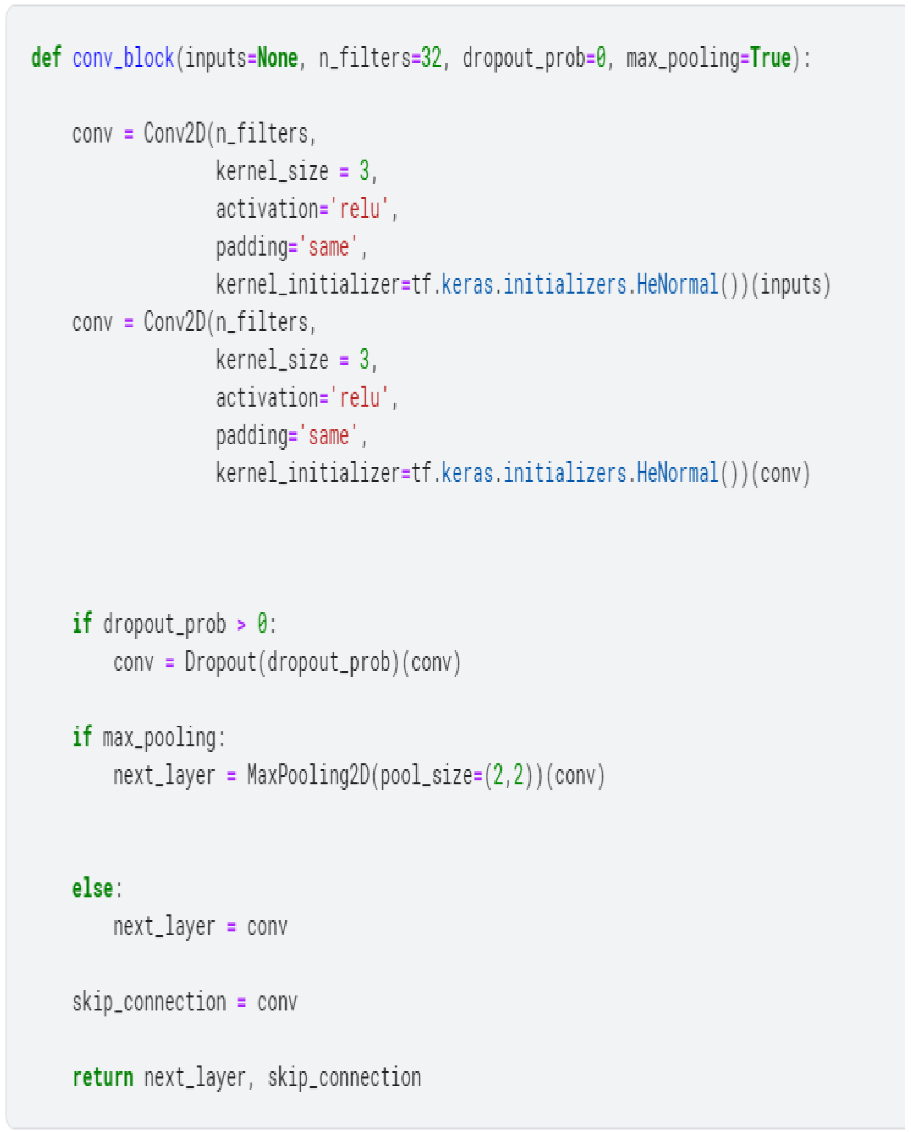
Encoder-decoder Structure

- This is called an encoder-decoder structure. Where the layers which down-sample the input are the part of the encoder and the layers which up-sample are part of the decoder
- When the model is trained for the task of semantic segmentation, the encoder outputs a tensor containing information about the objects, and its shape and size. The decoder takes this information and produces the segmentation maps.



Define The Conv Block For The Contracting Path

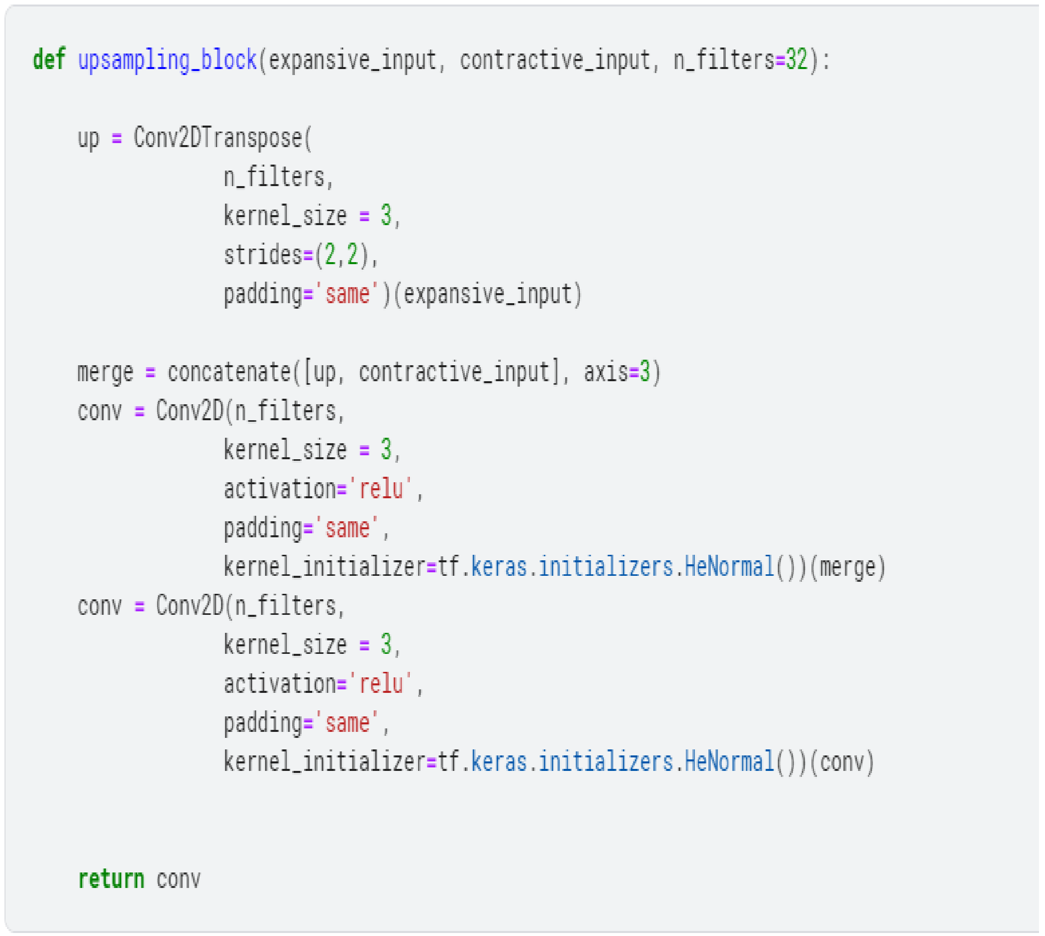
- The encoder network acts as the feature extractor and learns an abstract representation of the input image through a sequence of the encoder blocks. Here number of filters are 32 and two consecutive Convolution Layers are applied
- In the Encoder, the size of the image gradually reduces while the depth gradually increases.
- Each encoder block consists of two 3×3 convolutions, where each convolution is followed by a ReLU (Rectified Linear Unit) activation function.
- The ReLU activation function introduces non-linearity into the network, which helps in the better generalization of the training data.
- The block gives two result values as a return value. The first value is the activation value to be used in the next layer, and the second value is the value before pooling for the skip connection. The initial value declaration of the filter uses the He-Initialization method.
- The output of the ReLU acts as a skip connection for the corresponding decoder block. These skip connections provide additional information that helps the decoder to generate better semantic features.
- Next, follows a 2×2 max-pooling, where the spatial dimensions (height and width) of the feature maps are reduced by half.
- This reduces the computational cost by decreasing the number of trainable parameters.



```
def conv_block(inputs=None, n_filters=32, dropout_prob=0, max_pooling=True):

    conv = Conv2D(n_filters,
                  kernel_size = 3,
                  activation='relu',
                  padding='same',
                  kernel_initializer=tf.keras.initializers.HeNormal())(inputs)
    conv = Conv2D(n_filters,
                  kernel_size = 3,
                  activation='relu',
                  padding='same',
                  kernel_initializer=tf.keras.initializers.HeNormal())(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
```

Define the up-sampling block for the expanding path

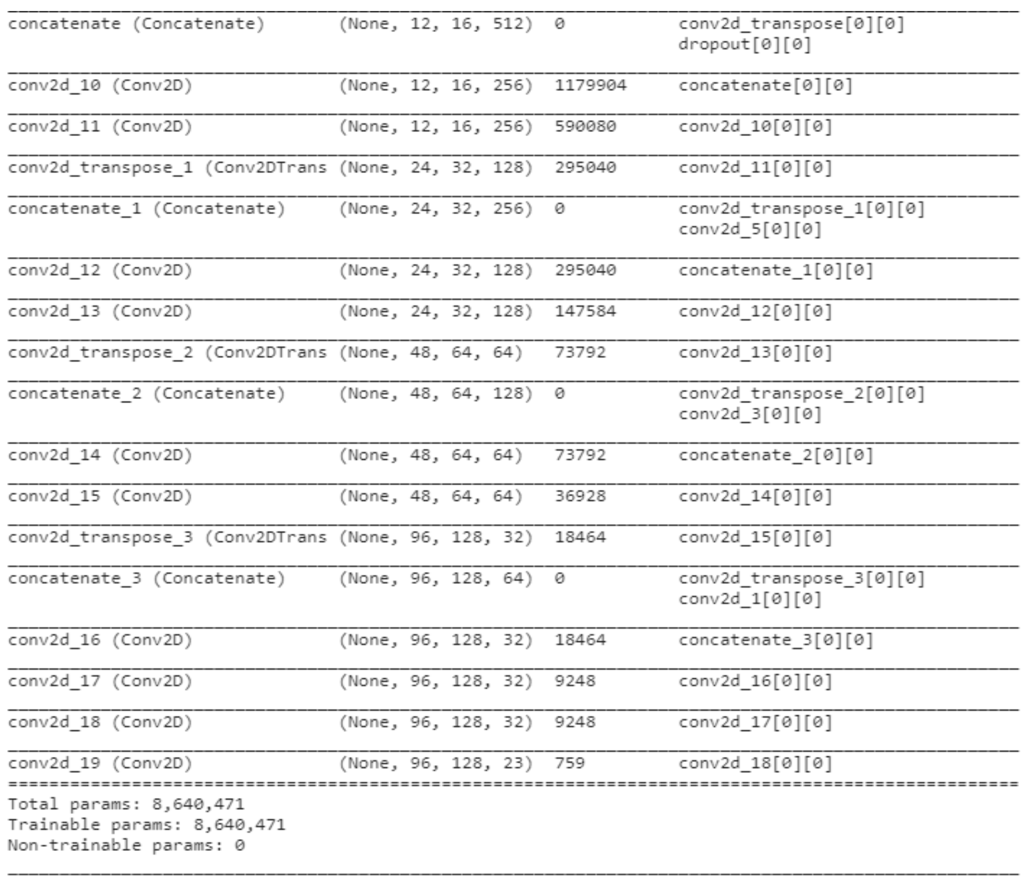
- The expansion path (Decoder) where we apply transposed convolutions along with regular convolutions
- In the decoder, the size of the image gradually increases and the depth gradually decreases.
- The decoder network is used to take the abstract representation and generate a semantic segmentation mask.
- Intuitively, the Decoder recovers the “WHERE” information (precise localization) by gradually applying up-sampling
- To get better precise locations, at every step of the decoder we use skip connections by concatenating the output of the transposed convolution layers with the feature maps from the Encoder at the same level:
- After every concatenation we again apply two consecutive regular convolutions so that the model can learn to assemble a more precise output
- This is what gives the architecture a symmetric U-shape, hence the name UNET
- On a high level, we have the following relationship: Input (128x128x1) (spatial dimx feature channels)=> Encoder =>(8x8x256) => Decoder =>Ouput (128x128x1)
- The U-Net combines the location information from the down sampling path to finally obtain a general information combining localization and context, which is necessary to predict a good segmentation map.



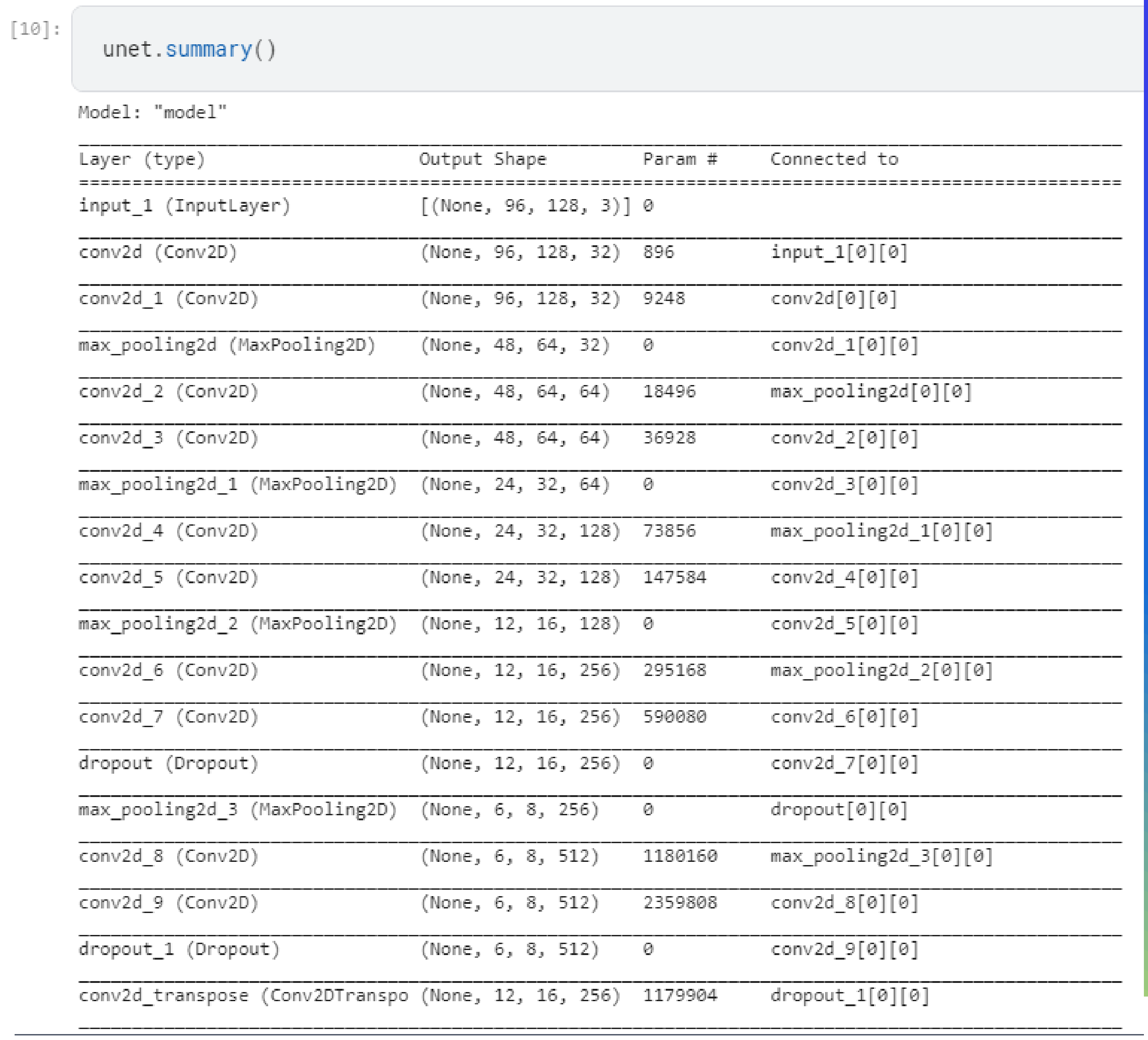
```
def upsampling_block(expansive_input, contractive_input, n_filters=32):

    up = Conv2DTranspose(
                 n_filters,
                 kernel_size = 3,
                 strides=(2,2),
                 padding='same')(expansive_input)

    merge = concatenate([up, contractive_input], axis=3)
    conv = Conv2D(n_filters,
                 kernel_size = 3,
                 activation='relu',
                 padding='same',
                 kernel_initializer=tf.keras.initializers.HeNormal())(merge)
    conv = Conv2D(n_filters,
                 kernel_size = 3,
                 activation='relu',
                 padding='same',
                 kernel_initializer=tf.keras.initializers.HeNormal())(conv)


    return conv
```

Define the u-net model

- we will Define the unet model, which composes of a set of conv blocks and upsampling blocks



```
_____________________________________________________________________________________________________
concatenate (Concatenate)       (None, 12, 16, 512)  0           conv2d_transpose[0][0]
                                                                 dropout[0][0]
_____________________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 12, 16, 256)  1179904     concatenate[0][0]
_____________________________________________________________________________________________________
conv2d_11 (Conv2D)              (None, 12, 16, 256)  590080      conv2d_10[0][0]
_____________________________________________________________________________________________________
conv2d_transpose_1 (Conv2DTrans (None, 24, 32, 128)  295040      conv2d_11[0][0]
_____________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 24, 32, 256)  0           conv2d_transpose_1[0][0]
                                                                 conv2d_5[0][0]
_____________________________________________________________________________________________________
conv2d_12 (Conv2D)              (None, 24, 32, 128)  295040      concatenate_1[0][0]
_____________________________________________________________________________________________________
conv2d_13 (Conv2D)              (None, 24, 32, 128)  147584      conv2d_12[0][0]
_____________________________________________________________________________________________________
conv2d_transpose_2 (Conv2DTrans (None, 48, 64, 64)   73792       conv2d_13[0][0]
_____________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 48, 64, 128)  0           conv2d_transpose_2[0][0]
                                                                 conv2d_3[0][0]
_____________________________________________________________________________________________________
conv2d_14 (Conv2D)              (None, 48, 64, 64)   73792       concatenate_2[0][0]
_____________________________________________________________________________________________________
conv2d_15 (Conv2D)              (None, 48, 64, 64)   36928       conv2d_14[0][0]
_____________________________________________________________________________________________________
conv2d_transpose_3 (Conv2DTrans (None, 96, 128, 32)  18464       conv2d_15[0][0]
_____________________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 96, 128, 64)  0           conv2d_transpose_3[0][0]
                                                                 conv2d_1[0][0]
_____________________________________________________________________________________________________
conv2d_16 (Conv2D)              (None, 96, 128, 32)  18464       concatenate_3[0][0]
_____________________________________________________________________________________________________
conv2d_17 (Conv2D)              (None, 96, 128, 32)  9248        conv2d_16[0][0]
_____________________________________________________________________________________________________
conv2d_18 (Conv2D)              (None, 96, 128, 32)  9248        conv2d_17[0][0]
_____________________________________________________________________________________________________
conv2d_19 (Conv2D)              (None, 96, 128, 23)  759         conv2d_18[0][0]
=====================================================================================================
Total params: 8,640,471
Trainable params: 8,640,471
Non-trainable params: 0
_____________________________________________________________________________________________________
```

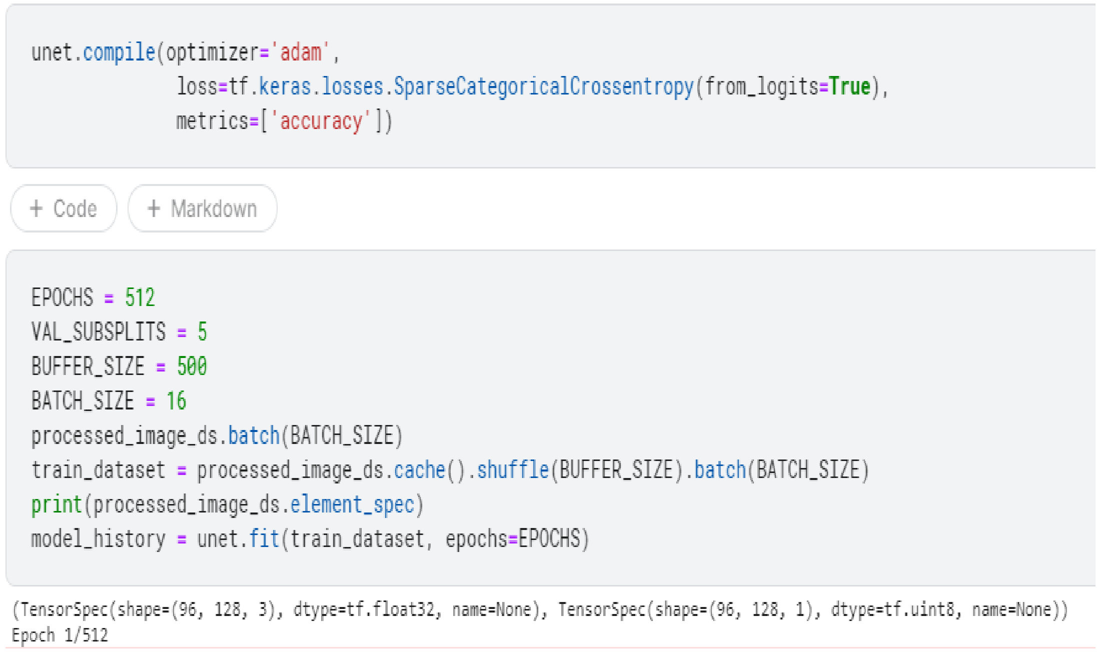


```
[10]:  unet.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 96, 128, 3)] 0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 96, 128, 32)  896         input_1[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 96, 128, 32)  9248        conv2d[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 48, 64, 32)   0           conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 48, 64, 64)   18496       max_pooling2d[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 48, 64, 64)   36928       conv2d_2[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 24, 32, 64)   0           conv2d_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 24, 32, 128)  73856       max_pooling2d_1[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 24, 32, 128)  147584      conv2d_4[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 12, 16, 128)  0           conv2d_5[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 12, 16, 256)  295168      max_pooling2d_2[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 12, 16, 256)  590080      conv2d_6[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 12, 16, 256)  0           conv2d_7[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 6, 8, 256)    0           dropout[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, 6, 8, 512)    1180160     max_pooling2d_3[0][0]
__________________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, 6, 8, 512)    2359808     conv2d_8[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 6, 8, 512)    0           conv2d_9[0][0]
__________________________________________________________________________________________________
conv2d_transpose (Conv2DTranspo (None, 12, 16, 256)  1179904     dropout_1[0][0]
__________________________________________________________________________________________________
```

Model Compilation

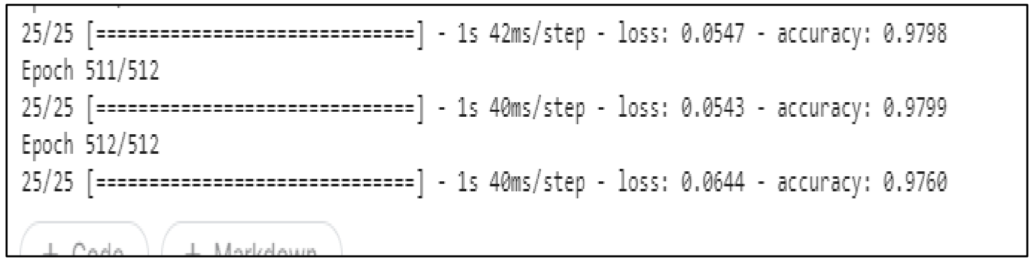
- After we finished building the UNet model, we compile this model.
- For this experiment, we uses Adam optimizer, crossentropy as loss function and accuracy as the metric to measure the performance.



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

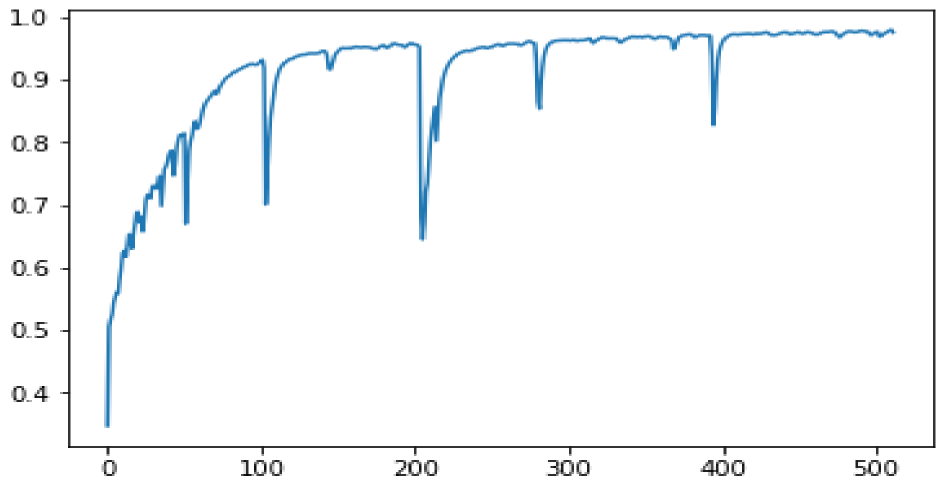
```
EPOCHS = 512
VAL_SUBSPLITS = 5
BUFFER_SIZE = 500
BATCH_SIZE = 16
processed_image_ds.batch(BATCH_SIZE)
train_dataset = processed_image_ds.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
print(processed_image_ds.element_spec)
model_history = unet.fit(train_dataset, epochs=EPOCHS)
```

```
(TensorSpec(shape=(96, 128, 3), dtype=tf.float32, name=None), TensorSpec(shape=(96, 128, 1), dtype=tf.uint8, name=None))
Epoch 1/512
```

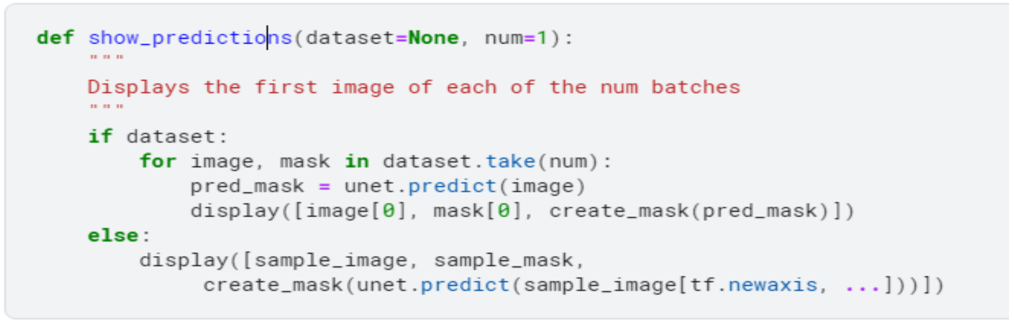


```
25/25 [==============================] - 1s 42ms/step - loss: 0.0547 - accuracy: 0.9798
Epoch 511/512
25/25 [==============================] - 1s 40ms/step - loss: 0.0543 - accuracy: 0.9799
Epoch 512/512
25/25 [==============================] - 1s 40ms/step - loss: 0.0644 - accuracy: 0.9760
```

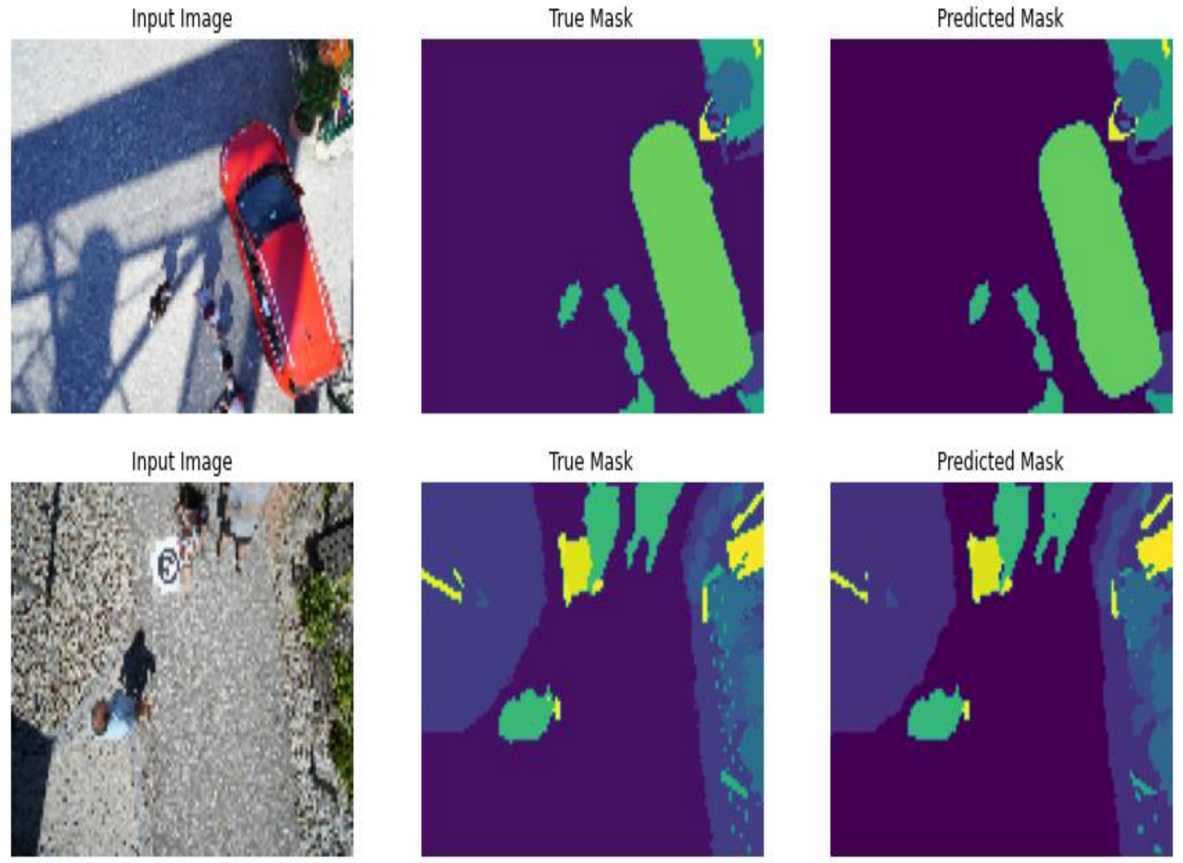
Model Accuracy

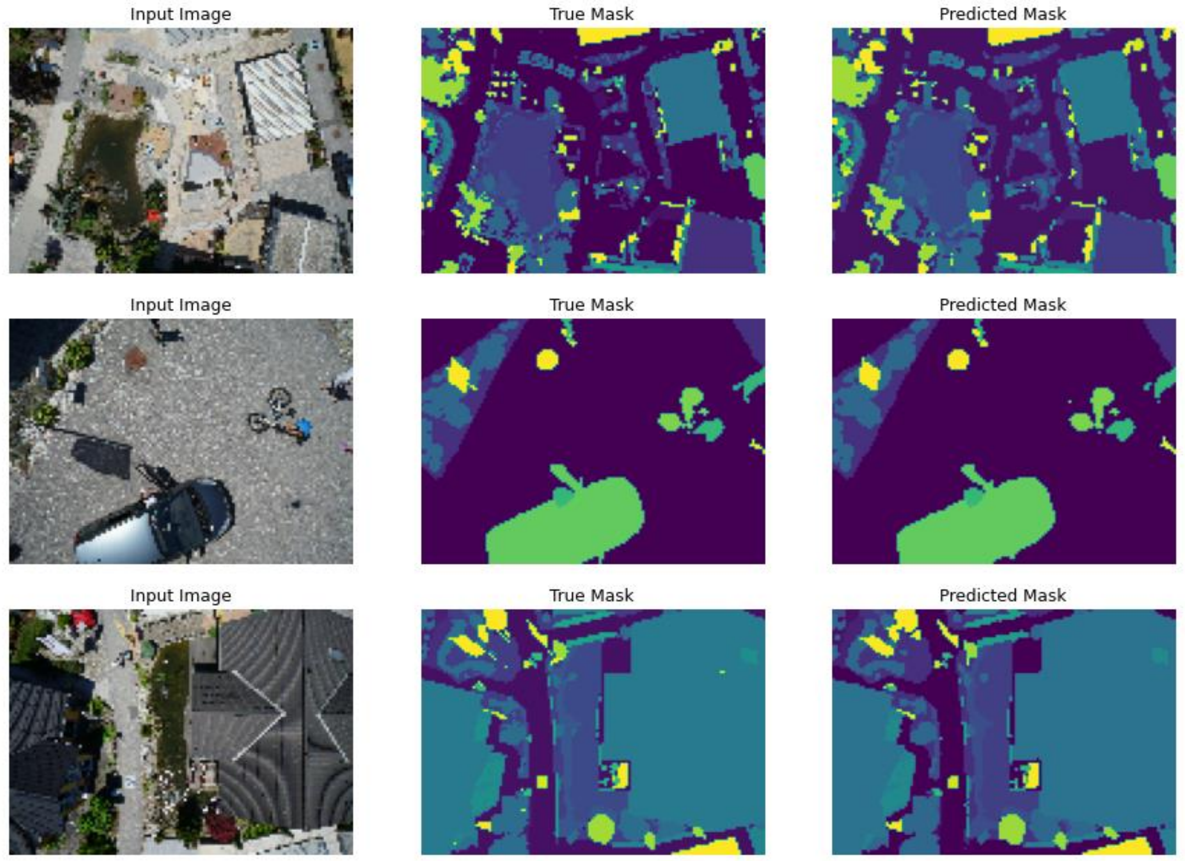


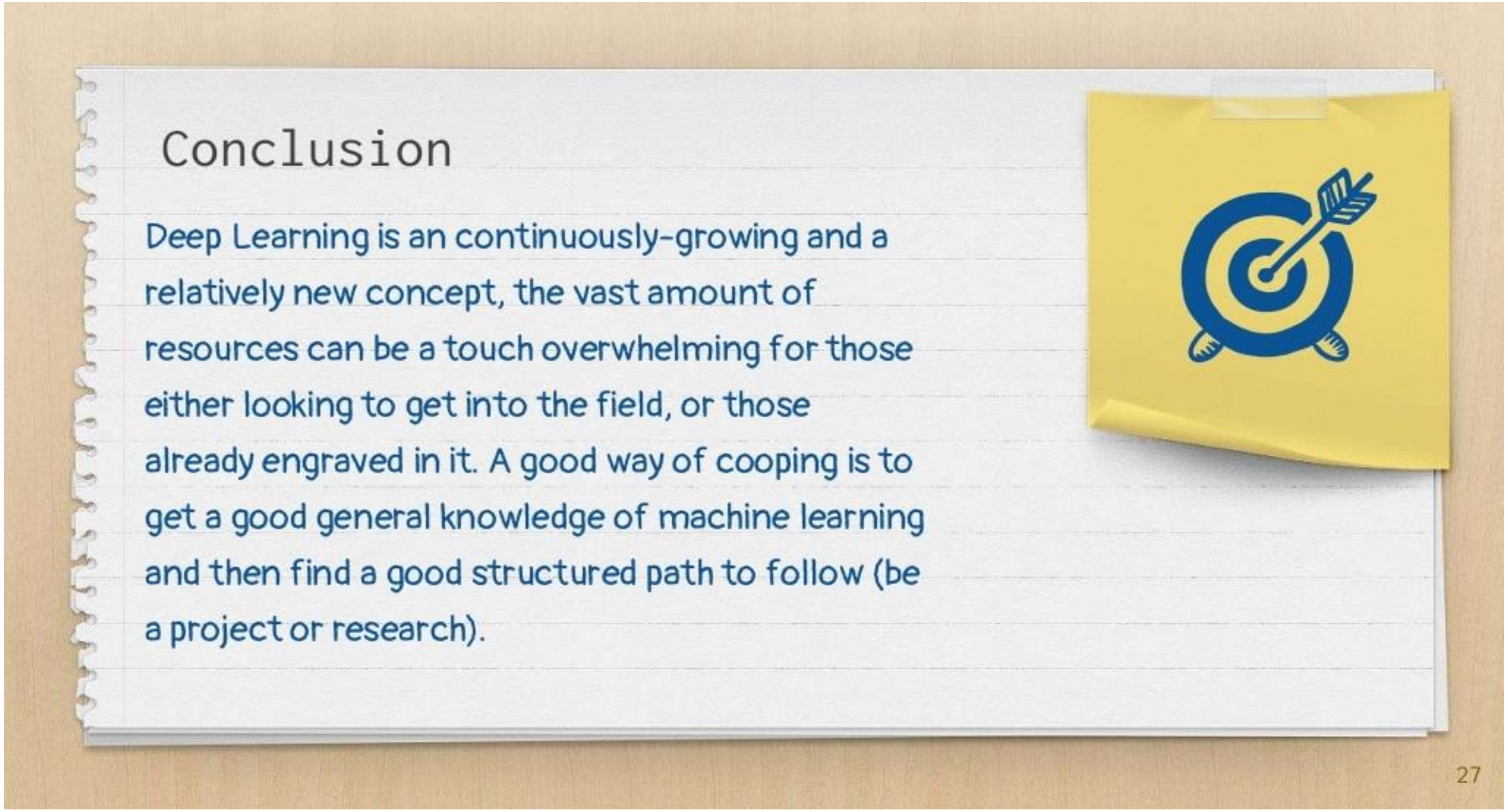
Results



```
def show_predictions(dataset=None, num=1):
    """
    Displays the first image of each of the num batches
    """
    if dataset:
        for image, mask in dataset.take(num):
            pred_mask = unet.predict(image)
            display([image[0], mask[0], create_mask(pred_mask)])
    else:
        display([sample_image, sample_mask,
             create_mask(unet.predict(sample_image[tf.newaxis, ...]))])
```







Conclusion

Deep Learning is an continuously-growing and a relatively new concept, the vast amount of resources can be a touch overwhelming for those either looking to get into the field, or those already engraved in it. A good way of cooping is to get a good general knowledge of machine learning and then find a good structured path to follow (be a project or research).

Thanks!

Any questions?

You can find me at:

anamnawaz@jejunu.ac.kr