In [1]:
```
import os
import cv2
import json
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.optimizers import Adam
from PIL import Image
from skimage.transform import resize
from sklearn.model_selection import train_test_split
%matplotlib inline
```

In [5]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [6]:
```
images_dir ='/content/drive/MyDrive/png/train'
masks_dir = '/content/drive/MyDrive/png/train_labels'
val_images_dir ='/content/drive/MyDrive/png/val'
val_masks_dir = '/content/drive/MyDrive/png/val_labels'
test_images_dir='/content/drive/MyDrive/png/test'
test_masks_dir='/content/drive/MyDrive/png/test_labels'
```

In [7]:
```
images_listdir = os.listdir(images_dir)
images_listdir.sort()
masks_listdir = os.listdir(masks_dir)
masks_listdir.sort()
random_images = np.random.choice(masks_listdir, size = 9, replace = False)
random_images.sort()
test_images_listdir = os.listdir(test_images_dir)
test_images_listdir.sort()
test_masks_listdir = os.listdir(test_masks_dir)
test_masks_listdir.sort()
```

In [8]:
```
print(len(images_listdir))
print(len(masks_listdir))
print(len(test_images_listdir))
print(len(test_masks_listdir))
```

```
137
137
10
10
```

In [9]:
```
image_size=512
input_image_size=(512,512)
```

In [10]:
```
def read_image(path):
    img = cv2.imread(path)
    img = cv2.resize(img, (image_size, image_size))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    return img
```

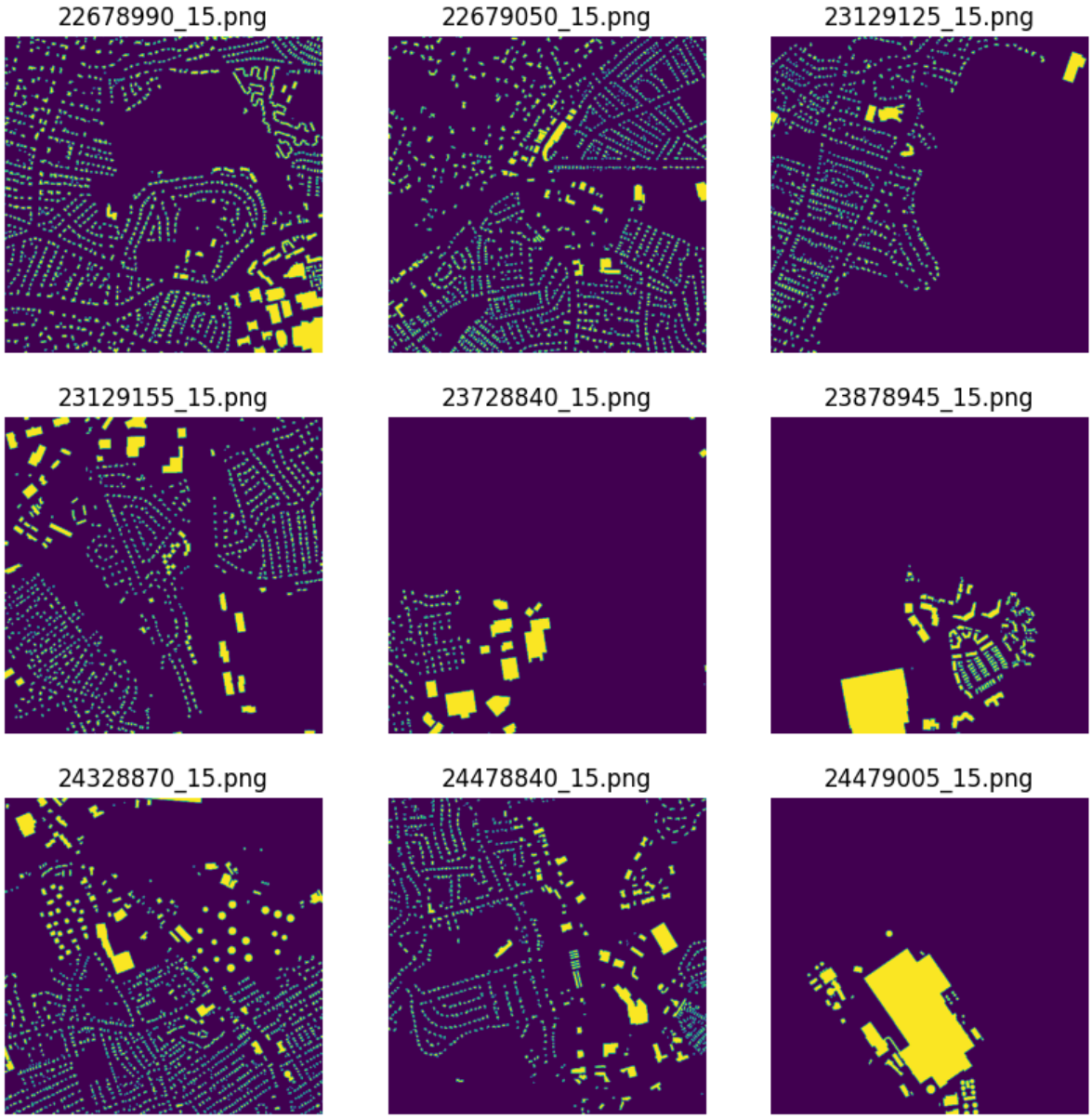
In [11]:
```
number = 30
```

In [12]:
```
rows = 3
cols = 3
fig, ax = plt.subplots(rows, cols, figsize = (10,10))
```

```
for i, ax in enumerate(ax.flat):
    if i < len(random_images):
        img = read_image(f"{images_dir}/{random_images[i]}")
        ax.set_title(f"{random_images[i]}")
        ax.imshow(img)
        ax.axis('off')
```

In [13]:

```
fig, ax = plt.subplots(rows, cols, figsize = (10,10))
for i, ax in enumerate(ax.flat):
    if i < len(random_images):
        file=random_images[i]
        if os.path.exists(os.path.join(masks_dir,file)):
            img = read_image(f"{masks_dir}/{file}")
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            ax.set_title(f"{random_images[i]}")
            ax.imshow(img)
            ax.axis('off')
        else:
            print('not exist')
```



In [14]:
```
MASKS=np.zeros((1,image_size, image_size, 1), dtype=bool)
IMAGES=np.zeros((1,image_size, image_size, 3),dtype=np.uint8)

for j,file in enumerate(masks_listdir[0:number]):   ##the smaller, the faster
    try:
        image = read_image(f"{images_dir}/{file}")
        image_ex = np.expand_dims(image, axis=0)
        IMAGES = np.vstack([IMAGES, image_ex])
        file2=file[0:-4]+'.png'
        mask = read_image(f"{masks_dir}/{file2}")
        mask = cv2.cvtColor(mask, cv2.COLOR_BGR2GRAY)
        mask = mask.reshape(512,512,1)
        mask_ex = np.expand_dims(mask, axis=0)
        MASKS = np.vstack([MASKS, mask_ex])
    except:
        print(file)
        continue
```

In [15]:
```
TMASKS=np.zeros((1,image_size, image_size,1), dtype=bool)
TIMAGES=np.zeros((1,image_size, image_size,3),dtype=np.uint8)

for j,file in enumerate(test_images_listdir):
    try:
        image = read_image(f"{test_images_dir}/{file}")
        image_ex = np.expand_dims(image, axis=0)
```

```
        TIMAGES = np.vstack([TIMAGES, image_ex])
        file2=file[0:-4]+'.png'
        mask = read_image(f"{test_masks_dir}/{file2}")
        mask = cv2.cvtColor(mask, cv2.COLOR_BGR2GRAY)
        mask = mask.reshape(512,512,1)
        mask_ex = np.expand_dims(mask, axis=0)
        TMASKS = np.vstack([TMASKS, mask_ex])
    except:
        print(file)
        continue
```

In [16]:
```
images=np.array(IMAGES)[1:number+1]
masks=np.array(MASKS)[1:number+1]
print(images.shape,masks.shape)

test_images=np.array(TIMAGES)[1:]
test_masks=np.array(TMASKS)[1:]
print(test_images.shape,test_masks.shape)
```

```
(30, 512, 512, 3) (30, 512, 512, 1)
(10, 512, 512, 3) (10, 512, 512, 1)
```

In [17]:
```
images_train, images_test, masks_train, masks_test = train_test_split(images, masks
print(len(images_train), len(masks_train))
```

```
24 24
```

In [18]:
```
def conv_block(input, num_filters):
    conv = tf.keras.layers.Conv2D(num_filters, 3, padding="same")(input)
    conv = tf.keras.layers.BatchNormalization()(conv)
    conv = tf.keras.layers.Activation("relu")(conv)
    conv = tf.keras.layers.Conv2D(num_filters, 3, padding="same")(conv)
    conv = tf.keras.layers.BatchNormalization()(conv)
    conv = tf.keras.layers.Activation("relu")(conv)
    return conv

def encoder_block(input, num_filters):
    skip = conv_block(input, num_filters)
    pool = tf.keras.layers.MaxPool2D((2,2))(skip)
    return skip, pool

def decoder_block(input, skip, num_filters):
    up_conv = tf.keras.layers.Conv2DTranspose(num_filters, (2,2), strides=2, paddir
    #conv = tf.keras.layers.Concatenate()([up_conv, skip])
    conv = conv_block(up_conv, num_filters)
    return conv

def Unet(input_shape):
    inputs = tf.keras.layers.Input(input_shape)

    skip1, pool1 = encoder_block(inputs, 64)
    skip2, pool2 = encoder_block(pool1, 128)
    skip3, pool3 = encoder_block(pool2, 256)
    skip4, pool4 = encoder_block(pool3, 512)

    bridge = conv_block(pool4, 1024)

    decode1 = decoder_block(bridge, skip4, 512)
    decode2 = decoder_block(decode1, skip3, 256)
    decode3 = decoder_block(decode2, skip2, 128)
    decode4 = decoder_block(decode3, skip1, 64)
    outputs = tf.keras.layers.Conv2D(1, 1, padding="same", activation="sigmoid")(de
    model = tf.keras.models.Model(inputs, outputs, name="U-Net")
    return model
```

```
unet_model = Unet((512,512,3))
unet_model.compile(optimizer=Adam(learning_rate=0.00001), loss= 'binary_crossentrop
unet_model.summary()
```

```
Model: "U-Net"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 512, 512, 3)]     0

 conv2d (Conv2D)             (None, 512, 512, 64)      1792

 batch_normalization (Batch  (None, 512, 512, 64)      256
 Normalization)

 activation (Activation)     (None, 512, 512, 64)      0

 conv2d_1 (Conv2D)           (None, 512, 512, 64)      36928

 batch_normalization_1 (Bat  (None, 512, 512, 64)      256
 chNormalization)

 activation_1 (Activation)   (None, 512, 512, 64)      0

 max_pooling2d (MaxPooling2  (None, 256, 256, 64)      0
 D)

 conv2d_2 (Conv2D)           (None, 256, 256, 128)     73856

 batch_normalization_2 (Bat  (None, 256, 256, 128)     512
 chNormalization)

 activation_2 (Activation)   (None, 256, 256, 128)     0

 conv2d_3 (Conv2D)           (None, 256, 256, 128)     147584

 batch_normalization_3 (Bat  (None, 256, 256, 128)     512
 chNormalization)

 activation_3 (Activation)   (None, 256, 256, 128)     0

 max_pooling2d_1 (MaxPoolin  (None, 128, 128, 128)     0
 g2D)

 conv2d_4 (Conv2D)           (None, 128, 128, 256)     295168

 batch_normalization_4 (Bat  (None, 128, 128, 256)     1024
 chNormalization)

 activation_4 (Activation)   (None, 128, 128, 256)     0

 conv2d_5 (Conv2D)           (None, 128, 128, 256)     590080

 batch_normalization_5 (Bat  (None, 128, 128, 256)     1024
 chNormalization)

 activation_5 (Activation)   (None, 128, 128, 256)     0

 max_pooling2d_2 (MaxPoolin  (None, 64, 64, 256)       0
 g2D)

 conv2d_6 (Conv2D)           (None, 64, 64, 512)       1180160

 batch_normalization_6 (Bat  (None, 64, 64, 512)       2048
 chNormalization)

 activation_6 (Activation)   (None, 64, 64, 512)       0
```

```
conv2d_7 (Conv2D)           (None, 64, 64, 512)       2359808

batch_normalization_7 (Bat  (None, 64, 64, 512)       2048
chNormalization)

activation_7 (Activation)   (None, 64, 64, 512)       0

max_pooling2d_3 (MaxPoolin  (None, 32, 32, 512)       0
g2D)

conv2d_8 (Conv2D)           (None, 32, 32, 1024)      4719616

batch_normalization_8 (Bat  (None, 32, 32, 1024)      4096
chNormalization)

activation_8 (Activation)   (None, 32, 32, 1024)      0

conv2d_9 (Conv2D)           (None, 32, 32, 1024)      9438208

batch_normalization_9 (Bat  (None, 32, 32, 1024)      4096
chNormalization)

activation_9 (Activation)   (None, 32, 32, 1024)      0

conv2d_transpose (Conv2DTr  (None, 64, 64, 512)       2097664
anspose)

conv2d_10 (Conv2D)          (None, 64, 64, 512)       2359808

batch_normalization_10 (Ba  (None, 64, 64, 512)       2048
tchNormalization)

activation_10 (Activation)  (None, 64, 64, 512)       0

conv2d_11 (Conv2D)          (None, 64, 64, 512)       2359808

batch_normalization_11 (Ba  (None, 64, 64, 512)       2048
tchNormalization)

activation_11 (Activation)  (None, 64, 64, 512)       0

conv2d_transpose_1 (Conv2D  (None, 128, 128, 256)     524544
Transpose)

conv2d_12 (Conv2D)          (None, 128, 128, 256)     590080

batch_normalization_12 (Ba  (None, 128, 128, 256)     1024
tchNormalization)

activation_12 (Activation)  (None, 128, 128, 256)     0

conv2d_13 (Conv2D)          (None, 128, 128, 256)     590080

batch_normalization_13 (Ba  (None, 128, 128, 256)     1024
tchNormalization)

activation_13 (Activation)  (None, 128, 128, 256)     0

conv2d_transpose_2 (Conv2D  (None, 256, 256, 128)     131200
Transpose)

conv2d_14 (Conv2D)          (None, 256, 256, 128)     147584

batch_normalization_14 (Ba  (None, 256, 256, 128)     512
```

```
tchNormalization)

activation_14 (Activation)  (None, 256, 256, 128)     0

conv2d_15 (Conv2D)          (None, 256, 256, 128)     147584

batch_normalization_15 (Ba  (None, 256, 256, 128)     512
tchNormalization)

activation_15 (Activation)  (None, 256, 256, 128)     0

conv2d_transpose_3 (Conv2D  (None, 512, 512, 64)      32832
Transpose)

conv2d_16 (Conv2D)          (None, 512, 512, 64)      36928

batch_normalization_16 (Ba  (None, 512, 512, 64)      256
tchNormalization)

activation_16 (Activation)  (None, 512, 512, 64)      0

conv2d_17 (Conv2D)          (None, 512, 512, 64)      36928

batch_normalization_17 (Ba  (None, 512, 512, 64)      256
tchNormalization)

activation_17 (Activation)  (None, 512, 512, 64)      0

conv2d_18 (Conv2D)          (None, 512, 512, 1)       65

=================================================================
Total params: 27921857 (106.51 MB)
Trainable params: 27910081 (106.47 MB)
Non-trainable params: 11776 (46.00 KB)
_________________________________________________________________
```

In [19]:

```
unet_result = unet_model.fit(
    images_train, masks_train,
    validation_data =(images_test, masks_test), batch_size = 4, epochs =100)
```

```
Epoch 1/100
6/6 [==============================] - 44s 2s/step - loss: -13.8924 - accuracy: 0.
3227 - val_loss: 1.2216 - val_accuracy: 0.8326
Epoch 2/100
6/6 [==============================] - 5s 828ms/step - loss: -20.0545 - accuracy:
0.3986 - val_loss: 1.8043 - val_accuracy: 0.8326
Epoch 3/100
6/6 [==============================] - 5s 829ms/step - loss: -25.8550 - accuracy:
0.4874 - val_loss: 2.4450 - val_accuracy: 0.8326
Epoch 4/100
6/6 [==============================] - 5s 886ms/step - loss: -29.7150 - accuracy:
0.5355 - val_loss: 3.1688 - val_accuracy: 0.8326
Epoch 5/100
6/6 [==============================] - 5s 840ms/step - loss: -36.7751 - accuracy:
0.5522 - val_loss: 4.0342 - val_accuracy: 0.8326
Epoch 6/100
6/6 [==============================] - 5s 889ms/step - loss: -42.5181 - accuracy:
0.5541 - val_loss: 5.0341 - val_accuracy: 0.8326
Epoch 7/100
6/6 [==============================] - 5s 896ms/step - loss: -45.8048 - accuracy:
0.5450 - val_loss: 6.1635 - val_accuracy: 0.8326
Epoch 8/100
6/6 [==============================] - 5s 899ms/step - loss: -52.3652 - accuracy:
0.5413 - val_loss: 7.4298 - val_accuracy: 0.8326
Epoch 9/100
6/6 [==============================] - 5s 856ms/step - loss: -57.4572 - accuracy:
0.5260 - val_loss: 8.8393 - val_accuracy: 0.8326
Epoch 10/100
6/6 [==============================] - 5s 851ms/step - loss: -62.4201 - accuracy:
0.5168 - val_loss: 10.3772 - val_accuracy: 0.8326
Epoch 11/100
6/6 [==============================] - 5s 853ms/step - loss: -70.2536 - accuracy:
0.5326 - val_loss: 12.0124 - val_accuracy: 0.8326
Epoch 12/100
6/6 [==============================] - 5s 904ms/step - loss: -73.8173 - accuracy:
0.5403 - val_loss: 13.7184 - val_accuracy: 0.8326
Epoch 13/100
6/6 [==============================] - 5s 862ms/step - loss: -84.0974 - accuracy:
0.5427 - val_loss: 15.5522 - val_accuracy: 0.8326
Epoch 14/100
6/6 [==============================] - 5s 914ms/step - loss: -89.5203 - accuracy:
0.5570 - val_loss: 17.4730 - val_accuracy: 0.8326
Epoch 15/100
6/6 [==============================] - 5s 912ms/step - loss: -96.1745 - accuracy:
0.5544 - val_loss: 19.6302 - val_accuracy: 0.8326
Epoch 16/100
6/6 [==============================] - 5s 872ms/step - loss: -103.0396 - accuracy:
0.5516 - val_loss: 21.9948 - val_accuracy: 0.8326
Epoch 17/100
6/6 [==============================] - 5s 920ms/step - loss: -106.8237 - accuracy:
0.5492 - val_loss: 24.6299 - val_accuracy: 0.8326
Epoch 18/100
6/6 [==============================] - 5s 922ms/step - loss: -115.2326 - accuracy:
0.5545 - val_loss: 27.5040 - val_accuracy: 0.8326
Epoch 19/100
6/6 [==============================] - 5s 883ms/step - loss: -119.6371 - accuracy:
0.5604 - val_loss: 30.6850 - val_accuracy: 0.8326
Epoch 20/100
6/6 [==============================] - 5s 924ms/step - loss: -123.9795 - accuracy:
0.5734 - val_loss: 34.0274 - val_accuracy: 0.8326
Epoch 21/100
6/6 [==============================] - 5s 933ms/step - loss: -130.4419 - accuracy:
0.5688 - val_loss: 37.6628 - val_accuracy: 0.8326
Epoch 22/100
```

```
6/6 [==============================] - 5s 930ms/step - loss: -135.5816 - accuracy:
0.5748 - val_loss: 41.4446 - val_accuracy: 0.8326
Epoch 23/100
6/6 [==============================] - 5s 936ms/step - loss: -139.1033 - accuracy:
0.5800 - val_loss: 45.4672 - val_accuracy: 0.8326
Epoch 24/100
6/6 [==============================] - 5s 938ms/step - loss: -140.5593 - accuracy:
0.5818 - val_loss: 49.5135 - val_accuracy: 0.8326
Epoch 25/100
6/6 [==============================] - 5s 897ms/step - loss: -147.2651 - accuracy:
0.5862 - val_loss: 53.4432 - val_accuracy: 0.8326
Epoch 26/100
6/6 [==============================] - 5s 939ms/step - loss: -150.3007 - accuracy:
0.5923 - val_loss: 57.2206 - val_accuracy: 0.8326
Epoch 27/100
6/6 [==============================] - 5s 940ms/step - loss: -157.8978 - accuracy:
0.5938 - val_loss: 61.0101 - val_accuracy: 0.8326
Epoch 28/100
6/6 [==============================] - 5s 902ms/step - loss: -158.9753 - accuracy:
0.5931 - val_loss: 64.6344 - val_accuracy: 0.8326
Epoch 29/100
6/6 [==============================] - 5s 897ms/step - loss: -166.8380 - accuracy:
0.5914 - val_loss: 68.4763 - val_accuracy: 0.8326
Epoch 30/100
6/6 [==============================] - 5s 897ms/step - loss: -168.4730 - accuracy:
0.6001 - val_loss: 72.1078 - val_accuracy: 0.8326
Epoch 31/100
6/6 [==============================] - 5s 896ms/step - loss: -170.8943 - accuracy:
0.6018 - val_loss: 75.7472 - val_accuracy: 0.8326
Epoch 32/100
6/6 [==============================] - 5s 899ms/step - loss: -173.0132 - accuracy:
0.6043 - val_loss: 79.2851 - val_accuracy: 0.8326
Epoch 33/100
6/6 [==============================] - 5s 903ms/step - loss: -175.5124 - accuracy:
0.6117 - val_loss: 82.8097 - val_accuracy: 0.8326
Epoch 34/100
6/6 [==============================] - 5s 897ms/step - loss: -175.1844 - accuracy:
0.6171 - val_loss: 86.7792 - val_accuracy: 0.8326
Epoch 35/100
6/6 [==============================] - 5s 900ms/step - loss: -185.6065 - accuracy:
0.6164 - val_loss: 90.2746 - val_accuracy: 0.8326
Epoch 36/100
6/6 [==============================] - 5s 905ms/step - loss: -187.0884 - accuracy:
0.6146 - val_loss: 94.2269 - val_accuracy: 0.8326
Epoch 37/100
6/6 [==============================] - 6s 946ms/step - loss: -188.4901 - accuracy:
0.6204 - val_loss: 96.8935 - val_accuracy: 0.8326
Epoch 38/100
6/6 [==============================] - 5s 914ms/step - loss: -197.0610 - accuracy:
0.6236 - val_loss: 100.3771 - val_accuracy: 0.8326
Epoch 39/100
6/6 [==============================] - 5s 904ms/step - loss: -198.5569 - accuracy:
0.6208 - val_loss: 102.7038 - val_accuracy: 0.8326
Epoch 40/100
6/6 [==============================] - 5s 912ms/step - loss: -203.3491 - accuracy:
0.6265 - val_loss: 105.7221 - val_accuracy: 0.8326
Epoch 41/100
6/6 [==============================] - 5s 910ms/step - loss: -202.2186 - accuracy:
0.6246 - val_loss: 107.0068 - val_accuracy: 0.8326
Epoch 42/100
6/6 [==============================] - 6s 951ms/step - loss: -210.2491 - accuracy:
0.6309 - val_loss: 109.9855 - val_accuracy: 0.8326
Epoch 43/100
6/6 [==============================] - 5s 918ms/step - loss: -208.9252 - accuracy:
```

```
0.6272 - val_loss: 111.5998 - val_accuracy: 0.8326
Epoch 44/100
6/6 [==============================] - 5s 912ms/step - loss: -209.8917 - accuracy:
0.6325 - val_loss: 113.9667 - val_accuracy: 0.8326
Epoch 45/100
6/6 [==============================] - 6s 956ms/step - loss: -219.4907 - accuracy:
0.6351 - val_loss: 113.5021 - val_accuracy: 0.8326
Epoch 46/100
6/6 [==============================] - 5s 919ms/step - loss: -222.6605 - accuracy:
0.6375 - val_loss: 115.3215 - val_accuracy: 0.8326
Epoch 47/100
6/6 [==============================] - 5s 917ms/step - loss: -217.2791 - accuracy:
0.6402 - val_loss: 116.9439 - val_accuracy: 0.8326
Epoch 48/100
6/6 [==============================] - 6s 957ms/step - loss: -226.4774 - accuracy:
0.6466 - val_loss: 116.2885 - val_accuracy: 0.8326
Epoch 49/100
6/6 [==============================] - 5s 918ms/step - loss: -230.4056 - accuracy:
0.6489 - val_loss: 118.8893 - val_accuracy: 0.8326
Epoch 50/100
6/6 [==============================] - 5s 919ms/step - loss: -232.3974 - accuracy:
0.6487 - val_loss: 113.8934 - val_accuracy: 0.8326
Epoch 51/100
6/6 [==============================] - 5s 921ms/step - loss: -228.8853 - accuracy:
0.6486 - val_loss: 116.7093 - val_accuracy: 0.8326
Epoch 52/100
6/6 [==============================] - 6s 960ms/step - loss: -236.8292 - accuracy:
0.6548 - val_loss: 110.2105 - val_accuracy: 0.8326
Epoch 53/100
6/6 [==============================] - 6s 961ms/step - loss: -236.5115 - accuracy:
0.6568 - val_loss: 116.7721 - val_accuracy: 0.8326
Epoch 54/100
6/6 [==============================] - 5s 925ms/step - loss: -244.2721 - accuracy:
0.6672 - val_loss: 107.8501 - val_accuracy: 0.8326
Epoch 55/100
6/6 [==============================] - 5s 925ms/step - loss: -245.8602 - accuracy:
0.6611 - val_loss: 110.8789 - val_accuracy: 0.8326
Epoch 56/100
6/6 [==============================] - 5s 917ms/step - loss: -249.8947 - accuracy:
0.6657 - val_loss: 100.7762 - val_accuracy: 0.8326
Epoch 57/100
6/6 [==============================] - 6s 966ms/step - loss: -253.4826 - accuracy:
0.6716 - val_loss: 110.1303 - val_accuracy: 0.8326
Epoch 58/100
6/6 [==============================] - 5s 924ms/step - loss: -252.2966 - accuracy:
0.6676 - val_loss: 95.3131 - val_accuracy: 0.8326
Epoch 59/100
6/6 [==============================] - 5s 924ms/step - loss: -255.0289 - accuracy:
0.6714 - val_loss: 103.3075 - val_accuracy: 0.8326
Epoch 60/100
6/6 [==============================] - 6s 965ms/step - loss: -259.0405 - accuracy:
0.6767 - val_loss: 92.7550 - val_accuracy: 0.8325
Epoch 61/100
6/6 [==============================] - 5s 929ms/step - loss: -259.0504 - accuracy:
0.6805 - val_loss: 100.3304 - val_accuracy: 0.8326
Epoch 62/100
6/6 [==============================] - 5s 924ms/step - loss: -255.6469 - accuracy:
0.6773 - val_loss: 91.1659 - val_accuracy: 0.8324
Epoch 63/100
6/6 [==============================] - 6s 964ms/step - loss: -262.4825 - accuracy:
0.6788 - val_loss: 83.7689 - val_accuracy: 0.8321
Epoch 64/100
6/6 [==============================] - 6s 967ms/step - loss: -253.5939 - accuracy:
0.6785 - val_loss: 87.8610 - val_accuracy: 0.8323
```

```
Epoch 65/100
6/6 [==============================] - 6s 970ms/step - loss: -262.2189 - accuracy:
0.6856 - val_loss: 75.9429 - val_accuracy: 0.8314
Epoch 66/100
6/6 [==============================] - 6s 969ms/step - loss: -263.0437 - accuracy:
0.6867 - val_loss: 72.1442 - val_accuracy: 0.8307
Epoch 67/100
6/6 [==============================] - 5s 930ms/step - loss: -265.7524 - accuracy:
0.6878 - val_loss: 73.8360 - val_accuracy: 0.8310
Epoch 68/100
6/6 [==============================] - 5s 928ms/step - loss: -272.2414 - accuracy:
0.6930 - val_loss: 64.8975 - val_accuracy: 0.8286
Epoch 69/100
6/6 [==============================] - 6s 965ms/step - loss: -272.7758 - accuracy:
0.6910 - val_loss: 63.2837 - val_accuracy: 0.8277
Epoch 70/100
6/6 [==============================] - 5s 934ms/step - loss: -277.1022 - accuracy:
0.6917 - val_loss: 57.7673 - val_accuracy: 0.8254
Epoch 71/100
6/6 [==============================] - 6s 967ms/step - loss: -276.7738 - accuracy:
0.6934 - val_loss: 57.3620 - val_accuracy: 0.8256
Epoch 72/100
6/6 [==============================] - 5s 931ms/step - loss: -276.8028 - accuracy:
0.6941 - val_loss: 53.9750 - val_accuracy: 0.8235
Epoch 73/100
6/6 [==============================] - 5s 930ms/step - loss: -280.7566 - accuracy:
0.6996 - val_loss: 56.2073 - val_accuracy: 0.8245
Epoch 74/100
6/6 [==============================] - 5s 928ms/step - loss: -279.9681 - accuracy:
0.6954 - val_loss: 39.0074 - val_accuracy: 0.8096
Epoch 75/100
6/6 [==============================] - 6s 933ms/step - loss: -272.2032 - accuracy:
0.6942 - val_loss: 50.3600 - val_accuracy: 0.8210
Epoch 76/100
6/6 [==============================] - 5s 927ms/step - loss: -282.6217 - accuracy:
0.6964 - val_loss: 29.8371 - val_accuracy: 0.8009
Epoch 77/100
6/6 [==============================] - 6s 971ms/step - loss: -287.0490 - accuracy:
0.7016 - val_loss: 36.3161 - val_accuracy: 0.8087
Epoch 78/100
6/6 [==============================] - 5s 936ms/step - loss: -288.7666 - accuracy:
0.7017 - val_loss: 21.1056 - val_accuracy: 0.7885
Epoch 79/100
6/6 [==============================] - 5s 930ms/step - loss: -284.9572 - accuracy:
0.7005 - val_loss: 37.7647 - val_accuracy: 0.8102
Epoch 80/100
6/6 [==============================] - 6s 966ms/step - loss: -289.1024 - accuracy:
0.7024 - val_loss: 10.3052 - val_accuracy: 0.7729
Epoch 81/100
6/6 [==============================] - 6s 972ms/step - loss: -286.7587 - accuracy:
0.7034 - val_loss: 21.9083 - val_accuracy: 0.7920
Epoch 82/100
6/6 [==============================] - 6s 972ms/step - loss: -294.4076 - accuracy:
0.7055 - val_loss: 4.8782 - val_accuracy: 0.7650
Epoch 83/100
6/6 [==============================] - 5s 935ms/step - loss: -290.1474 - accuracy:
0.7056 - val_loss: 19.0518 - val_accuracy: 0.7892
Epoch 84/100
6/6 [==============================] - 6s 971ms/step - loss: -295.1857 - accuracy:
0.7086 - val_loss: 2.7368 - val_accuracy: 0.7639
Epoch 85/100
6/6 [==============================] - 6s 971ms/step - loss: -293.9391 - accuracy:
0.7095 - val_loss: 5.5420 - val_accuracy: 0.7676
Epoch 86/100
```

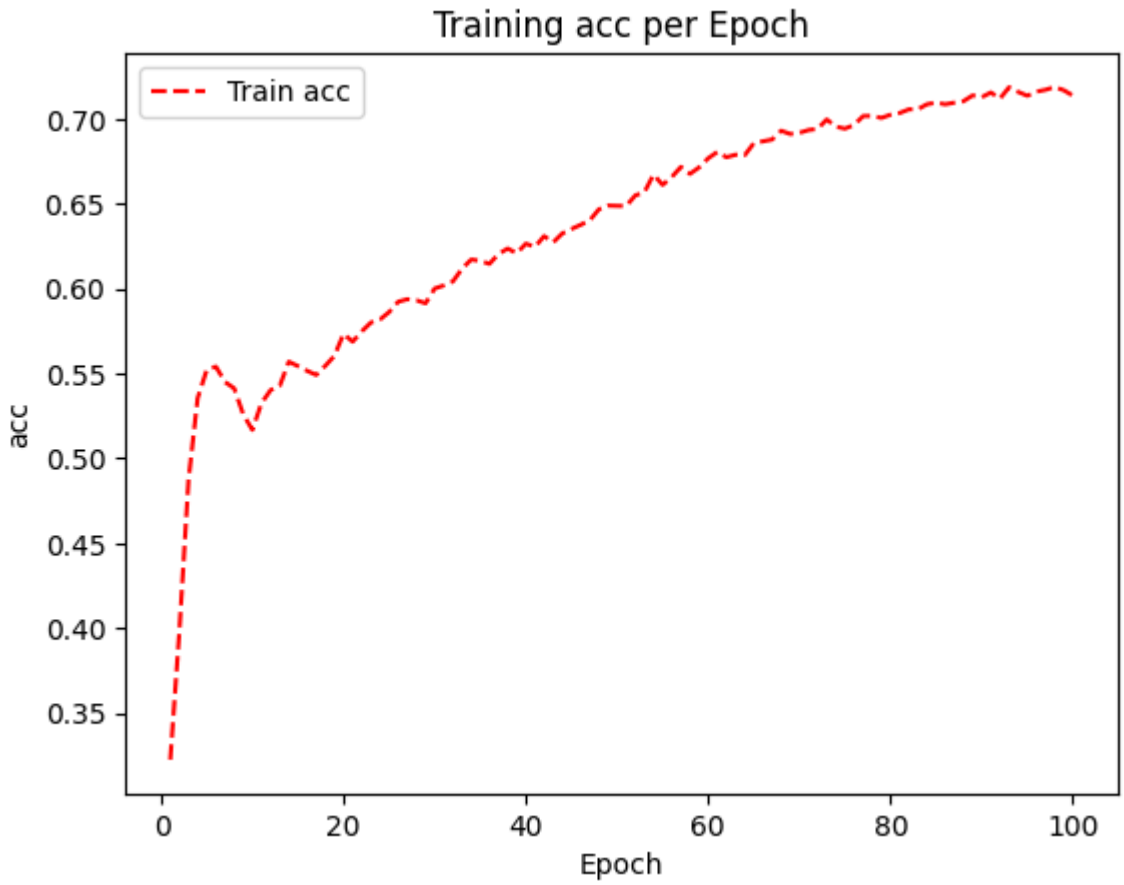
```
6/6 [==============================] - 6s 978ms/step - loss: -296.8681 - accuracy:
0.7085 - val_loss: -2.3604 - val_accuracy: 0.7539
Epoch 87/100
6/6 [==============================] - 6s 974ms/step - loss: -299.7935 - accuracy:
0.7095 - val_loss: 1.4498 - val_accuracy: 0.7628
Epoch 88/100
6/6 [==============================] - 5s 932ms/step - loss: -295.8925 - accuracy:
0.7104 - val_loss: -5.0261 - val_accuracy: 0.7517
Epoch 89/100
6/6 [==============================] - 5s 929ms/step - loss: -301.1413 - accuracy:
0.7137 - val_loss: -9.5664 - val_accuracy: 0.7481
Epoch 90/100
6/6 [==============================] - 6s 969ms/step - loss: -296.4143 - accuracy:
0.7127 - val_loss: 5.1068 - val_accuracy: 0.7710
Epoch 91/100
6/6 [==============================] - 6s 974ms/step - loss: -302.6319 - accuracy:
0.7155 - val_loss: -18.5112 - val_accuracy: 0.7292
Epoch 92/100
6/6 [==============================] - 6s 971ms/step - loss: -299.5598 - accuracy:
0.7118 - val_loss: -16.0748 - val_accuracy: 0.7350
Epoch 93/100
6/6 [==============================] - 6s 978ms/step - loss: -302.9877 - accuracy:
0.7186 - val_loss: -23.6722 - val_accuracy: 0.7267
Epoch 94/100
6/6 [==============================] - 5s 934ms/step - loss: -301.9479 - accuracy:
0.7158 - val_loss: 2.5311 - val_accuracy: 0.7686
Epoch 95/100
6/6 [==============================] - 6s 968ms/step - loss: -300.7713 - accuracy:
0.7135 - val_loss: -26.0557 - val_accuracy: 0.7197
Epoch 96/100
6/6 [==============================] - 5s 934ms/step - loss: -300.9677 - accuracy:
0.7160 - val_loss: -2.6085 - val_accuracy: 0.7578
Epoch 97/100
6/6 [==============================] - 5s 931ms/step - loss: -305.5011 - accuracy:
0.7171 - val_loss: -49.8669 - val_accuracy: 0.6791
Epoch 98/100
6/6 [==============================] - 6s 933ms/step - loss: -303.2489 - accuracy:
0.7187 - val_loss: -15.1535 - val_accuracy: 0.7407
Epoch 99/100
6/6 [==============================] - 5s 931ms/step - loss: -305.6929 - accuracy:
0.7168 - val_loss: -38.8938 - val_accuracy: 0.6973
Epoch 100/100
6/6 [==============================] - 6s 934ms/step - loss: -302.0756 - accuracy:
0.7136 - val_loss: -21.1894 - val_accuracy: 0.7312
```

In [20]:

```
print(unet_result.history.keys())
import matplotlib.pyplot as plt
f, ax = plt.subplots()
ax.plot([None] + unet_result.history['accuracy'], 'r--')
# Plot legend and use the best location automatically: loc = 0.
ax.legend(['Train acc'], loc = 0)
ax.set_title('Training acc per Epoch')
ax.set_xlabel('Epoch')
ax.set_ylabel('acc')
```

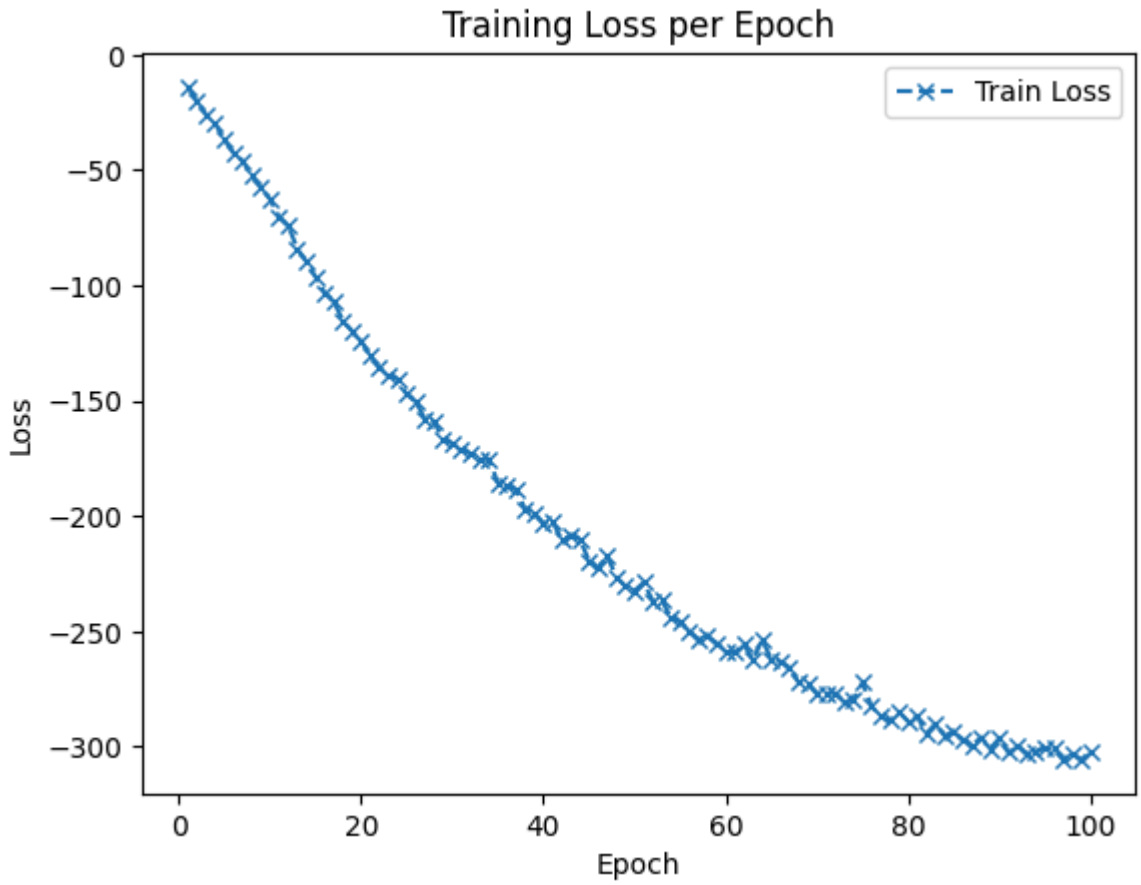
Out[20]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
Text(0, 0.5, 'acc')
```



In [21]:
```
f, ax = plt.subplots()
ax.plot([None] + unet_result.history['loss'], 'x--')
# Plot legend and use the best location automatically: loc = 0.
ax.legend(['Train Loss'], loc = 0)
ax.set_title('Training Loss per Epoch')
ax.set_xlabel('Epoch')
ax.set_ylabel('Loss')
```

Out[21]:
```
Text(0, 0.5, 'Loss')
```



In [32]:
```
def show_result(idx, og, unet, target, p):

    fig, axs = plt.subplots(1, 3, figsize=(12,12))
    axs[0].set_title("Original "+str(idx) )
    axs[0].imshow(og)
    axs[0].axis('off')

    axs[1].set_title("segnet: p>"+str(p))
    axs[1].imshow(unet)
    axs[1].axis('off')

    axs[2].set_title("Ground Truth")
    axs[2].imshow(target)
    axs[2].axis('off')

    plt.show()
```

In [33]:
```
unet_predict = unet_model.predict(images_test)
```

```
1/1 [==============================] - 0s 22ms/step
```

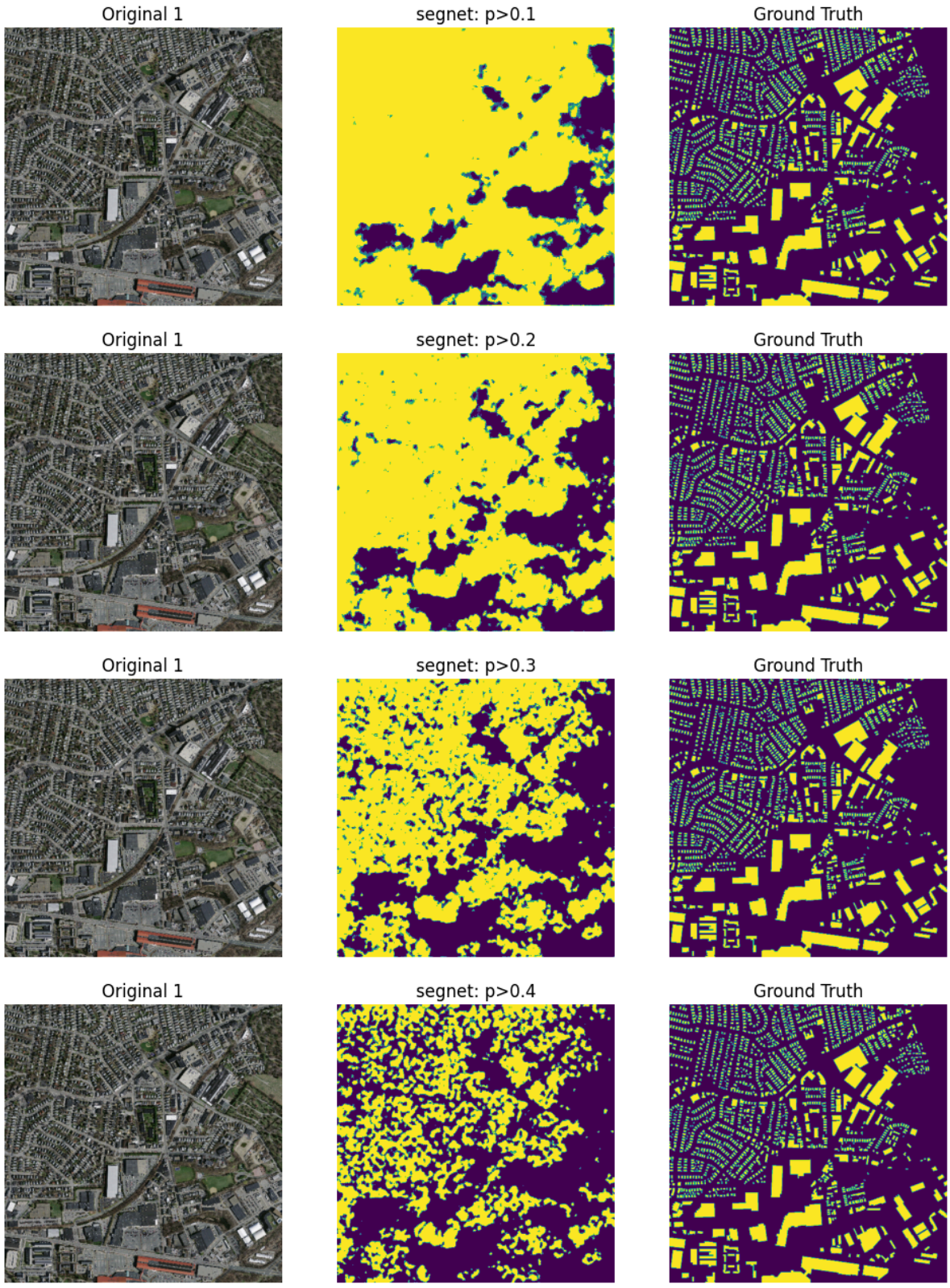
In [34]:
```
len(images_test)
```

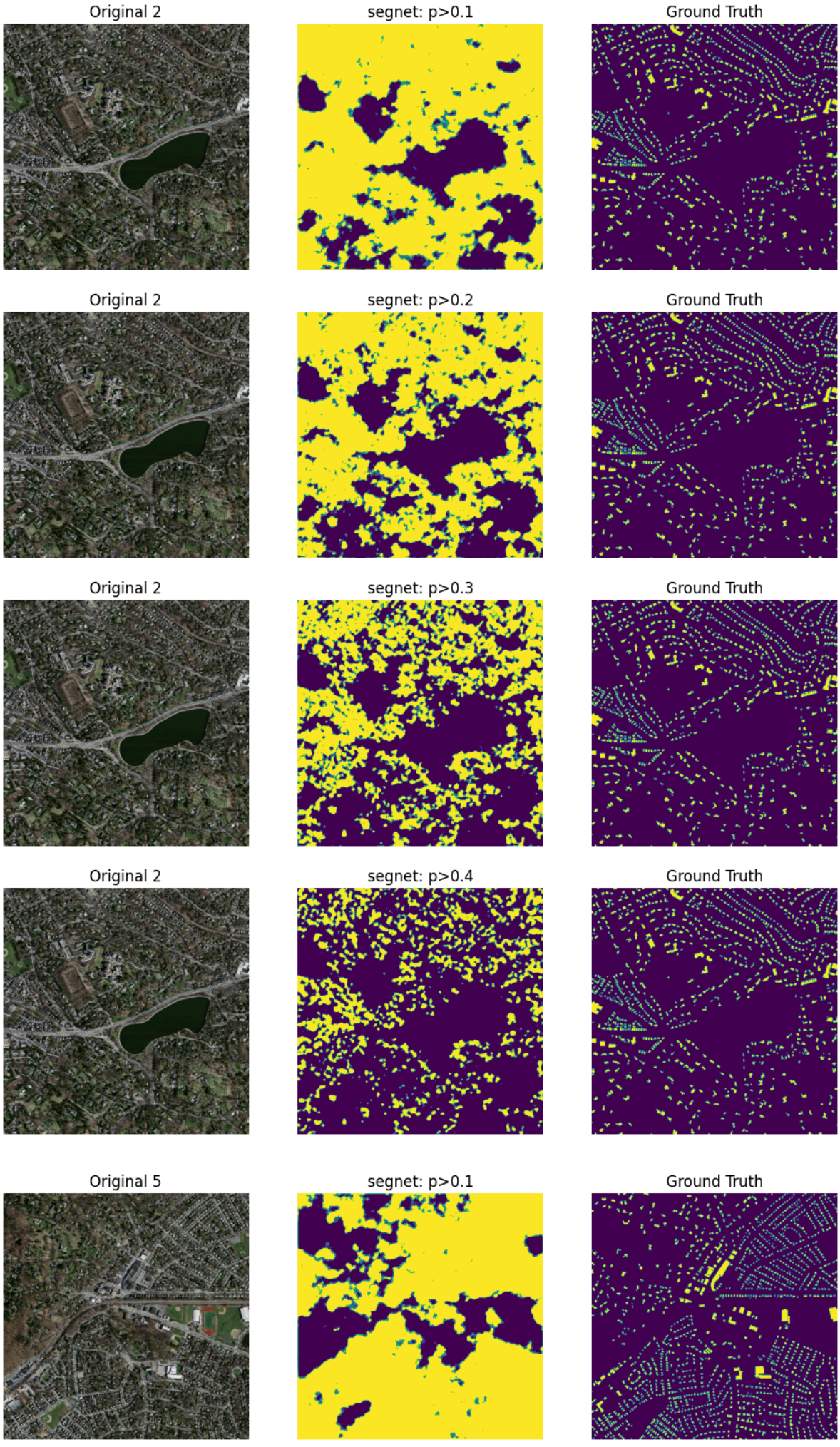
Out[34]:
```
6
```

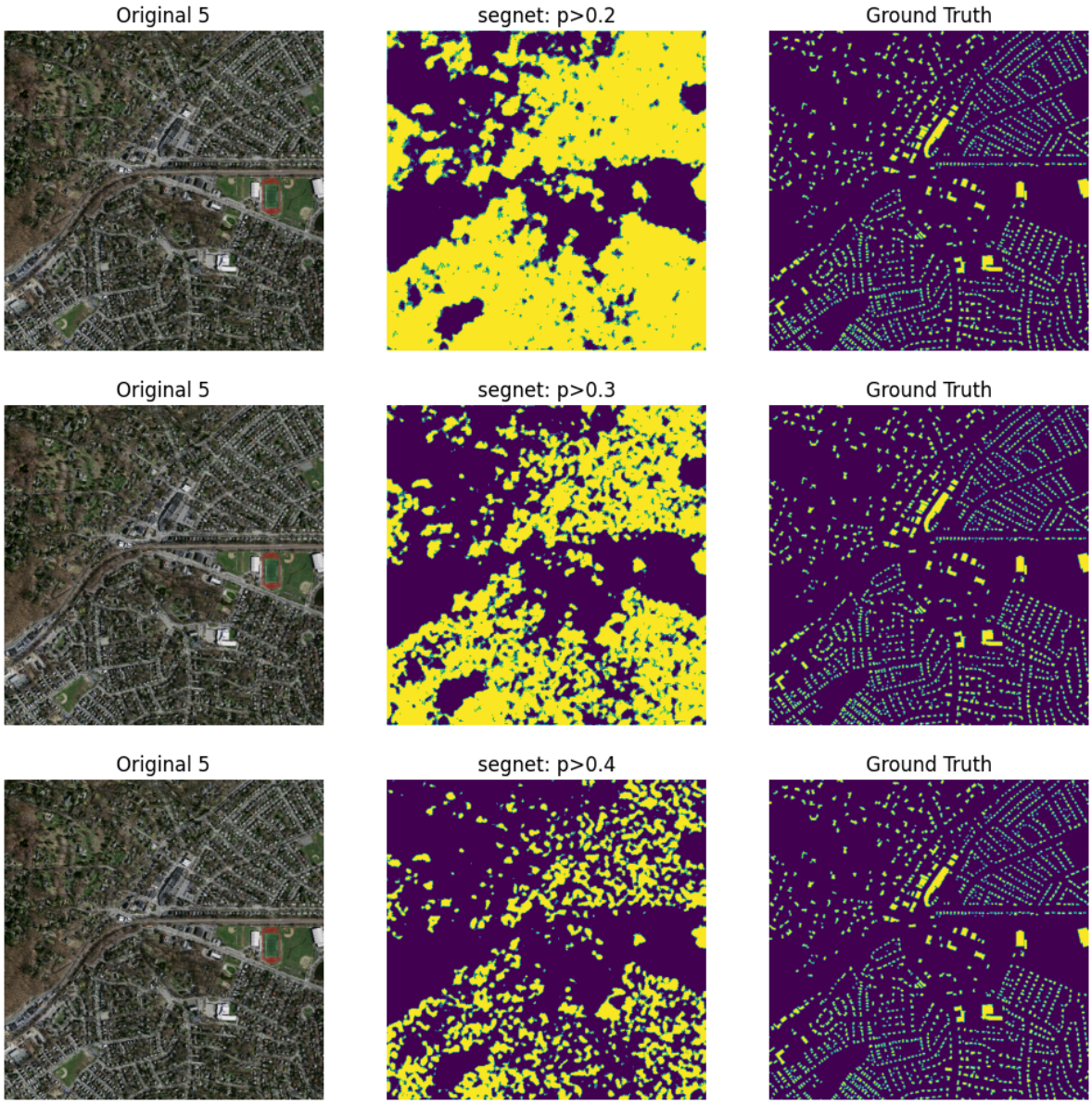
In [35]:
```
segnet_predict1 = (unet_predict > 0.1).astype(np.uint8)
segnet_predict2 = (unet_predict > 0.2).astype(np.uint8)
segnet_predict3 = (unet_predict > 0.3).astype(np.uint8)
segnet_predict4 = (unet_predict > 0.4).astype(np.uint8)
```

In [36]:
```
show_test_idx = random.sample(range(len(unet_predict)), 3)
for idx in show_test_idx:
    show_result(idx, images_test[idx], segnet_predict1[idx], masks_test[idx], 0.1)
    show_result(idx, images_test[idx], segnet_predict2[idx], masks_test[idx], 0.2)
```

```
    show_result(idx, images_test[idx], segnet_predict3[idx], masks_test[idx], 0.3)
    show_result(idx, images_test[idx], segnet_predict4[idx], masks_test[idx], 0.4)
    print()
```




Original 2
segnet: p>0.1
Ground Truth
Original 2
segnet: p>0.2
Ground Truth
Original 2
segnet: p>0.3
Ground Truth
Original 2
segnet: p>0.4
Ground Truth
Original 5
segnet: p>0.1
Ground Truth



In [37]:
```
unet_predictT = unet_model.predict(test_images)
```

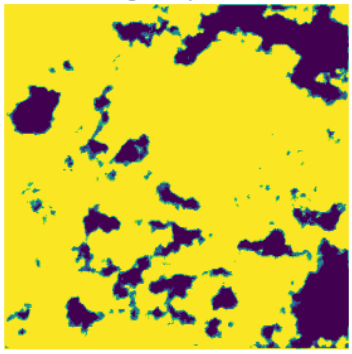
```
1/1 [==============================] - 0s 28ms/step
```

In [38]:
```
unet_predict1 = (unet_predictT > 0.1).astype(np.uint8)
unet_predict2 = (unet_predictT > 0.2).astype(np.uint8)
unet_predict3 = (unet_predictT > 0.3).astype(np.uint8)
unet_predict4 = (unet_predictT > 0.4).astype(np.uint8)
```

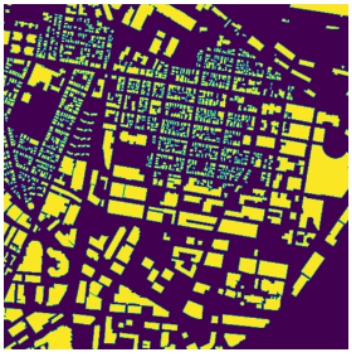
In [39]:
```
show_test_idx = random.sample(range(len(unet_predictT)), 3)
for idx in show_test_idx:
    show_result(idx, test_images[idx], unet_predict1[idx], test_masks[idx], 0.1)
    show_result(idx, test_images[idx], unet_predict2[idx], test_masks[idx], 0.2)
    show_result(idx, test_images[idx], unet_predict3[idx], test_masks[idx], 0.3)
    show_result(idx, test_images[idx], unet_predict4[idx], test_masks[idx], 0.4)
    print()
```

Original 3

segnet: p>0.1

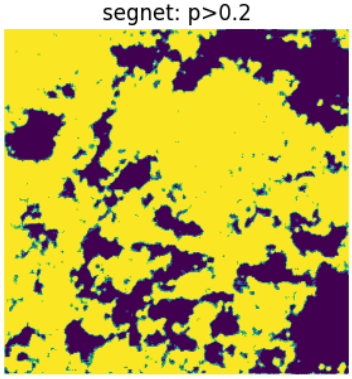


Ground Truth

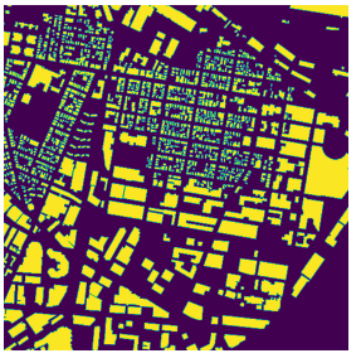


Original 3

segnet: p>0.2

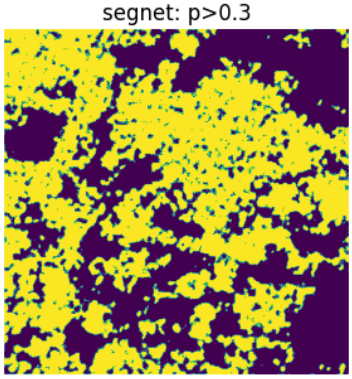


Ground Truth

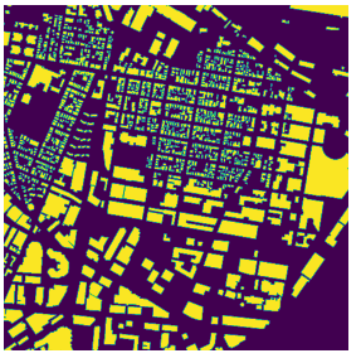


Original 3

segnet: p>0.3

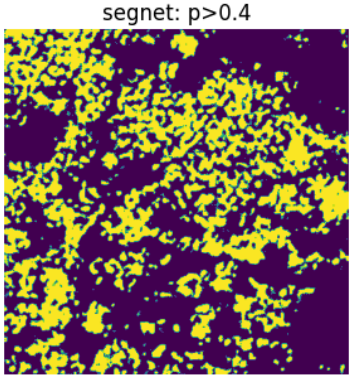


Ground Truth

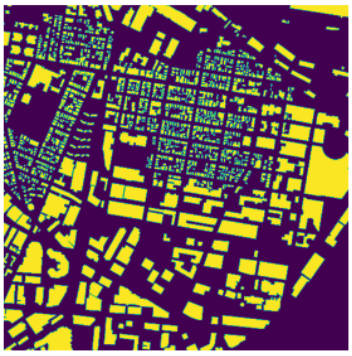


Original 3

segnet: p>0.4

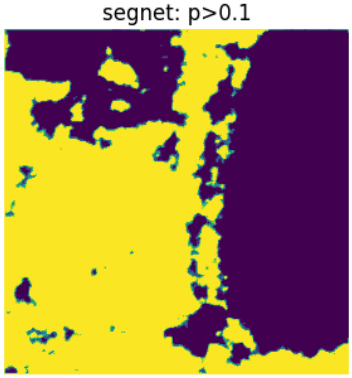


Ground Truth

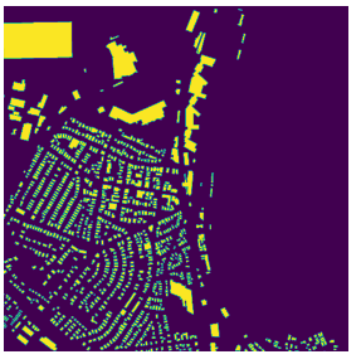


Original 9

segnet: p>0.1

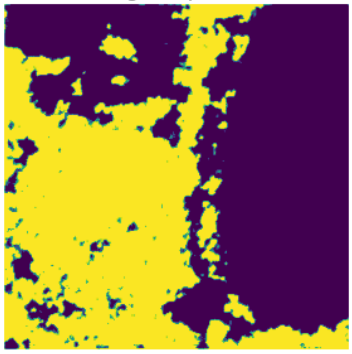


Ground Truth

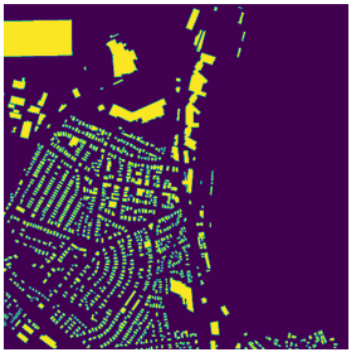


Original 9

segnet: p>0.2

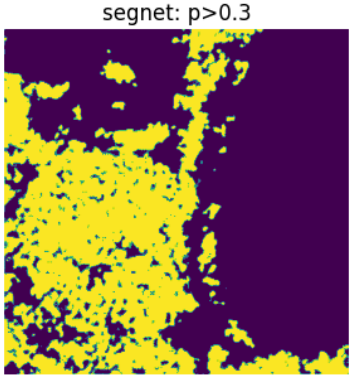


Ground Truth

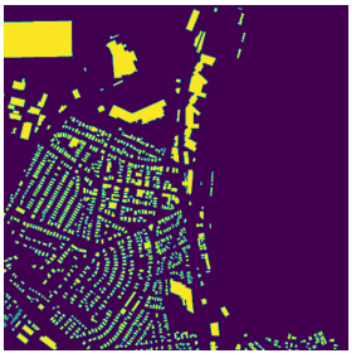


Original 9

segnet: p>0.3

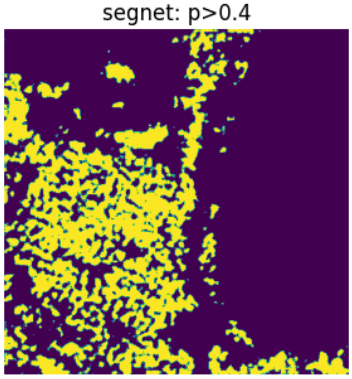


Ground Truth

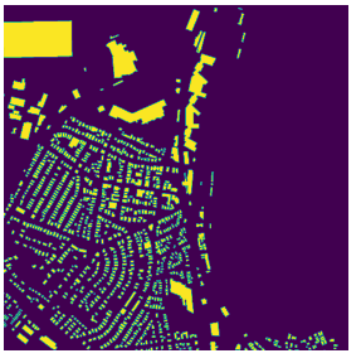


Original 9

segnet: p>0.4

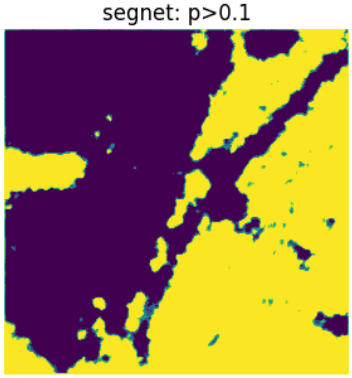


Ground Truth

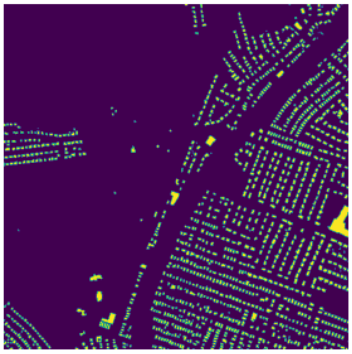


Original 8

segnet: p>0.1

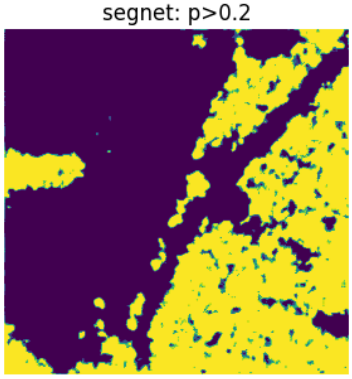


Ground Truth

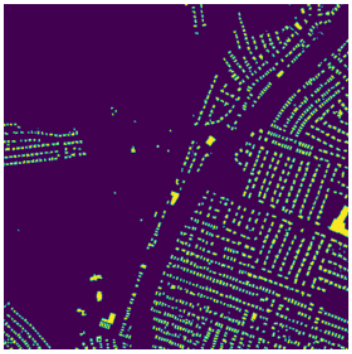


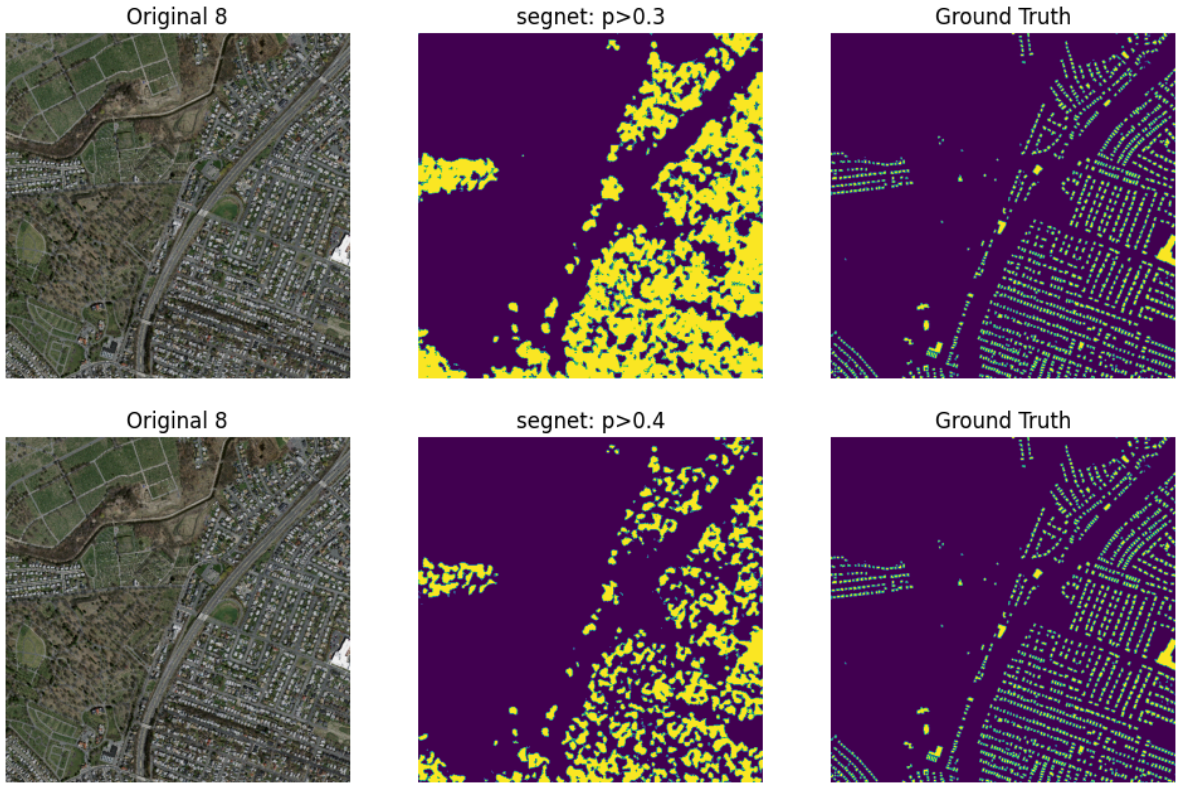
Original 8

segnet: p>0.2



Ground Truth




Original 8
segnet: p>0.3
Ground Truth
Original 8
segnet: p>0.4
Ground Truth

In [ ]: