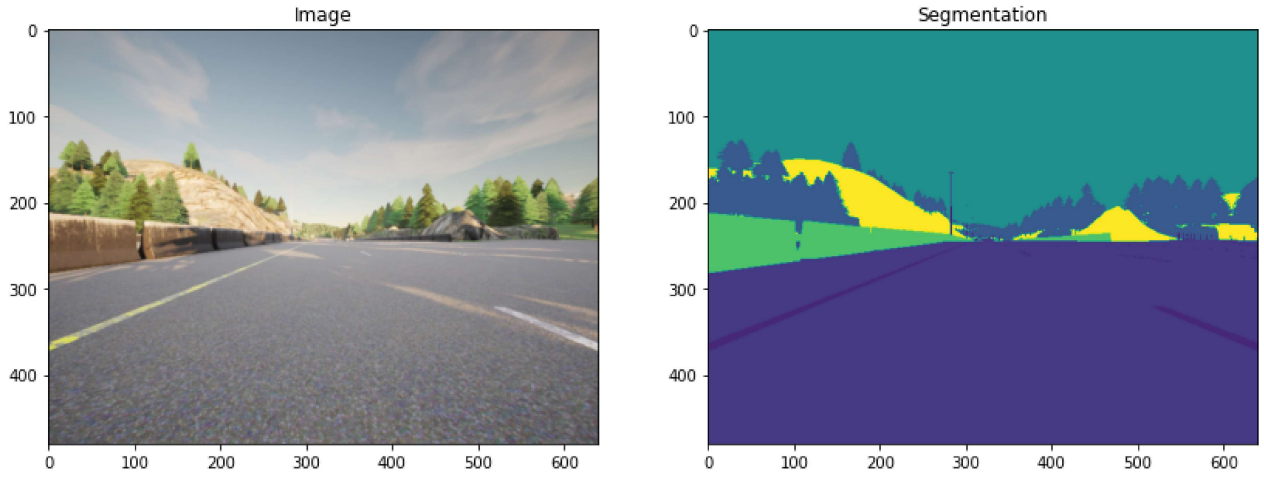
In [2]:
```
import tensorflow as tf
import numpy as np

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 import summary, comparator
```

In [3]:
```
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 = ''
image_path = os.path.join(path, './data/CameraRGB/')
mask_path = os.path.join(path, './data/CameraMask/')
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]
```

In [4]:
```
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.sha

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')
```

Out[4]:
```
Text(0.5, 1.0, 'Segmentation')
```



2.1 - Split Your 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'./data/CameraRGB/000026.png'>, <tf.Tensor:
shape=(), dtype=string, numpy=b'./data/CameraMask/000026.png'>)
(<tf.Tensor: shape=(), dtype=string, numpy=b'./data/CameraRGB/000027.png'>, <tf.Tensor:
shape=(), dtype=string, numpy=b'./data/CameraMask/000027.png'>)
(<tf.Tensor: shape=(), dtype=string, numpy=b'./data/CameraRGB/000028.png'>, <tf.Tensor:
shape=(), dtype=string, numpy=b'./data/CameraMask/000028.png'>)
```

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'./data/CameraRGB/002128.png', shape=(), dtype=string)
tf.Tensor(b'./data/CameraMask/002128.png', shape=(), dtype=string)
```

In [7]:
```
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)
```

In [8]:

```
# UNQ_C1
# GRADED FUNCTION: conv_block
def conv_block(inputs=None, n_filters=32, dropout_prob=0, max_pooling=True):
    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)

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


    if max_pooling:
        next_layer = MaxPooling2D()(conv)


    else:
        next_layer = conv

    skip_connection = conv

    return next_layer, skip_connection
```

In [9]:

```
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:')
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!
```

In [10]:

```
# UNQ_C2
# GRADED FUNCTION: upsampling_block
def upsampling_block(expansive_input, contractive_input, n_filters=32):
    up = Conv2DTranspose(
                 n_filters,    # number of filters
                 3,    # Kernel size
                 strides=(2,2),
                 padding='same')(expansive_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 [11]:

```
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!
```

In [12]:

```
# UNQ_C3
# GRADED FUNCTION: unet_model
def unet_model(input_size=(96, 128, 3), n_filters=32, n_classes=23):
    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) # Include a dropo

    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
```

In [13]:

```
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)
```

```
All tests passed!
```

In [14]:
```
img_height = 96
img_width = 128
num_channels = 3

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

In [15]:
```
unet.summary()
```

```
Model: "functional_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_6 (InputLayer)         [(None, 96, 128, 3)]      0         
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 96, 128, 32)       896       
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 96, 128, 32)       9248      
_________________________________________________________________
max_pooling2d_6 (MaxPooling2D (None, 48, 64, 32)       0         
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 48, 64, 64)        18496     
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 48, 64, 64)        36928     
_________________________________________________________________
max_pooling2d_7 (MaxPooling2D (None, 24, 32, 64)       0         
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 24, 32, 128)       73856     
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 24, 32, 128)       147584    
_________________________________________________________________
max_pooling2d_8 (MaxPooling2D (None, 12, 16, 128)      0         
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 12, 16, 256)       295168    
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 12, 16, 256)       590080    
_________________________________________________________________
dropout_3 (Dropout)          (None, 12, 16, 256)       0         
_________________________________________________________________
max_pooling2d_9 (MaxPooling2D (None, 6, 8, 256)        0         
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 6, 8, 512)         1180160   
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 6, 8, 512)         2359808   
```

dropout_4 (Dropout)	(None, 6, 8, 512)	0	conv2d_35[0][0]
conv2d_transpose_5 (Conv2DTrans	(None, 12, 16, 256)	1179904	dropout_4[0][0]
concatenate_5 (Concatenate)[0]	(None, 12, 16, 512)	0	conv2d_transpose_5[0] dropout_3[0][0]
conv2d_36 (Conv2D)	(None, 12, 16, 256)	1179904	concatenate_5[0][0]
conv2d_37 (Conv2D)	(None, 12, 16, 256)	590080	conv2d_36[0][0]
conv2d_transpose_6 (Conv2DTrans	(None, 24, 32, 128)	295040	conv2d_37[0][0]
concatenate_6 (Concatenate)[0]	(None, 24, 32, 256)	0	conv2d_transpose_6[0] conv2d_31[0][0]
conv2d_38 (Conv2D)	(None, 24, 32, 128)	295040	concatenate_6[0][0]
conv2d_39 (Conv2D)	(None, 24, 32, 128)	147584	conv2d_38[0][0]
conv2d_transpose_7 (Conv2DTrans	(None, 48, 64, 64)	73792	conv2d_39[0][0]
concatenate_7 (Concatenate)[0]	(None, 48, 64, 128)	0	conv2d_transpose_7[0] conv2d_29[0][0]
conv2d_40 (Conv2D)	(None, 48, 64, 64)	73792	concatenate_7[0][0]
conv2d_41 (Conv2D)	(None, 48, 64, 64)	36928	conv2d_40[0][0]
conv2d_transpose_8 (Conv2DTrans	(None, 96, 128, 32)	18464	conv2d_41[0][0]
concatenate_8 (Concatenate)[0]	(None, 96, 128, 64)	0	conv2d_transpose_8[0] conv2d_27[0][0]
conv2d_42 (Conv2D)	(None, 96, 128, 32)	18464	concatenate_8[0][0]
conv2d_43 (Conv2D)	(None, 96, 128, 32)	9248	conv2d_42[0][0]
conv2d_44 (Conv2D)	(None, 96, 128, 32)	9248	conv2d_43[0][0]
conv2d_45 (Conv2D)	(None, 96, 128, 23)	759	conv2d_44[0][0]

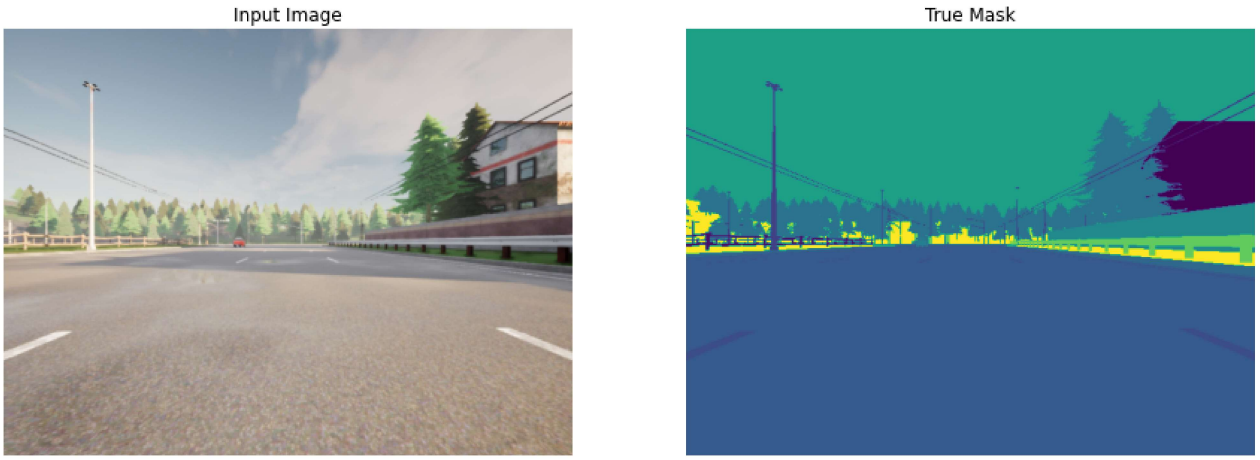
```
==========
Total params: 8,640,471
Trainable params: 8,640,471
Non-trainable params: 0
```

In [16]:
```
unet.compile(optimizer='adam',
             loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
             metrics=['accuracy'])
```

In [17]:
```
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()
```

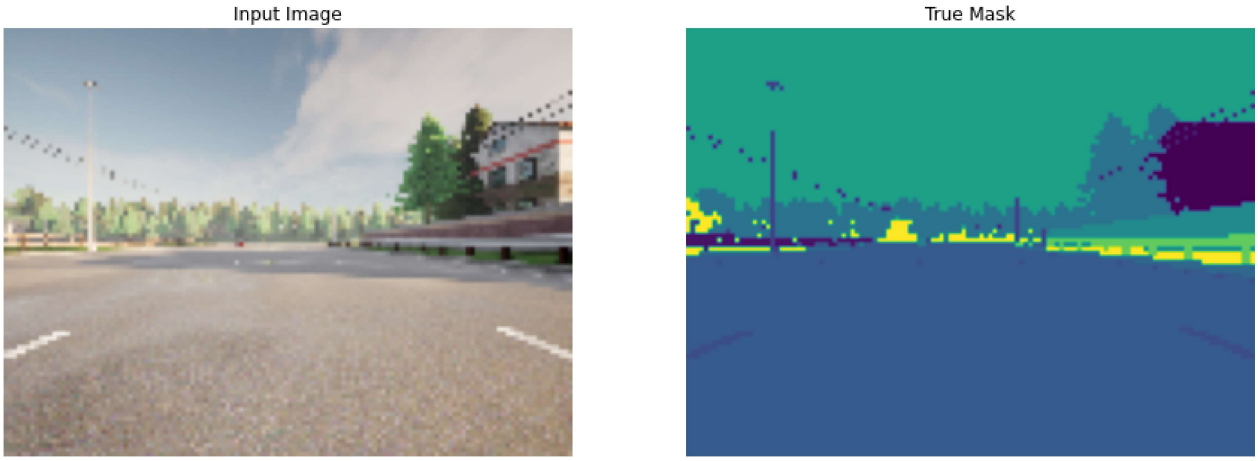
In [18]:
```
for image, mask in image_ds.take(1):
    sample_image, sample_mask = image, mask
    print(mask.shape)
display([sample_image, sample_mask])
```

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



In [19]:
```
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)
```



In [22]:
```
train_dataset
```

Out[22]:
```
<BatchDataset shapes: ((None, 96, 128, 3), (None, 96, 128, 1)), types: (tf.float32, tf.uint8)>
```

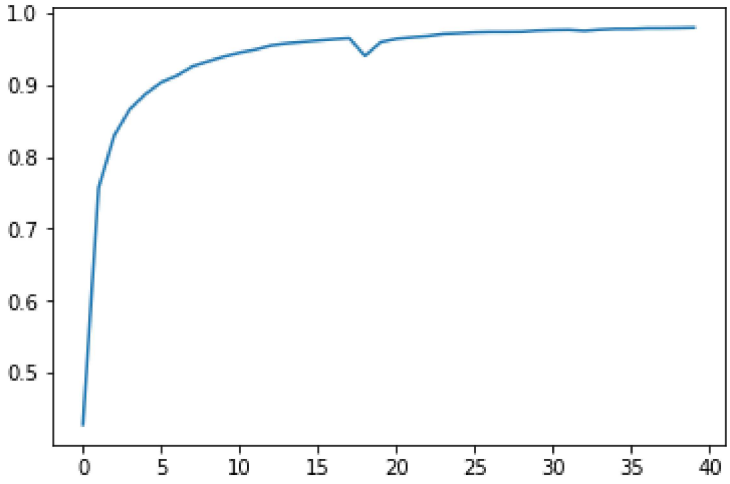
In [21]:
```
EPOCHS = 40
VAL_SUBSPLITS = 5
BUFFER_SIZE = 500
BATCH_SIZE = 32
processed_image_ds.batch(BATCH_SIZE)
train_dataset = processed_image_ds.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
print(processed_image_ds.element_spec)
# model_history = unet.fit(train_dataset, epochs=EPOCHS)
```

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

In [23]:
```
def create_mask(pred_mask):
    pred_mask = tf.argmax(pred_mask, axis=-1)
    pred_mask = pred_mask[..., tf.newaxis]
    return pred_mask[0]
```

In [24]:
```
plt.plot(model_history.history["accuracy"])
```

Out[24]:
```
[<matplotlib.lines.Line2D at 0x7f82845ffa58>]
```

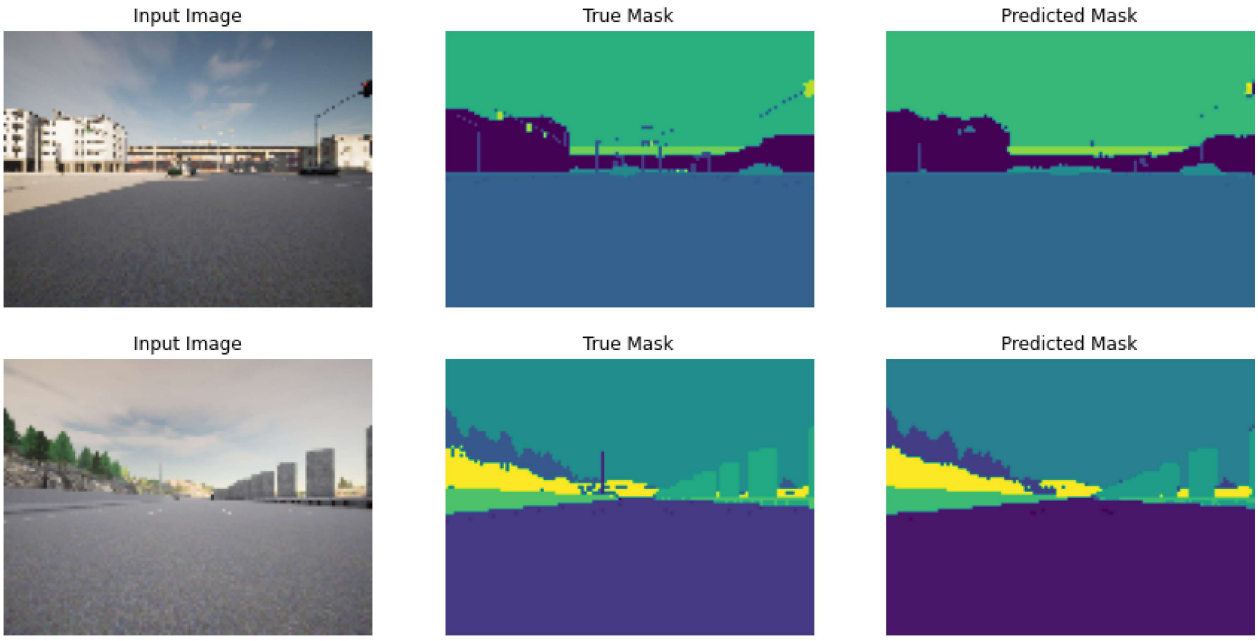


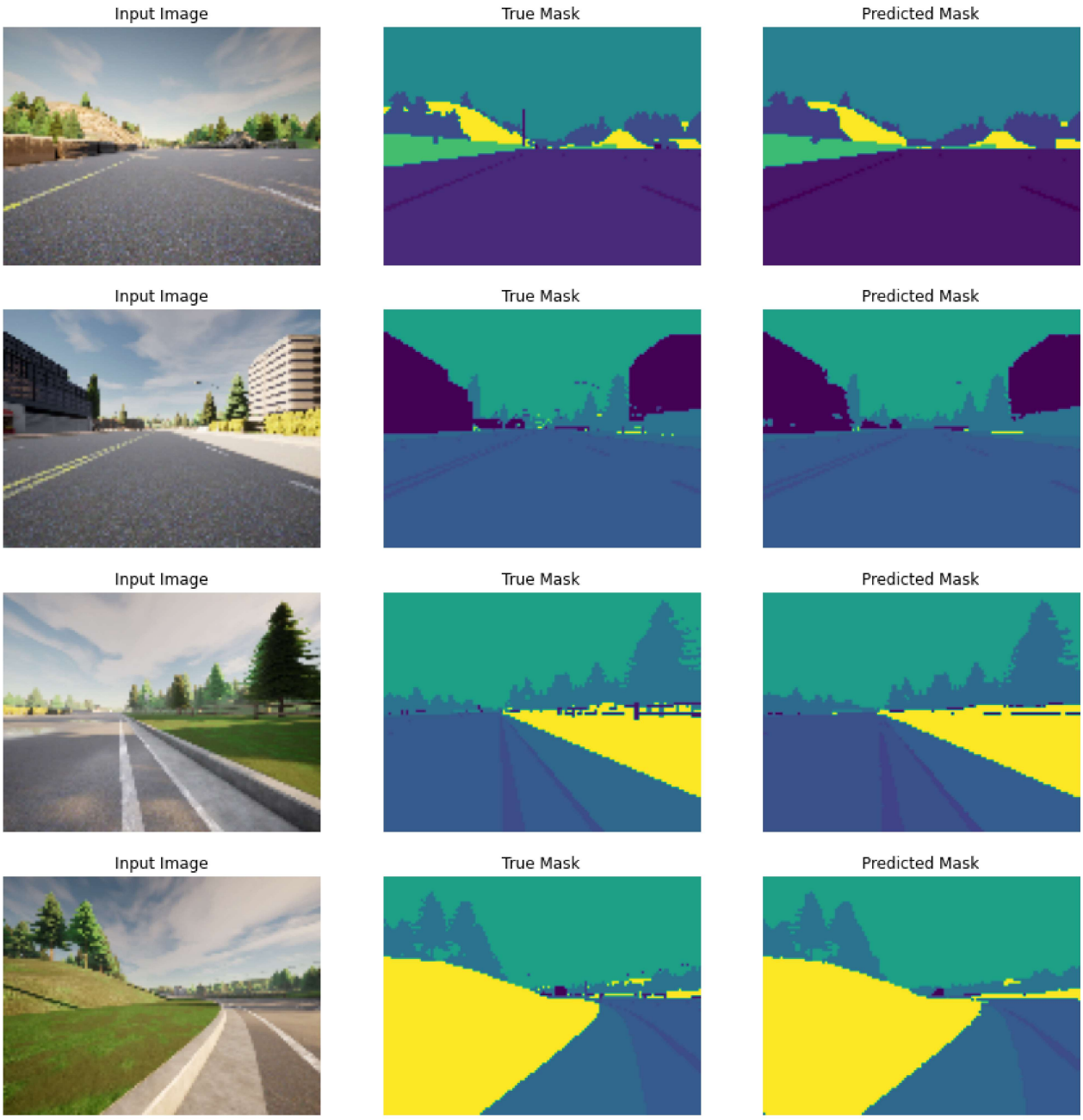
In [25]:

```
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 [26]:

```
show_predictions(train_dataset, 6)
```




Input Image
True Mask
Predicted Mask
Input Image
True Mask
Predicted Mask
Input Image
True Mask
Predicted Mask
Input Image
True Mask
Predicted Mask