Forest Image Segmentation - UNet

In [9]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import random
import cv2
import tensorflow as tf
from tensorflow.python.keras.models import Sequential
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation, MaxPool
from tensorflow.keras.models import Model
```

Directory Setup

In [11]:
```
base_directory = r'D:\data science\Practice\Unet\Forest Segmented\Forest Segmented'
images_folder = os.path.join(base_directory, 'image')
masks_folder = os.path.join(base_directory, 'mask')
data = pd.read_csv(os.path.join(base_directory, 'meta_data.csv'))
```

Data Augmentation

In [30]:
```
img_dim = 256

image_datagen = ImageDataGenerator(rescale=1./255,validation_split=0.15)
mask_datagen = ImageDataGenerator(rescale=1./255,validation_split=0.15)

train_image_generator = image_datagen.flow_from_directory(
        r'D:\data science\Practice\Unet\Forest Segmented\Forest Segmented',
        target_size=(img_dim, img_dim),
        class_mode = None,
        classes = ['image'],
        batch_size = 8,
        seed=42,
        subset='training')

train_mask_generator = mask_datagen.flow_from_directory(
        r'D:\data science\Practice\Unet\Forest Segmented\Forest Segmented',
        target_size=(img_dim, img_dim),
        class_mode = None,
        classes = ['mask'],
        color_mode = 'grayscale',
        batch_size = 8,
        seed=42,
        subset='training')
```

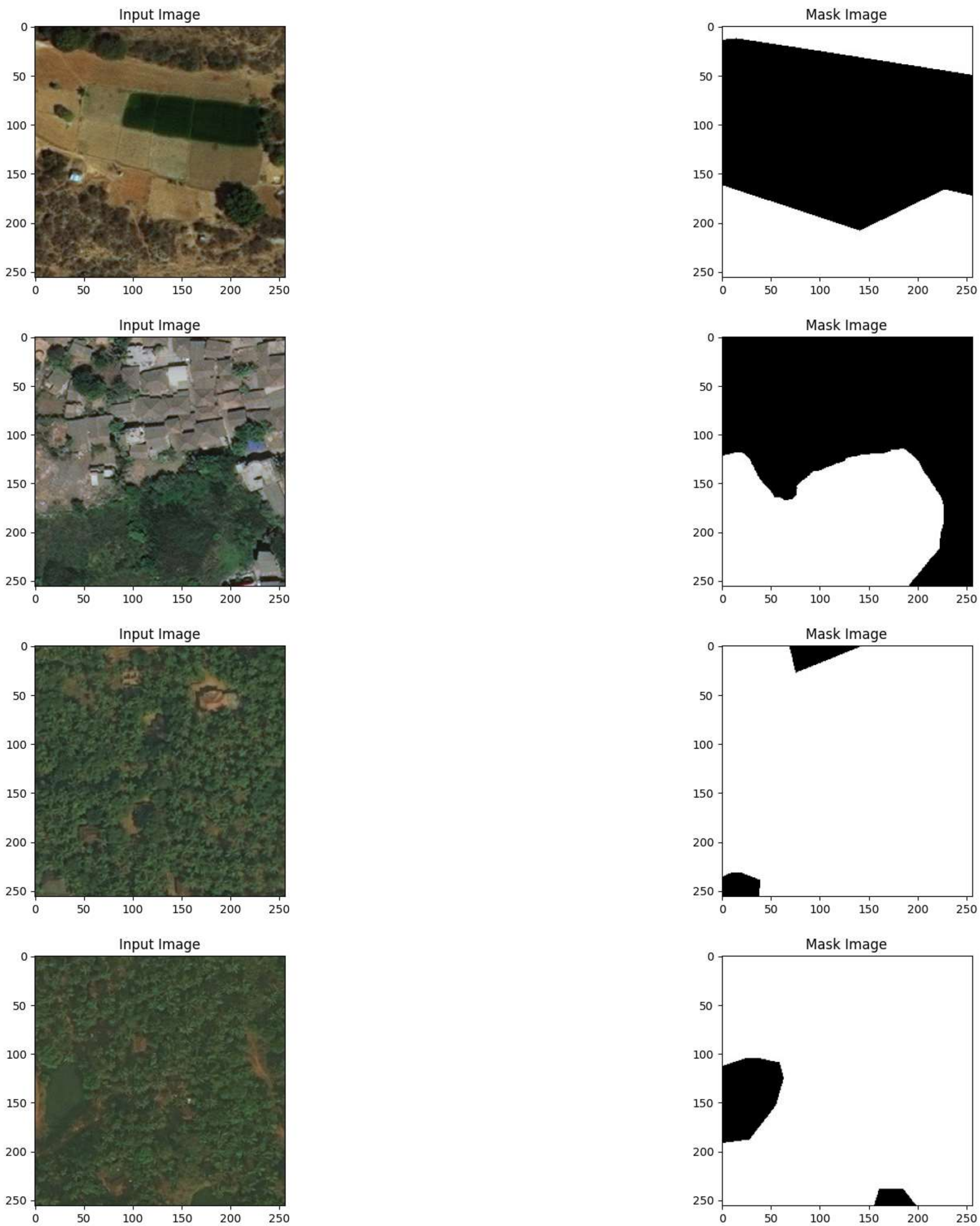
```
val_image_generator = image_datagen.flow_from_directory(
        r'D:\data science\Practice\Unet\Forest Segmented\Forest Segmented',
        target_size=(img_dim, img_dim),
        class_mode = None,
        classes = ['image'],
        batch_size = 8,
        seed=42,
        subset='validation')

val_mask_generator = mask_datagen.flow_from_directory(
        r'D:\data science\Practice\Unet\Forest Segmented\Forest Segmented',
        target_size=(img_dim, img_dim),
        class_mode = None,
        classes = ['mask'],
        color_mode = 'grayscale',
        batch_size = 8,
        seed=42,
        subset='validation')

train_generator = zip(train_image_generator, train_mask_generator)
val_generator = zip(val_image_generator, val_mask_generator)
```

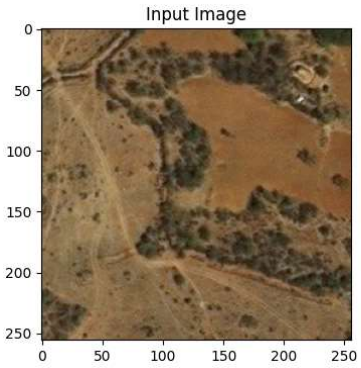
```
Found 4342 images belonging to 1 classes.
Found 4342 images belonging to 1 classes.
Found 766 images belonging to 1 classes.
Found 766 images belonging to 1 classes.
```

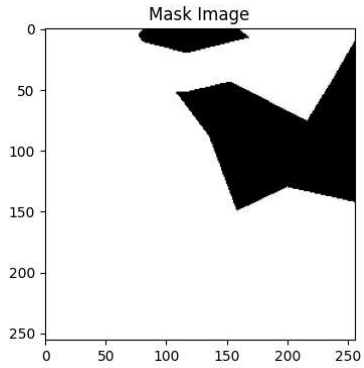
Display Sample Training Images

In [31]:
```
training_samples_size = train_image_generator.samples
val_samples_size = val_image_generator.samples
```

In [32]:
```
n = 0
for i,m in train_generator:
    img,mask = i,m

    if n < 5:
        fig, axs = plt.subplots(1 , 2, figsize=(20,4))
        axs[0].imshow(img[0])
        axs[0].set_title('Input Image')
        axs[1].imshow(mask[0],cmap='gray')
        axs[1].set_title('Mask Image')
        plt.show()
        n+=1
    else:
        break
```


Input Image
Mask Image
Input Image
Mask Image
Input Image
Mask Image
Input Image
Mask Image





Unet Model

In [33]:

```
def build_unet(input_shape):
    def conv_block(input, num_filters):
        x = Conv2D(num_filters, 3, padding="same")(input)
        x = BatchNormalization()(x)
        x = Activation("relu")(x)

        x = Conv2D(num_filters, 3, padding="same")(x)
        x = BatchNormalization()(x)
        x = Activation("relu")(x)

        return x

    def encoder_block(input, num_filters):
        x = conv_block(input, num_filters)
        p = MaxPool2D((2, 2))(x)
        return x, p

    def decoder_block(input, skip_features, num_filters):
        x = Conv2DTranspose(num_filters, (2, 2), strides=2, padding="same")(input)
        x = Concatenate()([x, skip_features])
        x = conv_block(x, num_filters)
        return x

    inputs = Input(input_shape)

    s1, p1 = encoder_block(inputs, 32)
    s2, p2 = encoder_block(p1, 64)
    s3, p3 = encoder_block(p2, 128)
    s4, p4 = encoder_block(p3, 256)

    b1 = conv_block(p4, 512)

    d1 = decoder_block(b1, s4, 256)
    d2 = decoder_block(d1, s3, 128)
    d3 = decoder_block(d2, s2, 64)
    d4 = decoder_block(d3, s1, 32)

    outputs = Conv2D(1, 1, padding="same", activation="sigmoid")(d4)
```

```
    model = Model(inputs, outputs, name="U-Net")
    return model
```

In [34]:
```
input_shape = (img_dim, img_dim, 3)
model = build_unet(input_shape)
model.compile(optimizer = tf.keras.optimizers.Adam(lr = 0.001), loss = ['binary_cro
model.summary()
```

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

 conv2d_57 (Conv2D)             (None, 256, 256, 32  896         ['input_4[0][0]']
                                )

 batch_normalization_54 (BatchN  (None, 256, 256, 32  128        ['conv2d_57[0][0]']
 ormalization)                  )

 activation_54 (Activation)     (None, 256, 256, 32  0           ['batch_normalizati
on_54[0][0]']
                                )

 conv2d_58 (Conv2D)             (None, 256, 256, 32  9248        ['activation_54[0]
[0]']
                                )

 batch_normalization_55 (BatchN  (None, 256, 256, 32  128        ['conv2d_58[0][0]']
 ormalization)                  )

 activation_55 (Activation)     (None, 256, 256, 32  0           ['batch_normalizati
on_55[0][0]']
                                )

 max_pooling2d_12 (MaxPooling2D  (None, 128, 128, 32  0          ['activation_55[0]
[0]']
 )                              )

 conv2d_59 (Conv2D)             (None, 128, 128, 64  18496       ['max_pooling2d_12
[0][0]']
                                )

 batch_normalization_56 (BatchN  (None, 128, 128, 64  256        ['conv2d_59[0][0]']
 ormalization)                  )

 activation_56 (Activation)     (None, 128, 128, 64  0           ['batch_normalizati
on_56[0][0]']
                                )

 conv2d_60 (Conv2D)             (None, 128, 128, 64  36928       ['activation_56[0]
[0]']
                                )

 batch_normalization_57 (BatchN  (None, 128, 128, 64  256        ['conv2d_60[0][0]']
 ormalization)                  )

 activation_57 (Activation)     (None, 128, 128, 64  0           ['batch_normalizati
on_57[0][0]']
                                )
```

```
 max_pooling2d_13 (MaxPooling2D  (None, 64, 64, 64)  0           ['activation_57[0]
[0]']
 )

 conv2d_61 (Conv2D)             (None, 64, 64, 128)  73856       ['max_pooling2d_13
[0][0]']

 batch_normalization_58 (BatchN  (None, 64, 64, 128)  512        ['conv2d_61[0][0]']
 ormalization)

 activation_58 (Activation)     (None, 64, 64, 128)  0           ['batch_normalizati
on_58[0][0]']

 conv2d_62 (Conv2D)             (None, 64, 64, 128)  147584      ['activation_58[0]
[0]']

 batch_normalization_59 (BatchN  (None, 64, 64, 128)  512        ['conv2d_62[0][0]']
 ormalization)

 activation_59 (Activation)     (None, 64, 64, 128)  0           ['batch_normalizati
on_59[0][0]']

 max_pooling2d_14 (MaxPooling2D  (None, 32, 32, 128)  0          ['activation_59[0]
[0]']
 )

 conv2d_63 (Conv2D)             (None, 32, 32, 256)  295168      ['max_pooling2d_14
[0][0]']

 batch_normalization_60 (BatchN  (None, 32, 32, 256)  1024       ['conv2d_63[0][0]']
 ormalization)

 activation_60 (Activation)     (None, 32, 32, 256)  0           ['batch_normalizati
on_60[0][0]']

 conv2d_64 (Conv2D)             (None, 32, 32, 256)  590080      ['activation_60[0]
[0]']

 batch_normalization_61 (BatchN  (None, 32, 32, 256)  1024       ['conv2d_64[0][0]']
 ormalization)

 activation_61 (Activation)     (None, 32, 32, 256)  0           ['batch_normalizati
on_61[0][0]']

 max_pooling2d_15 (MaxPooling2D  (None, 16, 16, 256)  0          ['activation_61[0]
[0]']
 )

 conv2d_65 (Conv2D)             (None, 16, 16, 512)  1180160     ['max_pooling2d_15
[0][0]']

 batch_normalization_62 (BatchN  (None, 16, 16, 512)  2048       ['conv2d_65[0][0]']
 ormalization)

 activation_62 (Activation)     (None, 16, 16, 512)  0           ['batch_normalizati
on_62[0][0]']
```

```
 conv2d_66 (Conv2D)             (None, 16, 16, 512)  2359808     ['activation_62[0]
[0]']

 batch_normalization_63 (BatchN  (None, 16, 16, 512)  2048       ['conv2d_66[0][0]']
 ormalization)

 activation_63 (Activation)     (None, 16, 16, 512)  0           ['batch_normalizati
on_63[0][0]']

 conv2d_transpose_12 (Conv2DTra  (None, 32, 32, 256)  524544     ['activation_63[0]
[0]']
 nspose)

 concatenate_12 (Concatenate)   (None, 32, 32, 512)  0           ['conv2d_transpose_
12[0][0]',
                                                                  'activation_61[0]
[0]']

 conv2d_67 (Conv2D)             (None, 32, 32, 256)  1179904     ['concatenate_12[0]
[0]']

 batch_normalization_64 (BatchN  (None, 32, 32, 256)  1024       ['conv2d_67[0][0]']
 ormalization)

 activation_64 (Activation)     (None, 32, 32, 256)  0           ['batch_normalizati
on_64[0][0]']

 conv2d_68 (Conv2D)             (None, 32, 32, 256)  590080      ['activation_64[0]
[0]']

 batch_normalization_65 (BatchN  (None, 32, 32, 256)  1024       ['conv2d_68[0][0]']
 ormalization)

 activation_65 (Activation)     (None, 32, 32, 256)  0           ['batch_normalizati
on_65[0][0]']

 conv2d_transpose_13 (Conv2DTra  (None, 64, 64, 128)  131200     ['activation_65[0]
[0]']
 nspose)

 concatenate_13 (Concatenate)   (None, 64, 64, 256)  0           ['conv2d_transpose_
13[0][0]',
                                                                  'activation_59[0]
[0]']

 conv2d_69 (Conv2D)             (None, 64, 64, 128)  295040      ['concatenate_13[0]
[0]']

 batch_normalization_66 (BatchN  (None, 64, 64, 128)  512        ['conv2d_69[0][0]']
 ormalization)

 activation_66 (Activation)     (None, 64, 64, 128)  0           ['batch_normalizati
on_66[0][0]']

 conv2d_70 (Conv2D)             (None, 64, 64, 128)  147584      ['activation_66[0]
```

```
[0]']

 batch_normalization_67 (BatchN  (None, 64, 64, 128)  512        ['conv2d_70[0][0]']
 ormalization)

 activation_67 (Activation)     (None, 64, 64, 128)  0          ['batch_normalizati
on_67[0][0]']

 conv2d_transpose_14 (Conv2DTra  (None, 128, 128, 64  32832      ['activation_67[0]
[0]']
 nspose)                        )

 concatenate_14 (Concatenate)   (None, 128, 128, 12  0          ['conv2d_transpose_
14[0][0]',
                                8)                               'activation_57[0]
[0]']

 conv2d_71 (Conv2D)             (None, 128, 128, 64  73792       ['concatenate_14[0]
[0]']
                                )

 batch_normalization_68 (BatchN  (None, 128, 128, 64  256        ['conv2d_71[0][0]']
 ormalization)                  )

 activation_68 (Activation)     (None, 128, 128, 64  0          ['batch_normalizati
on_68[0][0]']
                                )

 conv2d_72 (Conv2D)             (None, 128, 128, 64  36928       ['activation_68[0]
[0]']
                                )

 batch_normalization_69 (BatchN  (None, 128, 128, 64  256        ['conv2d_72[0][0]']
 ormalization)                  )

 activation_69 (Activation)     (None, 128, 128, 64  0          ['batch_normalizati
on_69[0][0]']
                                )

 conv2d_transpose_15 (Conv2DTra  (None, 256, 256, 32  8224       ['activation_69[0]
[0]']
 nspose)                        )

 concatenate_15 (Concatenate)   (None, 256, 256, 64  0          ['conv2d_transpose_
15[0][0]',
                                )                                'activation_55[0]
[0]']

 conv2d_73 (Conv2D)             (None, 256, 256, 32  18464       ['concatenate_15[0]
[0]']
                                )

 batch_normalization_70 (BatchN  (None, 256, 256, 32  128        ['conv2d_73[0][0]']
 ormalization)                  )

 activation_70 (Activation)     (None, 256, 256, 32  0          ['batch_normalizati
```

```
on_70[0][0]']
                                )

 conv2d_74 (Conv2D)             (None, 256, 256, 32  9248        ['activation_70[0]
[0]']
                                )

 batch_normalization_71 (BatchN  (None, 256, 256, 32  128        ['conv2d_74[0][0]']
 ormalization)                  )

 activation_71 (Activation)     (None, 256, 256, 32  0           ['batch_normalizati
on_71[0][0]']
                                )

 conv2d_75 (Conv2D)             (None, 256, 256, 1)  33          ['activation_71[0]
[0]']

==================================================================================
==============
Total params: 7,771,873
Trainable params: 7,765,985
Non-trainable params: 5,888
__________________________________________________________________________________
```

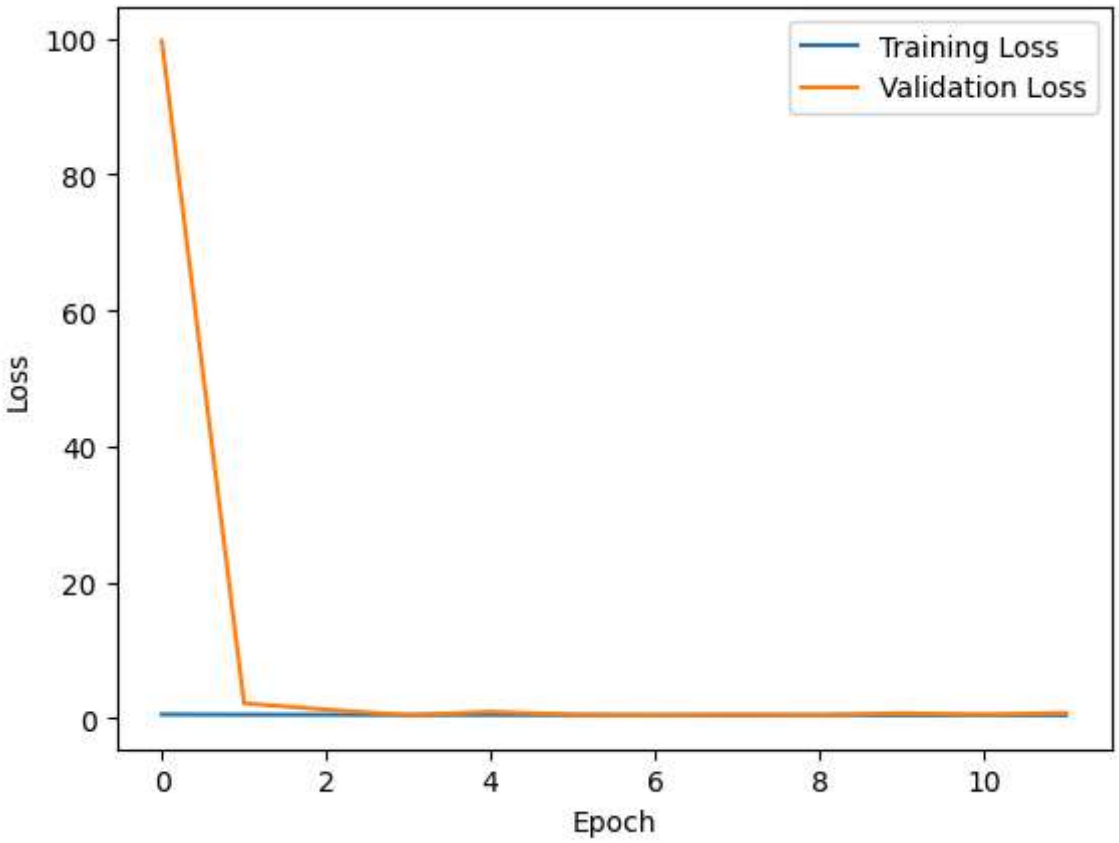
Model Training

In [35]:
```
earlystop = EarlyStopping(monitor = 'val_loss',
                          min_delta = 0,
                          patience = 5,
                          verbose = 1,
                          restore_best_weights = True)



history = model.fit(train_generator,
                    steps_per_epoch=training_samples_size//32,
                    validation_data=val_generator,
                    validation_steps=val_samples_size//32,
                    epochs=50, callbacks=[earlystop])
```

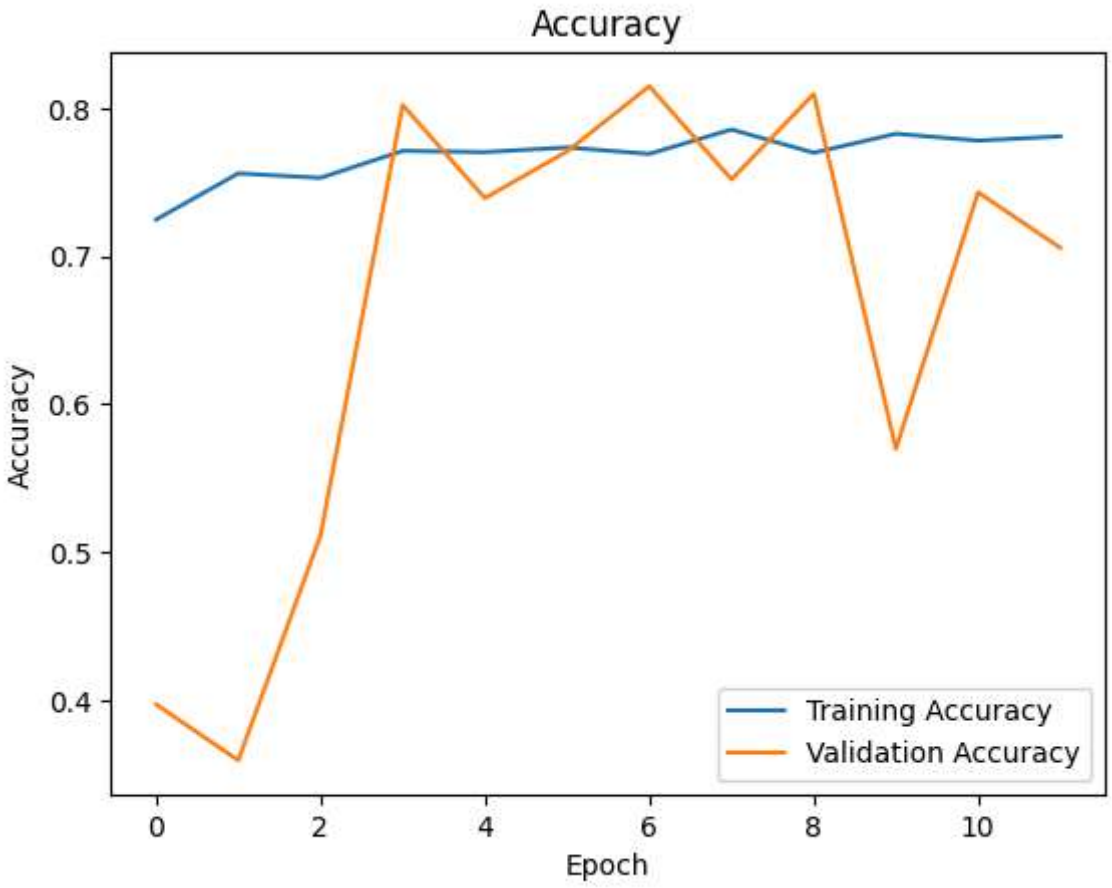
```
Epoch 1/50
135/135 [==============================] - 54s 354ms/step - loss: 0.5592 - accuracy:
0.7247 - val_loss: 99.6630 - val_accuracy: 0.3975
Epoch 2/50
135/135 [==============================] - 47s 351ms/step - loss: 0.5199 - accuracy:
0.7559 - val_loss: 2.1873 - val_accuracy: 0.3597
Epoch 3/50
135/135 [==============================] - 48s 352ms/step - loss: 0.5173 - accuracy:
0.7529 - val_loss: 1.2753 - val_accuracy: 0.5116
Epoch 4/50
135/135 [==============================] - 51s 376ms/step - loss: 0.4886 - accuracy:
0.7711 - val_loss: 0.4415 - val_accuracy: 0.8021
Epoch 5/50
135/135 [==============================] - 48s 353ms/step - loss: 0.4952 - accuracy:
0.7703 - val_loss: 0.9443 - val_accuracy: 0.7393
Epoch 6/50
135/135 [==============================] - 48s 353ms/step - loss: 0.4788 - accuracy:
0.7734 - val_loss: 0.5278 - val_accuracy: 0.7707
Epoch 7/50
135/135 [==============================] - 48s 354ms/step - loss: 0.4873 - accuracy:
0.7691 - val_loss: 0.4221 - val_accuracy: 0.8148
Epoch 8/50
135/135 [==============================] - 48s 353ms/step - loss: 0.4586 - accuracy:
0.7856 - val_loss: 0.5412 - val_accuracy: 0.7518
Epoch 9/50
135/135 [==============================] - 47s 345ms/step - loss: 0.4879 - accuracy:
0.7699 - val_loss: 0.4353 - val_accuracy: 0.8095
Epoch 10/50
135/135 [==============================] - 47s 349ms/step - loss: 0.4616 - accuracy:
0.7827 - val_loss: 0.7347 - val_accuracy: 0.5700
Epoch 11/50
135/135 [==============================] - 48s 354ms/step - loss: 0.4648 - accuracy:
0.7781 - val_loss: 0.5161 - val_accuracy: 0.7432
Epoch 12/50
135/135 [==============================] - ETA: 0s - loss: 0.4684 - accuracy: 0.7809
Restoring model weights from the end of the best epoch: 7.
135/135 [==============================] - 48s 355ms/step - loss: 0.4684 - accuracy:
0.7809 - val_loss: 0.7903 - val_accuracy: 0.7057
Epoch 12: early stopping
```

Results

In [36]:
```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [37]:
```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy')
plt.show()
```



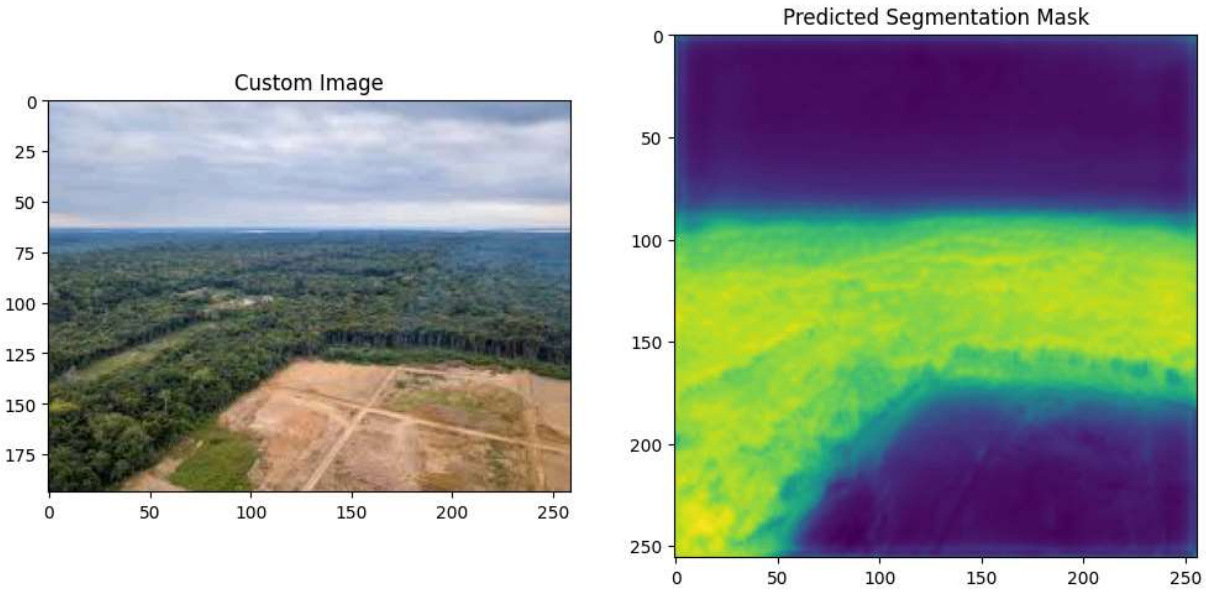
Testing

In [41]:

```
from skimage.io import imread
from skimage.transform import resize


# Function to predict segmentation for a custom image
def predict_segmentation(model, custom_image_path):
    custom_image = imread(custom_image_path)
    custom_image = resize(custom_image, (256, 256), preserve_range=True)  # Resize
    custom_image = custom_image.astype('float32') / 255.0
    custom_image = np.expand_dims(custom_image, axis=0)
    predicted_mask = model.predict(custom_image)
    return predicted_mask[0]

# Path to the custom image
custom_image_path = r"C:\Users\deepchanddc2\Downloads\images.jpg"

# Predict segmentation for the custom image
predicted_mask = predict_segmentation(model, custom_image_path)

# Visualize the custom image and its predicted mask
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(imread(custom_image_path))
plt.title('Custom Image')
```

```
plt.subplot(1, 2, 2)
plt.imshow(predicted_mask[:, :, 0], cmap='viridis')
plt.title('Predicted Segmentation Mask')

plt.show()
```

```
1/1 [==============================] - 0s 21ms/step
```



In [43]:
```
model.save(r'D:\data science\Practice\Unet\Model')

print("Model saved successfully")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 5 of 23). These functions will not be directly callable after loading.
INFO:tensorflow:Assets written to: D:\data science\Practice\Unet\Model\assets
INFO:tensorflow:Assets written to: D:\data science\Practice\Unet\Model\assets
Model saved successfully
```