Image Segmentation with U-Net

Using U-Net to predict a label for every single pixel in an image (an image from a self-driving car dataset).

- Building U-Net
- Implementing semantic image segmentation on the CARLA self-driving car dataset
- Apply sparse categorical crossentropy for pixelwise prediction

steps followed by code:

- 1- Load and Split the Data:
  - Split the dataset into Unmasked and Masked Images
  - Preprocess the Data
- 2- U-Net:
  - Encoder (Downsampling Block) --> conv_block
  - Decoder (Upsampling Block) --> upsampling_block
  - Build the Model --> unet_model
  - Set Model Dimensions
  - Loss Function
  - Dataset Handling
- 3- Train the Model:
  - Create Predicted Masks
  - Plot Model Accuracy
  - Show Predictions

In [2]:

```python
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

from test_utils_UNet import summary, comparator
```

1 - Load and Split the Data:

In [4]:

```python
import os
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import imageio

import matplotlib.pyplot as plt
%matplotlib inline

path = 'E:/new CV/DL/DL_projects/image_segmentation_UNet'
image_path = os.path.join(path, './data/CameraRGB/')
s.path.join(path, './data/CameraMask/')
```

```
image_list_orig = os.listdir(image_path)
image_list = [image_path+i for i in image_list_orig]
mask_list = [mask_path+i for i in image_list_orig]
```

Split the dataset into Unmasked and Masked Images:

In [5]:
```
image_list_ds = tf.data.Dataset.list_files(image_list, shuffle=False)
mask_list_ds = tf.data.Dataset.list_files(mask_list, shuffle=False)

for path in zip(image_list_ds.take(3), mask_list_ds.take(3)):
    print(path)
```

```
(<tf.Tensor: shape=(), dtype=string, numpy=b'E:\\new CV\\DL\\DL_projects\\image_segmenta
tion_UNet\\.\\data\\CameraRGB\\000026.png'>, <tf.Tensor: shape=(), dtype=string, numpy=
b'E:\\new CV\\DL\\DL_projects\\image_segmentation_UNet\\.\\data\\CameraMask\\000026.pn
g'>)
(<tf.Tensor: shape=(), dtype=string, numpy=b'E:\\new CV\\DL\\DL_projects\\image_segmenta
tion_UNet\\.\\data\\CameraRGB\\000027.png'>, <tf.Tensor: shape=(), dtype=string, numpy=
b'E:\\new CV\\DL\\DL_projects\\image_segmentation_UNet\\.\\data\\CameraMask\\000027.pn
g'>)
(<tf.Tensor: shape=(), dtype=string, numpy=b'E:\\new CV\\DL\\DL_projects\\image_segmenta
tion_UNet\\.\\data\\CameraRGB\\000028.png'>, <tf.Tensor: shape=(), dtype=string, numpy=
b'E:\\new CV\\DL\\DL_projects\\image_segmentation_UNet\\.\\data\\CameraMask\\000028.pn
g'>)
```

In [6]:
```
image_filenames = tf.constant(image_list)
masks_filenames = tf.constant(mask_list)

dataset = tf.data.Dataset.from_tensor_slices((image_filenames, masks_filenames))

for image, mask in dataset.take(1):
    print(image)
    print(mask)
```

```
tf.Tensor(b'E:/new CV/DL/DL_projects/image_segmentation_UNet\\./data/CameraRGB/000026.pn
g', shape=(), dtype=string)
tf.Tensor(b'E:/new CV/DL/DL_projects/image_segmentation_UNet\\./data/CameraMask/000026.p
ng', shape=(), dtype=string)
```

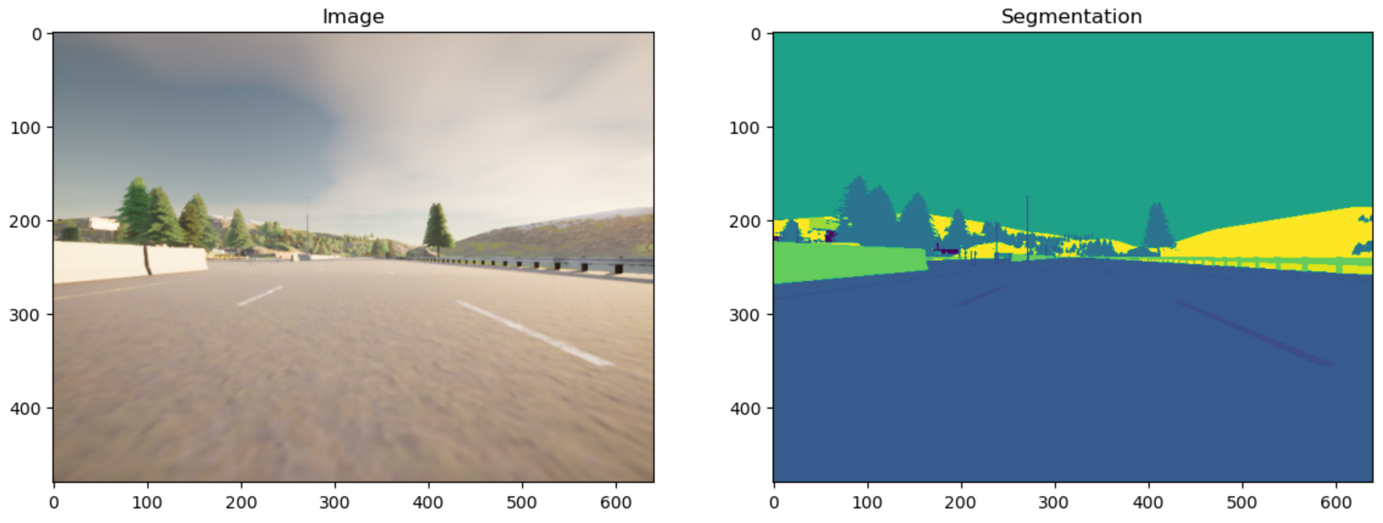
An example of the unmasked and the masked image from the dataset:

In [8]:
```
N = 2
img = imageio.imread(image_list[N])
mask = imageio.imread(mask_list[N])
#mask = np.array([max(mask[i, j]) for i in range(mask.shape[0]) for j in range(mask.shap

fig, arr = plt.subplots(1, 2, figsize=(14, 10))
arr[0].imshow(img)
arr[0].set_title('Image')
arr[1].imshow(mask[:, :, 0])
arr[1].set_title('Segmentation')
```

```
C:\Users\sanaz\AppData\Local\Temp\ipykernel_1712\2127741555.py:2: DeprecationWarning: St
arting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imrea
d. To keep the current behavior (and make this warning dissapear) use `import imageio.v2
as imageio` or call `imageio.v2.imread` directly.
  img = imageio.imread(image_list[N])
C:\Users\sanaz\AppData\Local\Temp\ipykernel_1712\2127741555.py:3: DeprecationWarning: St
arting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imrea
d. To keep the current behavior (and make this warning dissapear) use `import imageio.v2
as imageio` or call `imageio.v2.imread` directly.
  mask = imageio.imread(mask_list[N])
```

Out[8]:
```
Text(0.5, 1.0, 'Segmentation')
```

Loading [MathJax]/extensions/Safe.js



Preprocessing the data:

In [9]:
```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)

    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)
processed_image_ds = image_ds.map(preprocess)
```

2- U-Net:

- **Contracting path** (Encoder containing downsampling steps): It consists of the repeated application of two 3 x 3 same padding convolutions, each followed by a rectified linear unit (ReLU) and a 2 x 2 max pooling operation with stride 2 for downsampling. At each downsampling step, the number of feature channels is doubled.

- **Crop function**:This step crops the image from the contracting path and concatenates it to the current image on the expanding path to create a skip connection.

- **Expanding path** (Decoder containing upsampling steps): In detail, each step in the expanding path upsamples the feature map, followed by a 2 x 2 convolution (the transposed convolution). This transposed convolution halves the number of feature channels, while growing the height and width of the image. Next is a concatenation with the correspondingly cropped feature map from the contracting path, and two 3 x 3 convolutions, each followed by a ReLU.

- **Final Feature Mapping Block**: In the final layer, a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes.

Loading [MathJax]/extensions/Safe.js

Encoder (Downsampling Block) :

summary:

- Adding 2 Conv2D layers with n_filters with kernel_size set to 3, kernel_initializer set to he_normal, padding set to same and relu activation.
- if dropout_prob > 0, then adding a Dropout layer with parameter dropout_prob
- If max_pooling is set to True, then adding a MaxPooling2D layer with 2x2 pool size

In [10]:
```
def conv_block(inputs=None, n_filters=32, dropout_prob=0, max_pooling=True):
    """
    Convolutional downsampling block
    
    Arguments:
        inputs -- Input tensor
        n_filters -- Number of filters for the convolutional layers
        dropout_prob -- Dropout probability
        max_pooling -- Use MaxPooling2D to reduce the spatial dimensions of the output v
    Returns:
        next_layer, skip_connection --  Next layer and skip connection outputs
    """

    conv = Conv2D(n_filters, # Number of filters
                  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)
    
    # adding a dropout layer
    if dropout_prob > 0:
        conv = Dropout(dropout_prob)(conv)
         
         
    # adding a MaxPooling2D with 2x2 pool_size
    if max_pooling:
        next_layer = MaxPooling2D()(conv)
        
    else:
        next_layer = conv
        
    skip_connection = conv
    
    return next_layer, skip_connection
```

In [12]:
```
input_size=(96, 128, 3)
n_filters = 32
inputs = Input(input_size)
cblock1 = conv_block(inputs, n_filters * 1)
model1 = tf.keras.Model(inputs=inputs, outputs=cblock1)

output1 = [['InputLayer', [(None, 96, 128, 3)], 0],
            ['Conv2D', (None, 96, 128, 32), 896, 'same', 'relu', 'HeNormal'],
            ['Conv2D', (None, 96, 128, 32), 9248, 'same', 'relu', 'HeNormal'],
            ['MaxPooling2D', (None, 48, 64, 32), 0, (2, 2)]]

print('Block 1:')
```

Loading [MathJax]/extensions/Safe.js

```
for layer in summary(model1):
    print(layer)

comparator(summary(model1), output1)

inputs = Input(input_size)
cblock1 = conv_block(inputs, n_filters * 32, dropout_prob=0.1, max_pooling=True)
model2 = tf.keras.Model(inputs=inputs, outputs=cblock1)

output2 = [['InputLayer', [(None, 96, 128, 3)], 0],
            ['Conv2D', (None, 96, 128, 1024), 28672, 'same', 'relu', 'HeNormal'],
            ['Conv2D', (None, 96, 128, 1024), 9438208, 'same', 'relu', 'HeNormal'],
            ['Dropout', (None, 96, 128, 1024), 0, 0.1],
            ['MaxPooling2D', (None, 48, 64, 1024), 0, (2, 2)]]

print('\nBlock 2:')
for layer in summary(model2):
    print(layer)

comparator(summary(model2), output2)
```

```
Block 1:
['InputLayer', [(None, 96, 128, 3)], 0]
['Conv2D', (None, 96, 128, 32), 896, 'same', 'relu', 'HeNormal']
['Conv2D', (None, 96, 128, 32), 9248, 'same', 'relu', 'HeNormal']
['MaxPooling2D', (None, 48, 64, 32), 0, (2, 2)]
All tests passed!

Block 2:
['InputLayer', [(None, 96, 128, 3)], 0]
['Conv2D', (None, 96, 128, 1024), 28672, 'same', 'relu', 'HeNormal']
['Conv2D', (None, 96, 128, 1024), 9438208, 'same', 'relu', 'HeNormal']
['Dropout', (None, 96, 128, 1024), 0, 0.1]
['MaxPooling2D', (None, 48, 64, 1024), 0, (2, 2)]
All tests passed!
```

Decoder (Upsampling Block):

summary:

- Takes the arguments expansive_input (which is the input tensor from the previous layer) and contractive_input (the input tensor from the previous skip layer)
- The number of filters here is the same as in the downsampling block completed previously
- Your Conv2DTranspose layer will take n_filters with shape (3,3) and a stride of (2,2), with padding set to same. It's applied to expansive_input, or the input tensor from the previous layer.

In [13]:

```
def upsampling_block(expansive_input, contractive_input, n_filters=32):
    """
    Convolutional upsampling block

    Arguments:
        expansive_input -- Input tensor from previous layer
        contractive_input -- Input tensor from previous skip layer
        n_filters -- Number of filters for the convolutional layers
    Returns:
        conv -- Tensor output
    """

    up = Conv2DTranspose(
                 n_filters,    # number of filters
                 3,    # Kernel size
                 strides=(2,2),
```

Loading [MathJax]/extensions/Safe.js

```
                      padding='same')(expansive_input)

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

    return conv
```

In [14]:

```
input_size1=(12, 16, 256)
input_size2 = (24, 32, 128)
n_filters = 32
expansive_inputs = Input(input_size1)
contractive_inputs =  Input(input_size2)
cblock1 = upsampling_block(expansive_inputs, contractive_inputs, n_filters * 1)
model1 = tf.keras.Model(inputs=[expansive_inputs, contractive_inputs], outputs=cblock1)

output1 = [['InputLayer', [(None, 12, 16, 256)], 0],
            ['Conv2DTranspose', (None, 24, 32, 32), 73760],
            ['InputLayer', [(None, 24, 32, 128)], 0],
            ['Concatenate', (None, 24, 32, 160), 0],
            ['Conv2D', (None, 24, 32, 32), 46112, 'same', 'relu', 'HeNormal'],
            ['Conv2D', (None, 24, 32, 32), 9248, 'same', 'relu', 'HeNormal']]

print('Block 1:')
for layer in summary(model1):
    print(layer)

comparator(summary(model1), output1)
```

```
Block 1:
['InputLayer', [(None, 12, 16, 256)], 0]
['Conv2DTranspose', (None, 24, 32, 32), 73760]
['InputLayer', [(None, 24, 32, 128)], 0]
['Concatenate', (None, 24, 32, 160), 0]
['Conv2D', (None, 24, 32, 32), 46112, 'same', 'relu', 'HeNormal']
['Conv2D', (None, 24, 32, 32), 9248, 'same', 'relu', 'HeNormal']
All tests passed!
```

Build the model:

Putting all together, by chaining the encoder, bottleneck, and decoder. The number of output channels= 23 (That's because there are 23 possible labels for each pixel in this self-driving car dataset)

For the first half of the model:

- Begining with a conv block that takes the inputs of the model and the number of filters
- Then, chaining the first output element of each block to the input of the next convolutional block
- Next, doubling the number of filters at each step
- Beginning with conv_block4, add dropout_prob of 0.3
- For the final conv_block, setting dropout_prob to 0.3 again, and turning off max pooling

For the second half:

Loading [MathJax]/extensions/Safe.js

- Using cblock5 as expansive_input and cblock4 as contractive_input, with n_filters
- Chaining the output of the previous block as expansive_input and the corresponding contractive block output.
- At each step, usng half the number of filters of the previous block
- conv9 is a Conv2D layer with ReLU activation, He normal initializer, same padding
- Finally, conv10 is a Conv2D that takes the number of classes as the filter, a kernel size of 1, and same padding. The output of conv10 is the output of the model.

In [15]:
```
def unet_model(input_size=(96, 128, 3), n_filters=32, n_classes=23):
    """
    Unet model

    Arguments:
        input_size -- Input shape
        n_filters -- Number of filters for the convolutional layers
        n_classes -- Number of output classes
    Returns:
        model -- tf.keras.Model
    """
    inputs = Input(input_size)
    # Contracting Path (encoding)
    # Add a conv_block with the inputs of the unet_ model and n_filters
    cblock1 = conv_block(inputs, n_filters)
    # Chain the first element of the output of each block to be the input of the next co
    # Double the number of filters at each new step
    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) # Include a dropou
    # Include a dropout_prob of 0.3 for this layer, and avoid the max_pooling layer
    cblock5 = conv_block(cblock4[0], n_filters * 16, dropout_prob=0.3, max_pooling=False


    # Expanding Path (decoding)
    # Add the first upsampling_block.
    # Use the cblock5[0] as expansive_input and cblock4[1] as contractive_input and n_fi
    ublock6 = upsampling_block(cblock5[1], cblock4[1],  n_filters * 8)
    # Chain the output of the previous block as expansive_input and the corresponding co
    # At each step, using half the number of filters of the previous block
    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
```

In [17]:
```
import outputs
img_height = 96
img_width = 128
num_channels = 3

unet = unet_model((img_height, img_width, num_channels))
comparator(summary(unet), outputs.unet_model_output)
```

Loading [MathJax]/extensions/Safe.js

```
All tests passed!
```

Set Model Dimensions:

In [18]:
```
img_height = 96
img_width = 128
num_channels = 3

unet = unet_model((img_height, img_width, num_channels))
```

In [19]:
```
unet.summary()
```

```
Model: "model_6"
__________________________________________________________________________________________
__________
 Layer (type)                   Output Shape                  Param #     Connected to
==========================================================================================
==========
 input_8 (InputLayer)           [(None, 96, 128, 3)]          0           []

 conv2d_30 (Conv2D)             (None, 96, 128, 32)           896         ['input_8[0][0]']

 conv2d_31 (Conv2D)             (None, 96, 128, 32)           9248        ['conv2d_30[0][0]']

 max_pooling2d_8 (MaxPoolin     (None, 48, 64, 32)            0           ['conv2d_31[0][0]']
 g2D)

 conv2d_32 (Conv2D)             (None, 48, 64, 64)            18496       ['max_pooling2d_8[0]
[0]']

 conv2d_33 (Conv2D)             (None, 48, 64, 64)            36928       ['conv2d_32[0][0]']

 max_pooling2d_9 (MaxPoolin     (None, 24, 32, 64)            0           ['conv2d_33[0][0]']
 g2D)

 conv2d_34 (Conv2D)             (None, 24, 32, 128)           73856       ['max_pooling2d_9[0]
[0]']

 conv2d_35 (Conv2D)             (None, 24, 32, 128)           147584      ['conv2d_34[0][0]']

 max_pooling2d_10 (MaxPooli     (None, 12, 16, 128)           0           ['conv2d_35[0][0]']
 ng2D)

 conv2d_36 (Conv2D)             (None, 12, 16, 256)           295168      ['max_pooling2d_10
[0][0]']

 conv2d_37 (Conv2D)             (None, 12, 16, 256)           590080      ['conv2d_36[0][0]']

 dropout_4 (Dropout)            (None, 12, 16, 256)           0           ['conv2d_37[0][0]']

 max_pooling2d_11 (MaxPooli     (None, 6, 8, 256)             0           ['dropout_4[0][0]']
 ng2D)

 conv2d_38 (Conv2D)             (None, 6, 8, 512)             1180160     ['max_pooling2d_11
[0][0]']

 conv2d_39 (Conv2D)             (None, 6, 8, 512)             2359808     ['conv2d_38[0][0]']

 dropout_5 (Dropout)            (None, 6, 8, 512)             0           ['conv2d_39[0][0]']

 conv2d_transpose_5 (Conv2D     (None, 12, 16, 256)           1179904     ['dropout_5[0][0]']
 Transpose)

 concatenate_5 (Concatenate     (None, 12, 16, 512)           0           ['conv2d_transpose_5
[0][0]',
 )                                                                         'dropout_4[0][0]']

 conv2d_40 (Conv2D)             (None, 12, 16, 256)           1179904     ['concatenate_5[0]
[0]']

 conv2d_41 (Conv2D)             (None, 12, 16, 256)           590080      ['conv2d_40[0][0]']

 conv2d_transpose_6 (Conv2D     (None, 24, 32, 128)           295040      ['conv2d_41[0][0]']
 Transpose)

 concatenate_6 (Concatenate     (None, 24, 32, 256)           0           ['conv2d_transpose_6
```
Loading [MathJax]/extensions/Safe.js

```
[0][0]',
 )                                                                'conv2d_35[0][0]']

 conv2d_42 (Conv2D)          (None, 24, 32, 128)          295040    ['concatenate_6[0]
[0]']

 conv2d_43 (Conv2D)          (None, 24, 32, 128)          147584    ['conv2d_42[0][0]']

 conv2d_transpose_7 (Conv2D  (None, 48, 64, 64)           73792     ['conv2d_43[0][0]']
 Transpose)

 concatenate_7 (Concatenate  (None, 48, 64, 128)          0         ['conv2d_transpose_7
[0][0]',
 )                                                                'conv2d_33[0][0]']

 conv2d_44 (Conv2D)          (None, 48, 64, 64)           73792     ['concatenate_7[0]
[0]']

 conv2d_45 (Conv2D)          (None, 48, 64, 64)           36928     ['conv2d_44[0][0]']

 conv2d_transpose_8 (Conv2D  (None, 96, 128, 32)          18464     ['conv2d_45[0][0]']
 Transpose)

 concatenate_8 (Concatenate  (None, 96, 128, 64)          0         ['conv2d_transpose_8
[0][0]',
 )                                                                'conv2d_31[0][0]']

 conv2d_46 (Conv2D)          (None, 96, 128, 32)          18464     ['concatenate_8[0]
[0]']

 conv2d_47 (Conv2D)          (None, 96, 128, 32)          9248      ['conv2d_46[0][0]']

 conv2d_48 (Conv2D)          (None, 96, 128, 32)          9248      ['conv2d_47[0][0]']

 conv2d_49 (Conv2D)          (None, 96, 128, 23)          759       ['conv2d_48[0][0]']

==========================================================================================
==========
Total params: 8640471 (32.96 MB)
Trainable params: 8640471 (32.96 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________
__________
```

Loss Function:

Using sparse categorical crossentropy as loss function, to perform pixel-wise multiclass prediction.

In [20]:
```
unet.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Dataset Handling:

Defining a function to display both an input image, and its ground truth: the true mask

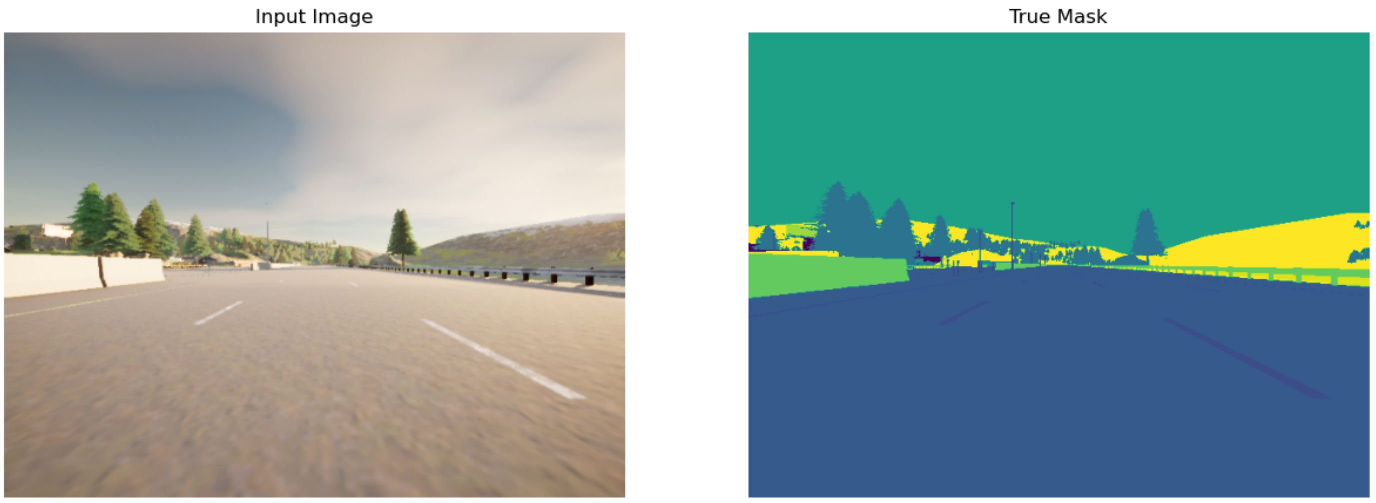
In [21]:
```
def display(display_list):
    plt.figure(figsize=(15, 15))

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

    for i in range(len(display_list)):
```

Loading [MathJax]/extensions/Safe.js

```
        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()
```

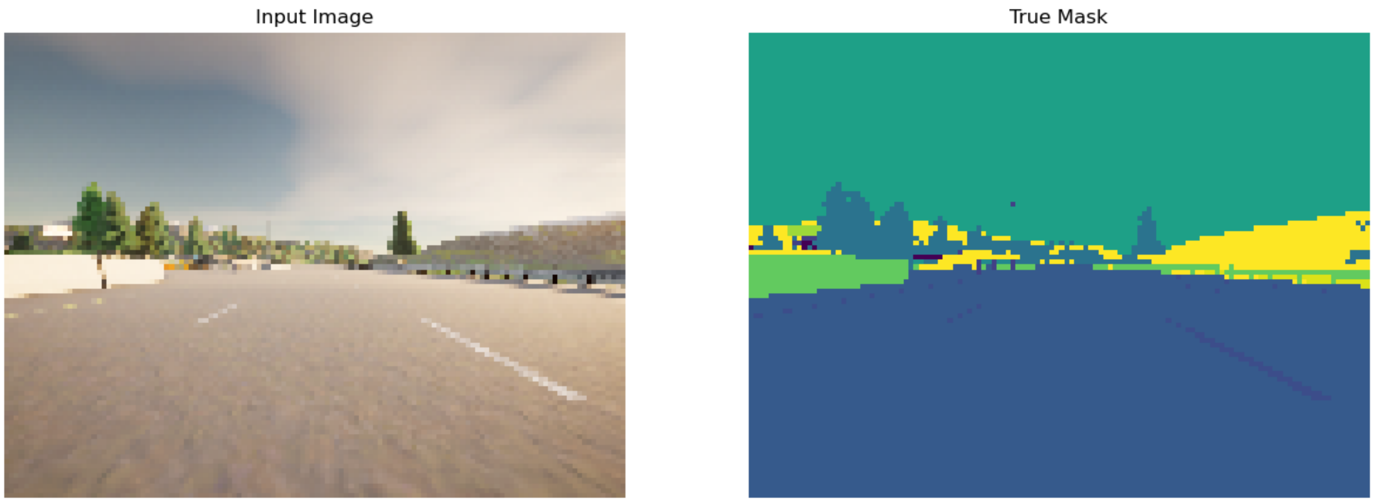
In [22]:
```
for image, mask in image_ds.take(1):
    sample_image, sample_mask = image, mask
    print(mask.shape)
display([sample_image, sample_mask])
```

```
(480, 640, 1)
```



In [23]:
```
for image, mask in processed_image_ds.take(1):
    sample_image, sample_mask = image, mask
    print(mask.shape)
display([sample_image, sample_mask])
```

```
(96, 128, 1)
```



3- Train the Model

In [25]:
```
EPOCHS = 5
VAL_SUBSPLITS = 5
BUFFER_SIZE = 500
BATCH_SIZE = 32
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)
```

Loading [MathJax]/extensions/Safe.js

```
(TensorSpec(shape=(96, 128, 3), dtype=tf.float32, name=None), TensorSpec(shape=(96, 128,
1), dtype=tf.uint8, name=None))
Epoch 1/5
34/34 [==============================] - 30s 874ms/step - loss: 0.4168 - accuracy: 0.877
5
Epoch 2/5
34/34 [==============================] - 29s 859ms/step - loss: 0.3562 - accuracy: 0.894
0
Epoch 3/5
34/34 [==============================] - 29s 866ms/step - loss: 0.3376 - accuracy: 0.897
8
Epoch 4/5
34/34 [==============================] - 29s 860ms/step - loss: 0.2981 - accuracy: 0.911
1
Epoch 5/5
34/34 [==============================] - 29s 864ms/step - loss: 0.2679 - accuracy: 0.920
9
```

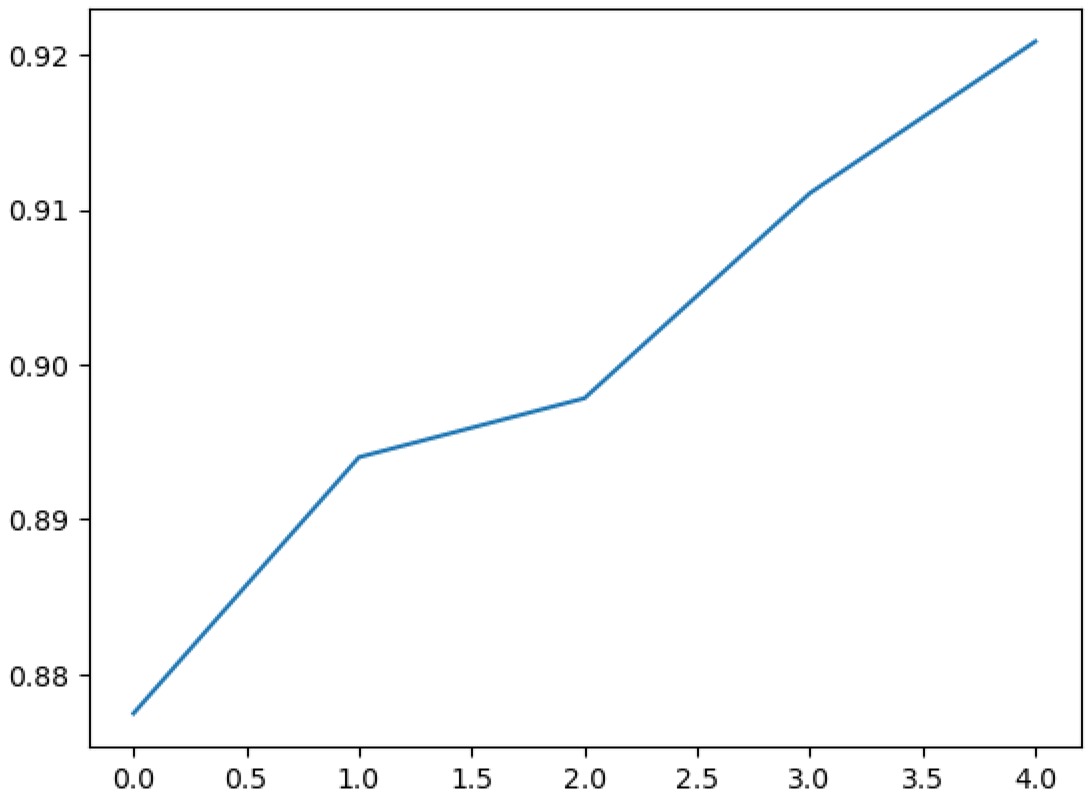
Create Predicted Masks:

In [26]:
```python
def create_mask(pred_mask):
    pred_mask = tf.argmax(pred_mask, axis=-1)
    pred_mask = pred_mask[..., tf.newaxis]
    return pred_mask[0]
```

Plot Model Accuracy:

In [27]:
```python
plt.plot(model_history.history["accuracy"])
```

Out[27]:
```
[<matplotlib.lines.Line2D at 0x1a49388d5b0>]
```



Show Model Predictions:

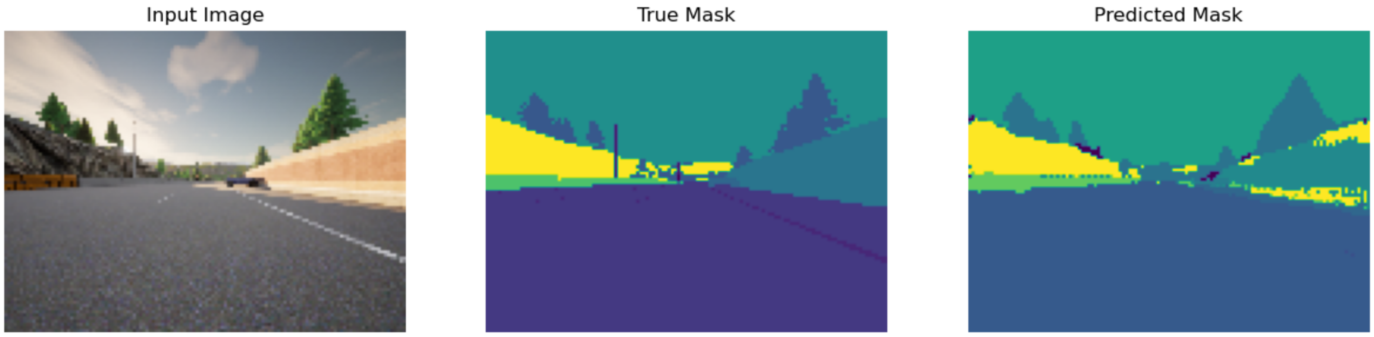
Checking predicted masks against the true mask and the original input image:

Loading [MathJax]/extensions/Safe.js

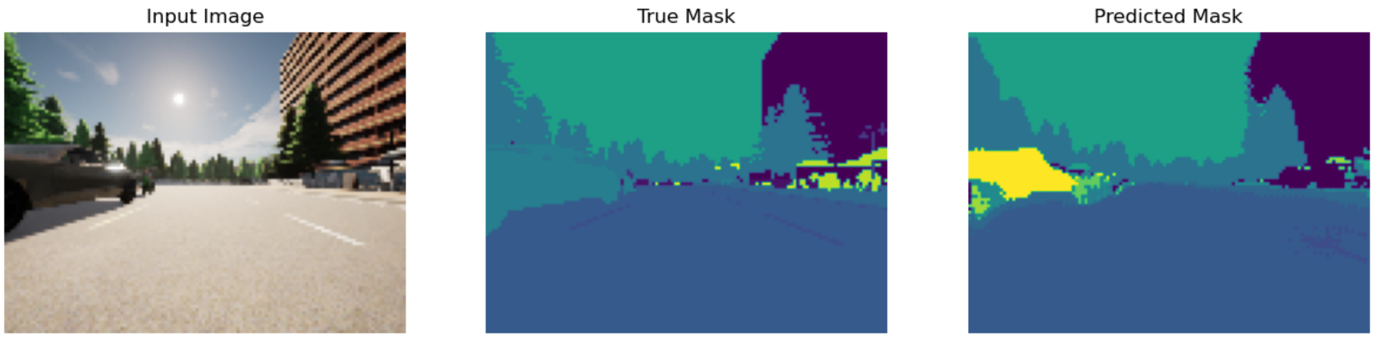
In [28]:
```python
def show_predictions(dataset=None, num=1):
    """
    Displays the first image of each of the num batches
    """
    if dataset:
        for image, mask in dataset.take(num):
            pred_mask = unet.predict(image)
            display([image[0], mask[0], create_mask(pred_mask)])
    else:
        display([sample_image, sample_mask,
             create_mask(unet.predict(sample_image[tf.newaxis, ...]))])
```

In [29]:
```python
show_predictions(train_dataset, 6)
```

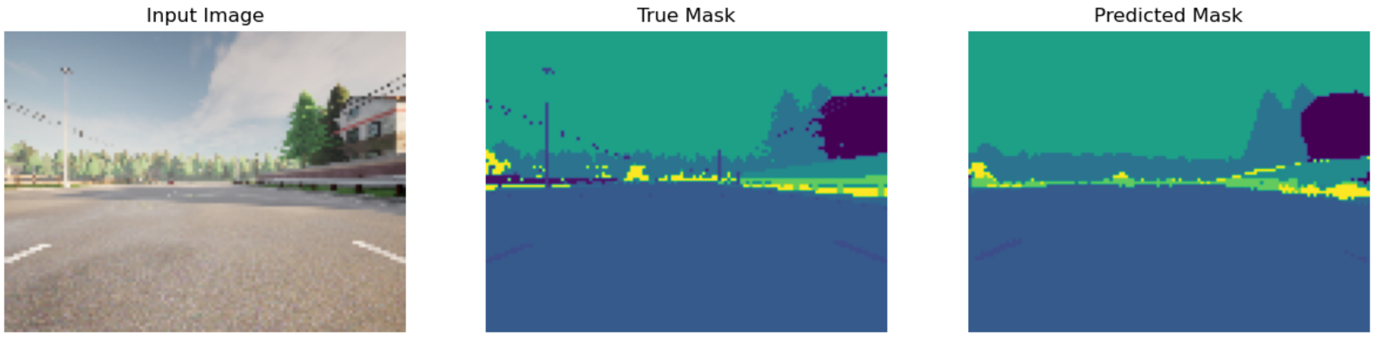
```
1/1 [==============================] - 0s 395ms/step
```



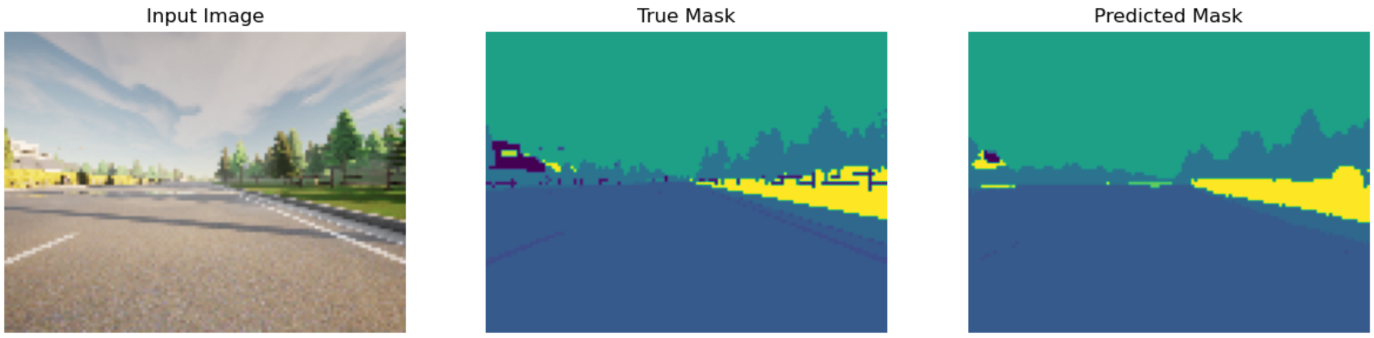
```
1/1 [==============================] - 0s 239ms/step
```



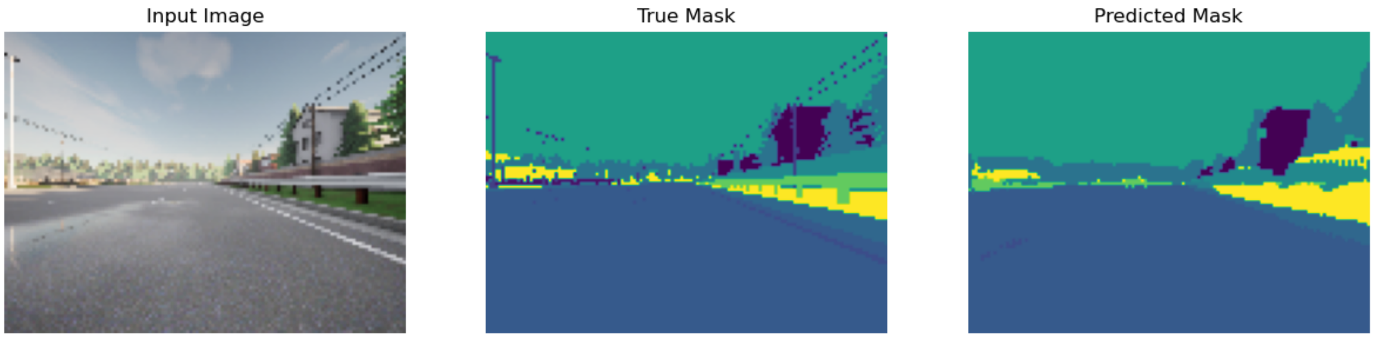
```
1/1 [==============================] - 0s 255ms/step
```



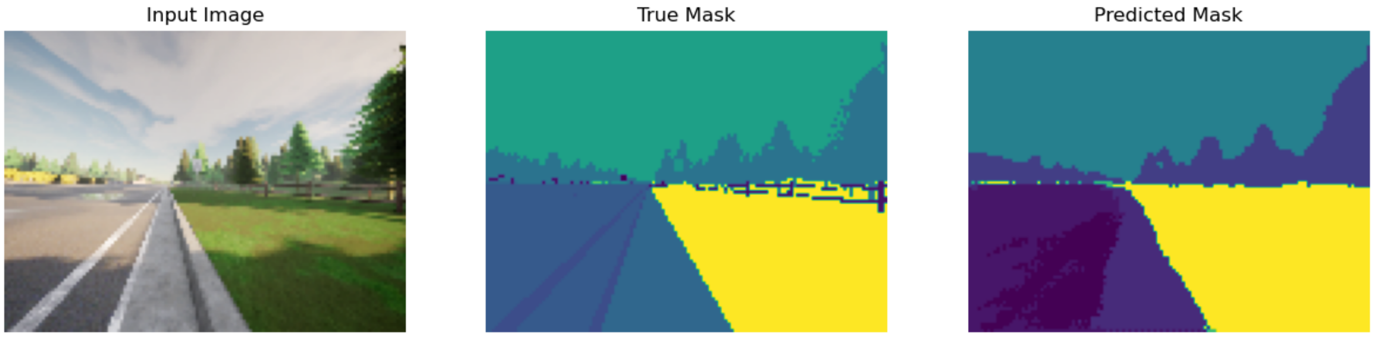
```
1/1 [==============================] - 0s 263ms/step
```



```
1/1 [==============================] - 0s 252ms/step
```



```
1/1 [==============================] - 0s 262ms/step
```



```
In [ ]:
```

Loading [MathJax]/extensions/Safe.js