Library Imports

In [1]:
```
!pip install pydicom
```

```
Collecting pydicom
  Downloading https://files.pythonhosted.org/packages/72/7b/6ed88f82dd33a32cdb43432dab7f84fcd40c49d63251442b3cfe0be983d4/pydicom-2.1.1-py3-none-any.whl (https://files.pythonhosted.org/packages/72/7b/6ed88f82dd33a32cdb43432dab7f84fcd40c49d63251442b3cfe0be983d4/pydicom-2.1.1-py3-none-any.whl) (1.9MB)
     |████████████████████████████████| 1.9MB 13.0MB/s
Installing collected packages: pydicom
Successfully installed pydicom-2.1.1
```

In [2]:
```
import pandas as pd
import numpy as np
import pydicom
from glob2 import glob
import glob2
from tqdm import tqdm
import shutil
import os
import pydicom
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
from PIL import Image
import warnings
from joblib import Parallel, delayed
warnings.filterwarnings("ignore")
```

Download Dataset and Extract

In [3]:
```
# https://gist.github.com/jayspeidell/d10b84b8d3da52df723beacc5b15cb27
!mkdir -p '/root/.kaggle/'
!cp '/content/kaggle.json' '/root/.kaggle/kaggle.json'
# !touch ~/.kaggle/kaggle.json
!chmod 600 ~/.kaggle/kaggle.json
```

In [4]:
```
!kaggle datasets download -d seesee/siim-train-test
```

```
Downloading siim-train-test.zip to /content
 99% 1.90G/1.92G [00:19<00:00, 23.3MB/s]
100% 1.92G/1.92G [00:19<00:00, 108MB/s]
```

In [5]:
```
# https://www.kaggle.com/seesee/siim-train-test
# !wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKi
```

In [6]:
```
print("Extracting 'siim-train-test.zip'\n")
!unzip -qq '/content/siim-train-test.zip'
print("Done Extracting 'siim-train-test.zip'")
```

```
Extracting 'siim-train-test.zip'

Done Extracting 'siim-train-test.zip'
```

Move files

In [7]:
```
!cp "/content/stage_2_sample_submission.csv" "siim/stage_2_sample_submission.csv"
```

In [8]:
```
# Create Directories to move files
destination_1 = 'siim/train_dicom'
destination_2 = 'siim/test_dicom'

if not os.path.isdir(destination_1):
    os.makedirs(destination_1)
if not os.path.isdir(destination_2):
    os.makedirs(destination_2)

def move_files(source, destination):
    '''
    This function takes source and destination paths as input
    and moves files from source to the destination folder.
    '''
    print(source)
    for filename in tqdm(glob2.glob(source)):
        shutil.move(str(filename), destination)

train_path = 'siim/dicom-images-train/**/*.dcm'
test_path = 'siim/dicom-images-test/**/*.dcm'

move_files(train_path, destination_1)
move_files(test_path, destination_2)
```

```
siim/dicom-images-train/**/*.dcm

100%|██████████| 12089/12089 [00:00<00:00, 22102.12it/s]
100%|██████████| 3205/3205 [00:00<00:00, 27592.86it/s]
siim/dicom-images-test/**/*.dcm
```

Prepocessing

In [9]:
```
train_data = pd.read_csv('siim/train-rle.csv', delimiter=',')
train_data.head()
```

Out[9]:

	ImageId	EncodedPixels
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	-1
1	1.2.276.0.7230010.3.1.4.8323329.13666.15178752...	557374 2 1015 8 1009 14 1002 20 997 26 990 32 ...
2	1.2.276.0.7230010.3.1.4.8323329.11028.15178752...	-1
3	1.2.276.0.7230010.3.1.4.8323329.10366.15178752...	514175 10 1008 29 994 30 993 32 991 33 990 34 ...
4	1.2.276.0.7230010.3.1.4.8323329.10016.15178752...	592184 33 976 58 956 73 941 88 926 102 917 109...

In [10]:
```
# add column if the file is duplicate or not
train_data['isDuplicate'] = train_data['ImageId'].duplicated()
train_data.head()
```

Out[10]:

	ImageId	EncodedPixels	isDuplicate
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	-1	False
1	1.2.276.0.7230010.3.1.4.8323329.13666.15178752...	557374 2 1015 8 1009 14 1002 20 997 26 990 32 ...	False
2	1.2.276.0.7230010.3.1.4.8323329.11028.15178752...	-1	False
3	1.2.276.0.7230010.3.1.4.8323329.10366.15178752...	514175 10 1008 29 994 30 993 32 991 33 990 34 ...	False
4	1.2.276.0.7230010.3.1.4.8323329.10016.15178752...	592184 33 976 58 956 73 941 88 926 102 917 109...	False

In [11]:
```
# check where the files are duplicate
dupImages = train_data.index[train_data['isDuplicate']==True]
print(f"We have total {len(dupImages)} duplicate image ids")
```

```
We have total 907 duplicate image ids
```

In [12]:
```
print(f"With duplicates we have total {len(train_data)} files.")
train_data = train_data.drop(list(dupImages))
print(f"Without duplicates we have total {len(train_data)} files.")
```

```
With duplicates we have total 12954 files.
Without duplicates we have total 12047 files.
```

In [13]:
```
train_data = train_data.drop('isDuplicate', axis=1)
train_data['ImagePath'] = 'siim/train_dicom/'+ train_data['ImageId']+'.dcm'
# save the .csv file for further use
train_data.to_csv('train_images_dicom.csv', index=False)
train_data.head()
```

Out[13]:

	ImageId	EncodedPixels	ImagePath
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	-1	siim/train_dicom/1.2.276.0.7230010.3.1.4.83233...
1	1.2.276.0.7230010.3.1.4.8323329.13666.15178752...	557374 2 1015 8 1009 14 1002 20 997 26 990 32 ...	siim/train_dicom/1.2.276.0.7230010.3.1.4.83233...
2	1.2.276.0.7230010.3.1.4.8323329.11028.15178752...	-1	siim/train_dicom/1.2.276.0.7230010.3.1.4.83233...
3	1.2.276.0.7230010.3.1.4.8323329.10366.15178752...	514175 10 1008 29 994 30 993 32 991 33 990 34 ...	siim/train_dicom/1.2.276.0.7230010.3.1.4.83233...
4	1.2.276.0.7230010.3.1.4.8323329.10016.15178752...	592184 33 976 58 956 73 941 88 926 102 917 109...	siim/train_dicom/1.2.276.0.7230010.3.1.4.83233...

In [14]:
```
test_data = pd.read_csv('siim/stage_2_sample_submission.csv', delimiter=',')
test_data = test_data.drop('EncodedPixels', axis=1)
test_data['ImagePath'] = 'siim/test_dicom/'+ test_data['ImageId']+'.dcm'
# save the .csv file for further use
test_data.to_csv('test_images_dicom.csv', index=False)
test_data.head()
```

Out[14]:

	ImageId	ImagePath
0	ID_c68e114ba	siim/test_dicom/ID_c68e114ba.dcm
1	ID_b5a797789	siim/test_dicom/ID_b5a797789.dcm
2	ID_490a04f54	siim/test_dicom/ID_490a04f54.dcm
3	ID_823ca20e1	siim/test_dicom/ID_823ca20e1.dcm
4	ID_5face2763	siim/test_dicom/ID_5face2763.dcm

PNG Conversion

We have the files in the form of **.dcm** files, we cannot use them directly for training the model. So we have to convert them into **.png** format. Also I have to create masks for respective images which also be in **.png** format.

So let's start,

In [15]:
```
train_data = pd.read_csv('train_images_dicom.csv')
train_data.head(2)
```

Out[15]:

	ImageId	EncodedPixels	ImagePath
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	-1	siim/train_dicom/1.2.276.0.7230010.3.1.4.83233...
1	1.2.276.0.7230010.3.1.4.8323329.13666.15178752...	557374 2 1015 8 1009 14 1002 20 997 26 990 32 ...	siim/train_dicom/1.2.276.0.7230010.3.1.4.83233...

In [16]:
```
train_data.shape
```

Out[16]:
```
(12047, 3)
```

In [17]:
```
test_data = pd.read_csv('test_images_dicom.csv')
test_data.head(2)
```

Out[17]:

	ImageId	ImagePath
0	ID_c68e114ba	siim/test_dicom/ID_c68e114ba.dcm
1	ID_b5a797789	siim/test_dicom/ID_b5a797789.dcm

In [18]:
```
test_data.shape
```

Out[18]:
```
(3205, 2)
```

Note: Below 4 cells to be executed only when needed .png conversion

In [19]:
```
# from skimage import exposure
def convert_to_png(filename):
    """
    filename : filename with extension '.dcm' with it's full path
    'This function creates png images from the dicom files'
    """
    # read dicom file
    ds = pydicom.read_file(str(filename))
    # convert dicom image to array
    img = ds.pixel_array
    # resize the image for fast computation
    img = cv2.resize(img, (256, 256))
    # create new file name
    fname = filename.replace(".dcm", ".png")
    fname = fname.replace("_dicom", "_png")
    # save the png image to disk
    cv2.imwrite(fname, img)
```

In [20]:
```
# Create Directories for png files
destination_1 = 'siim/train_png'
destination_2 = 'siim/test_png'

if not os.path.isdir(destination_1):
    os.makedirs(destination_1)
if not os.path.isdir(destination_2):
    os.makedirs(destination_2)
```

In [21]:
```
train_conversion = Parallel(n_jobs=-1, backend='threading')(delayed(
                            convert_to_png)(file) for file in tqdm(train_data['ImagePath'],
                                                   total=len(train_data['ImagePath'])))
```

```
100%|██████████| 12047/12047 [01:31<00:00, 132.13it/s]
```

In [22]:
```
test_conversion = Parallel(n_jobs=-1, backend='threading')(delayed(
                                        convert_to_png)(file) for file in tqdm(test_data['ImagePath'],
                                                   total=len(test_data['ImagePath'])))
```

```
100%|██████████| 3205/3205 [00:24<00:00, 131.69it/s]
```

In [23]:
```
print(os.listdir('siim/train_png/')[0])
print(os.listdir('siim/test_png/')[0])
```

```
1.2.276.0.7230010.3.1.4.8323329.13619.1517875246.877085.png
ID_76b781f21.png
```

Mask Creation

In [24]:
```
train_data = pd.read_csv('train_images_dicom.csv')
train_data.head(2)
```

Out[24]:

	ImageId	EncodedPixels	ImagePath
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	-1	siim/train_dicom/1.2.276.0.7230010.3.1.4.83233...
1	1.2.276.0.7230010.3.1.4.8323329.13666.15178752...	557374 2 1015 8 1009 14 1002 20 997 26 990 32 ...	siim/train_dicom/1.2.276.0.7230010.3.1.4.83233...

In [25]:
```
def rle2mask(rle, width, height):
    """
    RLE to mask conversion provided by competetion organizers with the dataset.
    """
    mask= np.zeros(width* height)
    array = np.asarray([int(x) for x in rle.split()])
    starts = array[0::2]
    lengths = array[1::2]

    current_position = 0
    for index, start in enumerate(starts):
        current_position += start
        mask[current_position:current_position+lengths[index]] = 255
        current_position += lengths[index]

    return mask.reshape(width, height)
```

In [26]:
```
# Create Directories for mask png files
destination_1 = 'siim/train_mask_png'
destination_2 = 'siim/test_mask_png'

if not os.path.isdir(destination_1):
    os.makedirs(destination_1)
if not os.path.isdir(destination_2):
    os.makedirs(destination_2)
```

In [27]:
```
def get_masks(data, destination):
    """
    data : DataFrame with Columns 'ImageId' and ' EncodedPixels'
    destination: Path for saving masks
    """
    print("\nCreating masks...")
    # for each image in the data
    for Id,pix in tqdm(data.values):
        # create filename for mask image
        fname = f"{destination}{Id}.png" #_mask
        # check if the encoding present
        # if present then decode the mask using rle2mask(rle, width, height) function
        if pix!= "-1":
            mask = rle2mask(pix, 1024, 1024).T
            # resize for fast computation
            mask = cv2.resize(mask, (256, 256))
            cv2.imwrite(fname, mask)
        else:
            mask = np.zeros((256, 256), dtype=np.uint8)
            cv2.imwrite(fname, mask)
    print("\nDone!")


# path for saving mask images
train_mask_path = 'siim/train_mask_png/'
# call the function for creating masks
get_masks(train_data[['ImageId', ' EncodedPixels']], train_mask_path)
```

```
  0%|          | 0/12047 [00:00<?, ?it/s]

Creating masks...

100%|██████████| 12047/12047 [00:20<00:00, 575.47it/s]


Done!
```

Now I am creating a DataFrame with Actual Images and their corresponding Masks

In [28]:
```
final_data = pd.read_csv('train_images_dicom.csv')
final_data = final_data.drop(['ImageId', ' EncodedPixels'], axis=1)
final_data['ImagePath'] = 'siim/train_png/'+ train_data['ImageId']+'.png'
final_data['MaskPath'] = 'siim/train_mask_png/'+ train_data['ImageId']+'.png' # _mask
# save file for future use
final_data.to_csv("final_data.csv", index=False)
print("Saved 'final_data.csv'")
final_data.head()
```

```
Saved 'final_data.csv'
```

Out[28]:

	ImagePath	MaskPath
0	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
1	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
2	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
3	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
4	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...

In [29]:
```
test_data = pd.read_csv('test_images_dicom.csv')
test_data.head()
```

Out[29]:

	ImageId	ImagePath
0	ID_c68e114ba	siim/test_dicom/ID_c68e114ba.dcm
1	ID_b5a797789	siim/test_dicom/ID_b5a797789.dcm
2	ID_490a04f54	siim/test_dicom/ID_490a04f54.dcm
3	ID_823ca20e1	siim/test_dicom/ID_823ca20e1.dcm
4	ID_5face2763	siim/test_dicom/ID_5face2763.dcm

In [30]:
```
final_test_data = pd.read_csv('test_images_dicom.csv')
final_test_data = final_test_data.drop(['ImageId'], axis=1)
final_test_data['ImagePath'] = 'siim/test_png/'+ test_data['ImageId']+'.png'
# final_test_data['MaskPath'] = 'siim/test_mask_png/'+ train_data['ImageId']+'_mask.png'
# save file for future use
final_test_data.to_csv("final_test_data.csv", index=False)
print("Saved 'final_test_data.csv'")
final_test_data.head()
```

```
Saved 'final_test_data.csv'
```

Out[30]:

	ImagePath
0	siim/test_png/ID_c68e114ba.png
1	siim/test_png/ID_b5a797789.png
2	siim/test_png/ID_490a04f54.png
3	siim/test_png/ID_823ca20e1.png
4	siim/test_png/ID_5face2763.png

Read CSV

In [31]:
```
final_data = pd.read_csv('final_data.csv')
# final_data = final_data.drop(['ImageId', ' EncodedPixels'], axis=1)
# final_data['ImagePath'] = 'siim/train_png/'+ train_data['ImageId']+'.png'
# final_data['MaskPath'] = 'siim/train_mask_png/'+ train_data['ImageId']+'_mask.png'
# # save file for future use
# final_data.to_csv("final_data.csv", index=False)
# print("Saved 'final_data.csv'")
final_data.head()
```

Out[31]:

	ImagePath	MaskPath
0	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
1	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
2	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
3	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
4	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...

TF DataPipeline

In [32]:
```
import tensorflow as tf
import keras
# tf.enable_eager_execution()
import os
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
# from hilbert import hilbertCurve
import imgaug.augmenters as iaa
import numpy as np
import random as rn
from keras import backend as K
#K.set_image_data_format('channels_first')
```

In [33]:
```
if tf.test.is_gpu_available(cuda_only=True):
  print('working on gpu')
else:
  print('warning: working on cpu')
# print(tf.__version__)
# print(keras.__version__)

print("Tensorflow:",tf.__version__)
print("Keras:",keras.__version__)
```

```
WARNING:tensorflow:From <ipython-input-33-fdbdf1bb57a1>:1: is_gpu_available (from tensorflow.python.framework.test_uti
l) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
working on gpu
Tensorflow: 2.3.0
Keras: 2.4.3
```

In [34]:
```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, LearningRateScheduler, ReduceLROnPlateau, Callback
```

In [35]:
```
all_mask_fn = glob2.glob('siim/train_mask_png/*')
# all_mask_fn = glob.glob('train/*')

mask_df = pd.DataFrame()
mask_df['file_names'] = all_mask_fn
mask_df['mask_percentage'] = 0
mask_df.set_index('file_names',inplace=True)
for fn in all_mask_fn:
    mask_df.loc[fn,'mask_percentage'] = np.array(Image.open(fn)).sum()/(256*256*255) #255 is bcz img range is 255

mask_df.reset_index(inplace=True)
mask_df['labels'] = 0
mask_df.loc[mask_df.mask_percentage>0,'labels'] = 1
#sns.distplot(mask_df.mask_percentage.values)
```

In [36]:
```
from sklearn.model_selection import train_test_split
X_train, X_val = train_test_split(final_data, stratify=mask_df.labels, test_size=0.15, random_state=42)

print(X_train.shape)
print(X_val.shape)
```

```
(10239, 2)
(1808, 2)
```

In [37]:
```
# image_shapes = [read_image(x).shape for x in X_train['ImagePath']]
#         # image_mask = read_mask(y)
```

In [38]:
```
# image_shapes
```

In [39]:
```
# plt.plot(image_shapes)
```

In [40]:
```
# 1. To load the dataset: image and mask paths
# 2. Building the TensorFlow Input Data Pipeline using tf.data API

import os
import numpy as np
import cv2
from glob2 import glob
import tensorflow as tf
from tensorflow.keras.layers import Concatenate
import imgaug.augmenters as iaa

# create augmentations
aug2 = iaa.Fliplr(1)
aug3 = iaa.Flipud(1)
aug4 = iaa.Emboss(alpha=(1), strength=1)
aug5 = iaa.DirectedEdgeDetect(alpha=(0.8), direction=(1.0))
aug6 = iaa.Sharpen(alpha=(1.0), lightness=(1.5))

def read_image(path):
    """
    This reads image
    ----------------
    path : image path
    ----------------
    """
    # image = tf.io.read_file(img_path)
    # image = tf.image.decode_png(image, channels=3)
    # image = tf.image.convert_image_dtype(image, tf.uint8)

    x = tf.io.read_file(path)
    x = tf.image.decode_png(x, channels=3)
    x = tf.image.convert_image_dtype(x, tf.float32)
    # x = x / 255.0
    # x = tf.image.convert_image_dtype(x, tf.float32)

    return x

def read_mask(path):
    """
    This reads mask
    ----------------
    path : mask path
    ----------------
    """
    y = tf.io.read_file(path)
    y = tf.image.decode_png(y, channels=1)
    y = tf.image.convert_image_dtype(y, tf.float32)
    # y = y / 255.0
    # x = np.expand_dims(x, axis=-1)
    # y = tf.image.convert_image_dtype(y, tf.float32)
    return y

def preprocess(x, y, augment):
    """
    This function reads images and masks and returns them
    ----------------
    x : image path
    y : mask path
    ----------------
    """
# def f(x, y, augment):
    # x = tf.image.decode_png(x, channels=3)
    # y =  tf.image.decode_png(y, channels=1)

image = read_image(x)
image_mask = read_mask(y)
image = tf.transpose(image, [2,0,1])
image_mask = tf.transpose(image_mask, [2,0,1])

    # Augmentations
    # a = np.random.uniform()

    # if augment==True:
    #     if a<=0.2 and a>=0.8:
    #         image = image
    #         image_mask = image_mask
    #     elif a>0.2 and a<0.4:
    #         image = aug2.augment_image(image)
    #         image_mask = aug2.augment_image(image_mask)
    #     elif a>0.4 and a<0.6:
    #         image = aug3.augment_image(image)
    #         image_mask = aug3.augment_image(image_mask)
    #     elif a>0.6 and a<0.8:
    #         image = aug4.augment_image(image)
    #         image_mask = aug4.augment_image(image_mask)
    #     elif a>0.8 and a<0.9:
    #         image = aug5.augment_image(image)
    #         image_mask = image_mask
```

```
        #     elif a>=0.5 and a<=0.7:
        #         image = aug6.augment_image(image)
        #         image_mask = aug6.augment_image(image_mask)

    # return image, image_mask


    # images, masks = tf.numpy_function(f, [x, y, augment], [tf.float32, tf.float32])
    # print(type(images))
    # images.set_shape([3, 256, 256])
    # masks.set_shape([1, 256, 256])
    # images = tf.image.convert_image_dtype(images, tf.float32)
    # print(masks)
    image_mask = Concatenate(axis=0)([image_mask, image_mask])
    # masks = tf.image.convert_image_dtype(masks, tf.float32)
    # print(masks)
    return image, image_mask

def tf_dataset(x, y, batch=8, augment=False):
    """
    This function creates input data pipeline
    --------------------
    x : all images paths
    y : all masks paths
    batch : batch size
    --------------------
    """
    augment=augment
    dataset = tf.data.Dataset.from_tensor_slices((x, y))
    dataset = dataset.shuffle(buffer_size=1000)
    dataset = dataset.map(lambda x,y: preprocess(x, y, augment), num_parallel_calls=tf.data.experimental.AUTOTUNE)
    dataset = dataset.batch(batch)
    dataset = dataset.repeat()
    dataset = dataset.prefetch(tf.data.experimental.AUTOTUNE)
    return dataset

if __name__ == "__main__":

    tr_images, tr_masks = X_train['ImagePath'], X_train['MaskPath']
    print(f"Images: {len(tr_images)} - Masks: {len(tr_masks)}")
    train_dataset = tf_dataset(tr_images, tr_masks, augment=False, batch=12)

    val_images, val_masks = X_val['ImagePath'], X_val['MaskPath']
    print(f"Images: {len(val_images)} - Masks: {len(val_masks)}")
    val_dataset = tf_dataset(val_images, val_masks, augment=False, batch=12)

    # for x, y in dataset:
    #     x = x[0] * 255
    #     y = y[0] * 255

    #     x = x.numpy()
    #     y = y.numpy()

    #     cv2.imwrite("image.png", x)

    #     y = np.squeeze(y, axis=-1)
    #     cv2.imwrite("mask.png", y)

        # break
```

```
Images: 10239 - Masks: 10239
Images: 1808 - Masks: 1808
```

In [41]:
```
print(train_dataset)

print(val_dataset)
```

```
<PrefetchDataset shapes: ((None, 3, None, None), (None, 2, None, None)), types: (tf.float32, tf.float32)>
<PrefetchDataset shapes: ((None, 3, None, None), (None, 2, None, None)), types: (tf.float32, tf.float32)>
```

In [42]:
```
# smooth = 1e-15
# def dice_coef(y_true, y_pred):
#     y_true = tf.keras.layers.Flatten()(y_true)
#     y_pred = tf.keras.layers.Flatten()(y_pred)
#     intersection = tf.reduce_sum(y_true * y_pred)
#     return (2. * intersection + smooth) / (tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) + smooth)

# def dice_loss(y_true, y_pred):
#     return 1.0 - dice_coef(y_true, y_pred)

# def iou(y_true, y_pred):
#     def f(y_true, y_pred):
#         intersection = (y_true * y_pred).sum()
#         union = y_true.sum() + y_pred.sum() - intersection
#         x = (intersection + smooth) / (union + smooth)
#         x = x.astype(np.float32)
#         return x
#     return tf.numpy_function(f, [y_true, y_pred], tf.float32)

# def bce_dice_loss(y_true, y_pred):
#     return binary_crossentropy(y_true, y_pred) + dice_loss(y_true, y_pred)
```

Double UNet

In [43]:
```
import tensorflow as tf
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model
from tensorflow.keras.applications import *

def squeeze_excite_block(inputs, ratio=8):
    init = inputs
    channel_axis = 1
    filters = init.shape[channel_axis]
    se_shape = (filters, 1, 1)

    se = GlobalAveragePooling2D(data_format='channels_first')(init)
    # se = Reshape(se_shape)(se)
    se = Dense(filters // ratio, activation='relu', kernel_initializer='he_normal', use_bias=False)(se)
    se = Dense(filters, activation='sigmoid', kernel_initializer='he_normal', use_bias=False)(se)
    se = Reshape(se_shape)(se)
    print(se.shape)

    x = Multiply()([init, se])
    return x

def conv_block(inputs, filters):
    x = inputs

    x = Conv2D(filters, (3, 3), padding="same",data_format='channels_first')(x)
    x = BatchNormalization(axis=1)(x)
    x = Activation('relu')(x)

    x = squeeze_excite_block(x)

    return x

def encoder1(inputs):
    skip_connections = []

    model = VGG19(include_top=False, weights='imagenet', input_tensor=inputs, input_shape=(3,256,256))
    # names = ["block1_conv2", "block2_conv2", "block3_conv4", "block4_conv4"]
    names = [ "block1_conv2", "block2_conv2"]
    for name in names:
        skip_connections.append(model.get_layer(name).output)

    # output = model.get_layer("block5_conv4").output
    output = model.get_layer("block3_conv4").output
    # print("<<VGG19")
    # for v in range(len(skip_connections)):
    #     print(skip_connections[v].shape)
    # print("VGG19>>")
    return output, skip_connections

def decoder1(inputs, skip_connections):
    # num_filters = [256, 128, 64, 32]
    num_filters = [64, 32]
    skip_connections.reverse()

    x = inputs

    for i, f in enumerate(num_filters):
        # x = UpSampling2D((2, 2), interpolation='bilinear')(x)
        x = UpSampling2D((2, 2), interpolation='bilinear',data_format='channels_first')(x)
        # print(x.shape, skip_connections[i].shape)
        x = Concatenate(axis=1)([x, skip_connections[i]])
        x = conv_block(x, f)

    return x

# def encoder2(inputs):
#     skip_connections = []
#
#     output = DenseNet121(include_top=False, weights='imagenet')(inputs)
#     model = tf.keras.models.Model(inputs, output)
#
#     names = ["input_2", "conv1/relu", "pool2_conv", "pool3_conv"]
#     for name in names:
#         skip_connections.append(model.get_layer(name).output)
#     output = model.get_layer("pool4_conv").output
#
#     return output, skip_connections

def encoder2(inputs):
    # num_filters = [32, 64, 128, 256]
    num_filters = [32, 64]
    skip_connections = []
    x = inputs
```

```
    for i, f in enumerate(num_filters):
        x = conv_block(x, f)
        skip_connections.append(x)
        x = MaxPool2D((2, 2),data_format='channels_first')(x)

    return x, skip_connections

def decoder2(inputs, skip_1, skip_2):
    # num_filters = [256, 128, 64, 32]
    num_filters = [256, 128]
    skip_2.reverse()
    x = inputs

    for i, f in enumerate(num_filters):
        x = UpSampling2D((2, 2), interpolation='bilinear',data_format='channels_first')(x)
        x = Concatenate(axis=1)([x, skip_1[i], skip_2[i]])
        x = conv_block(x, f)

    return x

def output_block(inputs):
    x = Conv2D(1, (1, 1), padding="same",data_format='channels_first')(inputs)
    x = Activation('sigmoid')(x)
    return x

def Upsample(tensor, size):
    """Bilinear upsampling"""
    def _upsample(x, size):
        return tf.image.resize(images=x, size=size)
    return Lambda(lambda x: _upsample(x, size), output_shape=size)(tensor)

def ASPP(x, filter):
    shape = x.shape

    y1 = AveragePooling2D(pool_size=(shape[2], shape[3]),data_format='channels_first')(x)
    y1 = Conv2D(filter, 1, padding="same",data_format='channels_first')(y1)
    y1 = BatchNormalization(axis=1)(y1)
    y1 = Activation("relu")(y1)
    y1 = UpSampling2D((shape[2], shape[3]), interpolation='bilinear',data_format='channels_first')(y1)

    y2 = Conv2D(filter, 1, dilation_rate=1, padding="same", use_bias=False,data_format='channels_first')(x)
    y2 = BatchNormalization(axis=1)(y2)
    y2 = Activation("relu")(y2)

    y3 = Conv2D(filter, 3, dilation_rate=6, padding="same", use_bias=False,data_format='channels_first')(x)
    y3 = BatchNormalization(axis=1)(y3)
    y3 = Activation("relu")(y3)

    y4 = Conv2D(filter, 3, dilation_rate=12, padding="same", use_bias=False,data_format='channels_first')(x)
    y4 = BatchNormalization(axis=1)(y4)
    y4 = Activation("relu")(y4)

    y5 = Conv2D(filter, 3, dilation_rate=18, padding="same", use_bias=False,data_format='channels_first')(x)
    y5 = BatchNormalization(axis=1)(y5)
    y5 = Activation("relu")(y5)

    y = Concatenate(axis=1)([y1, y2, y3, y4, y5])

    y = Conv2D(filter, 1, dilation_rate=1, padding="same", use_bias=False,data_format='channels_first')(y)
    y = BatchNormalization(axis=1)(y)
    y = Activation("relu")(y)

    return y

def build_model(shape):
    inputs = Input(shape)
    x, skip_1 = encoder1(inputs)
    x = ASPP(x, 64)
    x = decoder1(x, skip_1)
    outputs1 = output_block(x)

    x = inputs * outputs1

    x, skip_2 = encoder2(x)
    x = ASPP(x, 64)
    x = decoder2(x, skip_1, skip_2)
    outputs2 = output_block(x)
    outputs = Concatenate(axis=1)([outputs1, outputs2])

    model = Model(inputs, outputs)
    return model
# inp_shape = (256,256,3)
# model = build_model(inp_shape)
```

In [44]:
```
# K.set_image_data_format('channels_last')
K.set_image_data_format('channels_first')
inp_shape = (3,256,256)
model = build_model(inp_shape)
model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5 (https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5)
80142336/80134624 [==============================] - 1s 0us/step
(None, 64, 1, 1)
(None, 32, 1, 1)
(None, 32, 1, 1)
(None, 64, 1, 1)
(None, 256, 1, 1)
(None, 128, 1, 1)
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 3, 256, 256) 0
__________________________________________________________________________________________________
block1_conv1 (Conv2D)           (None, 64, 256, 256) 1792        input_1[0][0]
__________________________________________________________________________________________________
block1_conv2 (Conv2D)           (None, 64, 256, 256) 36928       block1_conv1[0][0]
```

In [45]:
```
model.inputs, model.outputs
```

Out[45]:
```
([<tf.Tensor 'input_1:0' shape=(None, 3, 256, 256) dtype=float32>],
 [<tf.Tensor 'concatenate_6/concat:0' shape=(None, 2, 256, 256) dtype=float32>])
```

Foreground and Background Percentage.

Note: Only run below if needed it takes more time

Mask and Background pixels

In [ ]:
```
fore_ground = []
back_ground = []


for m in tqdm(final_data['MaskPath']):
    mask = read_mask(m)
    # print(type(mask))
    # print(mask.shape)
    f_mask = K.flatten(mask).numpy()
    zero_count = list(f_mask).count(0.0)
    back_ground.append(zero_count)
    one_count = list(f_mask).count(1.0)
    fore_ground.append(one_count)
    # print(list(f_mask).count(1.0))
    # break

pixel_df = pd.DataFrame({'fore_ground':fore_ground, 'back_ground':back_ground})
pixel_df.to_csv('pixel_count_df.csv', index=False)
pixel_df.head()
```

```
100%|██████████| 12047/12047 [36:32<00:00,  5.49it/s]
```

Out[24]:

	fore_ground	back_ground
0	0	65536
1	1895	63567
2	0	65536
3	310	65200
4	2926	62444

In [ ]:
```
from IPython.display import Audio
sound_file = 'https://www.soundjay.com/button/beep-01a.mp3'

Audio(sound_file, autoplay=True)
```

Out[25]:

0:00 / 0:00

In [ ]:
```
pixel_df = pd.read_csv('pixel_count_df.csv')
pixel_df.head()
```

Out[26]:

	fore_ground	back_ground
0	0	65536
1	1895	63567
2	0	65536
3	310	65200
4	2926	62444

In [ ]:
```
FG_pixels = np.sum(pixel_df['fore_ground'])
BG_pixels = np.sum(pixel_df['back_ground'])
print(f"Fore Ground Pixels: {FG_pixels}")
print(f"Back Ground Pixels: {BG_pixels}")
FG = FG_pixels/(FG_pixels+BG_pixels)
BG = BG_pixels/(FG_pixels+BG_pixels)
print(f"Fore Ground Pixel Percentage: {FG*100} %")
print(f"Back Ground Pixel Percentage: {BG*100} %")
```

```
Fore Ground Pixels: 1884621
Back Ground Pixels: 787493342
Fore Ground Pixel Percentage: 0.23874760739932133 %
Back Ground Pixel Percentage: 99.76125239260068 %
```

In [ ]:
```
round(1/(FG*100), 1), round(1/(BG*100),2)
```

Out[77]:
```
(4.2, 0.01)
```

- 4.2 and 0.01 are the values I can use for weighting the metrics
- But the 0.01 is very low value so I will use 4 and 0.1 as my weight values

Weighted Loss Function and Metric

In [ ]:
```python
def w_bce( y_true, y_pred, weight1=4.  , weight0=1. ) :
    """
    This gives weighted Binary Cross Entropy
    ----------------------------------------
    y_true: Ground truth values
    y_pred: Predicted values
    weight1: value to be penalized for class 1
    weight0: value to be penalized for class 0
    ----------------------------------------
    """
    y_true = K.flatten(y_true)
    y_pred = K.flatten(y_pred)
    y_true = K.clip(y_true, K.epsilon(), 1-K.epsilon())
    y_pred = K.clip(y_pred, K.epsilon(), 1-K.epsilon())
    logloss = -(y_true * K.log(y_pred) * weight1 + (1 - y_true) * K.log(1 - y_pred) * weight0 )
    return K.mean(logloss, axis=-1)


def w_dice_coef(y_true, y_pred, w_0=1., w_1=4.):
    """
    ref:

    This gives weighted Dice Coefficient
    ------------------------------------
    y_true: Ground truth values
    y_pred: Predicted values
    ------------------------------------
    """
    # mean = 0.21649066
    # w_1 = 1/mean**2
    # w_0 = 1/(1-mean)**2
    y_true_f_1 = K.flatten(y_true)
    y_pred_f_1 = K.flatten(y_pred)
    y_true_f_0 = K.flatten(1-y_true)
    y_pred_f_0 = K.flatten(1-y_pred)

    intersection_0 = K.sum(y_true_f_0 * y_pred_f_0)
    intersection_1 = K.sum(y_true_f_1 * y_pred_f_1)

    return (2 * (w_0 * intersection_0 + w_1 * intersection_1) /
            ((w_0 * (K.sum(y_true_f_0) + K.sum(y_pred_f_0))) + (w_1 * (K.sum(y_true_f_1) + K.sum(y_pred_f_1)))))


smooth = 1.
def w_bce_dice_loss(y_true, y_pred):
    """
    Combined dice and binary cross entropy loss
    """
    return w_bce(y_true, y_pred, weight1=4. , weight0=1.) + (1 - w_dice_coef(y_true, y_pred, w_0=1., w_1=4.))
```

In [ ]:
```python
y_true = np.random.rand(1,2, 256, 256).astype(np.float32)
y_pred = np.random.rand(1,2, 256,256).astype(np.float32)

# w_bce( y_true, y_pred, weight1=0.1 , weight0=4. )
w_dice_coef(y_true, y_pred, w_0=1., w_1=4.)
w_bce_dice_loss(y_true, y_pred)
```

Out[56]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=2.5579276>
```

Compile and Fit

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remou
nt=True).
```

In [ ]:
```
import datetime
from tensorflow.keras.callbacks import *
from tensorflow.keras.losses import binary_crossentropy
# tf.keras.backend.set_image_data_format('channels_last')
# Define optimizer
optim = tf.keras.optimizers.Adam(0.0001)
# Clear any logs from previous runs
!rm -rf /content/logs/model_1/fit
# "/content/model_save/model/weights-{epoch:02d}-{val_auc_score:.4f}.hdf5"
# filepath= "best_Double_Unet-{epoch:02d}-{val_dice_coef:.4f}.hdf5"
filepath= "best_Double_Unet_weighted_metric.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_w_dice_coef', verbose=1, save_best_only=True, mode='max')
# lrschedule = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.000001)
# lrschedule = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5)
# earlystop = EarlyStopping(monitor='val_dice_coef', min_delta=0.25, patience=5, verbose=1)
log_dir="/content/logs/Double_Unet/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=0)
callback_list = [checkpoint, tensorboard_callback]
```

In [ ]:
```
train_dataset, val_dataset
```

Out[65]:
```
(<PrefetchDataset shapes: ((None, 3, None, None), (None, 2, None, None)), types: (tf.float32, tf.float32)>,
 <PrefetchDataset shapes: ((None, 3, None, None), (None, 2, None, None)), types: (tf.float32, tf.float32)>)
```

In [ ]:
```
# model(np.random.rand(1,3,256,256))
```

In [ ]:
```
# true = np.random.rand(1,2, 256, 256).astype(np.float32)
# pred = np.random.rand(1,2,256,256).astype(np.float32)
# bce_dice_loss(true.astype(np.float32),pred.astype(np.float32))
# # binary_crossentropy(true, pred)
```

In [ ]:
```
model.load_weights( '/content/drive/MyDrive/27_Case_study_2/best_Double_Unet.hdf5')
print('loaded saved model {best_Double_Unet.hdf5}')
```

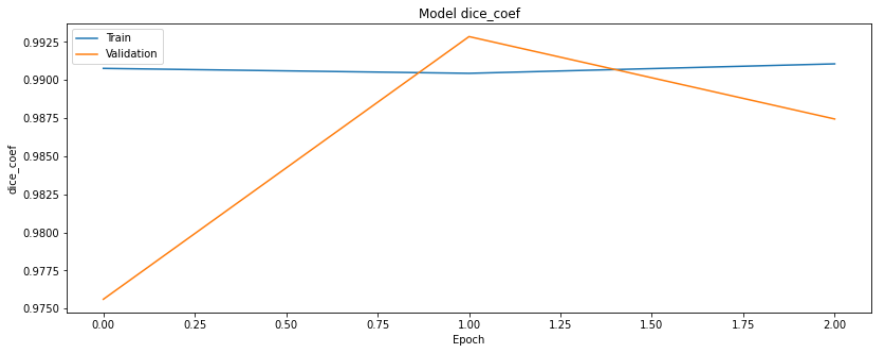
```
loaded saved model {best_Double_Unet.hdf5}
```

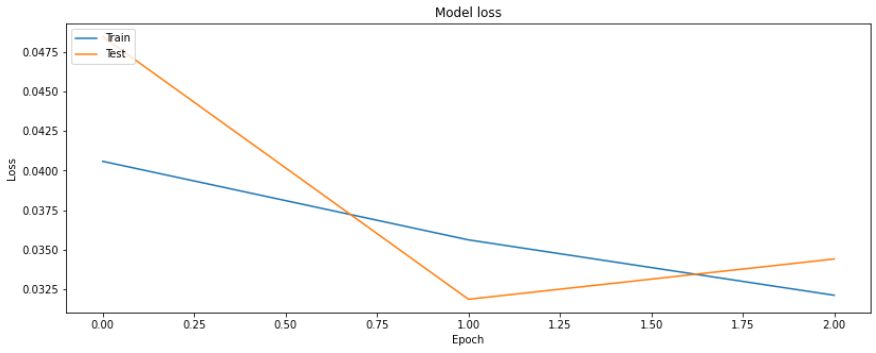
In [ ]:
```
model.compile(optimizer=optim, loss=w_bce_dice_loss, metrics=[w_dice_coef])
history = model.fit(train_dataset,validation_data=val_dataset,
                    epochs=3, steps_per_epoch=10239//12, validation_steps=1808//12, verbose=1,
                    callbacks=callback_list)
```

```
Epoch 1/3
  2/853 [..............................] - ETA: 15:55 - loss: 0.0224 - w_dice_coef: 0.9943WARNING:tensorflow:Callbacks
method `on_train_batch_end` is slow compared to the batch time (batch time: 0.4395s vs `on_train_batch_end` time: 0.998
5s). Check your callbacks.
853/853 [==============================] - ETA: 0s - loss: 0.0406 - w_dice_coef: 0.9908WARNING:tensorflow:Callbacks met
hod `on_test_batch_end` is slow compared to the batch time (batch time: 0.0141s vs `on_test_batch_end` time: 0.4266s).
Check your callbacks.

Epoch 00001: val_w_dice_coef improved from -inf to 0.97562, saving model to best_Double_Unet_weighted_metric.hdf5
853/853 [==============================] - 1175s 1s/step - loss: 0.0406 - w_dice_coef: 0.9908 - val_loss: 0.0485 - val_
w_dice_coef: 0.9756
Epoch 2/3
853/853 [==============================] - ETA: 0s - loss: 0.0356 - w_dice_coef: 0.9904
Epoch 00002: val_w_dice_coef improved from 0.97562 to 0.99286, saving model to best_Double_Unet_weighted_metric.hdf5
853/853 [==============================] - 1171s 1s/step - loss: 0.0356 - w_dice_coef: 0.9904 - val_loss: 0.0318 - val_
w_dice_coef: 0.9929
Epoch 3/3
853/853 [==============================] - ETA: 0s - loss: 0.0321 - w_dice_coef: 0.9911
Epoch 00003: val_w_dice_coef did not improve from 0.99286
853/853 [==============================] - 1172s 1s/step - loss: 0.0321 - w_dice_coef: 0.9911 - val_loss: 0.0344 - val_
w_dice_coef: 0.9874
```

In [ ]:
```
# Plot training & validation iou_score values
plt.figure(figsize=(30, 5))
plt.subplot(121)
plt.plot(history.history['w_dice_coef'])
plt.plot(history.history['val_w_dice_coef'])
plt.title('Model dice_coef')
plt.ylabel('dice_coef')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
# Plot training & validation loss values
plt.subplot(122)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```





Inference

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

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

In [ ]:
```
# !cp '/content/best_Double_Unet_weighted_metric.hdf5' '/content/drive/MyDrive/27_Case_study_2/best_Double_Unet_weighted_
```

In [ ]:
```
# !cp -r '/content/logs' '/content/drive/MyDrive/27_Case_study_2/weighted_double_unet_logs'
```

In [ ]:
```
# %load_ext tensorboard
```

In [ ]:
```
# %tensorboard --logdir="/content/logs/Double_Unet/fit/"
```

In [ ]:
```
# model.load_weights( '/content/drive/MyDrive/27_Case_study_2/best_Double_Unet.hdf5')
# model.load_weights( '/content/drive/MyDrive/27_Case_study_2/best_Double_Unet_weighted_metric.hdf5')
```

In [ ]:
```
X_val.columns
```

Out[47]:
```
Index(['ImagePath', 'MaskPath'], dtype='object')
```

In [49]:
```
def read_image(path):
    """
    This reads image
    ----------------
    path : image path
    ----------------
    """
    # image = tf.io.read_file(img_path)
    # image = tf.image.decode_png(image, channels=3)
    # image = tf.image.convert_image_dtype(image, tf.uint8)

    x = tf.io.read_file(path)
    x = tf.image.decode_png(x, channels=3)
    x = tf.image.convert_image_dtype(x, tf.float32)
    # x = x / 255.0
    # x = tf.image.convert_image_dtype(x, tf.float32)

    return x

def read_mask(path):
    """
    This reads mask
    ----------------
    path : mask path
    ----------------
    """
    y = tf.io.read_file(path)
    y = tf.image.decode_png(y, channels=1)
    # y = y / 255.0
    # x = np.expand_dims(x, axis=-1)
    # y = tf.image.convert_image_dtype(y, tf.float32)
    return y
```

In [50]:
```
def Predict(x, y):
    """
    This function predicts the mask for given input image and plots it.
    -------------------------------------------------------------------
    x     : image path
    y     : original mask path
    return: the original image, original mask and predicted mask
    -------------------------------------------------------------------
    """
    # read the original image
    image_orig = read_image(x)
    # read the original mask
    image_mask_orig = read_mask(y)
    # reshape image and mask as first channel image format
    image = tf.transpose(image_orig, [2,0,1])
    image_mask = tf.transpose(image_mask_orig, [2,0,1])
    # predict the mask using trained model
    predict_mask = model.predict(tf.expand_dims(image, axis=0))
    predict_mask = tf.transpose(predict_mask, [0,2,3,1])
    # return the original image, original mask and predicted mask
    return image_orig, image_mask_orig, predict_mask
```

In [ ]:
```
mask_df.columns
```

Out[50]:
```
Index(['file_names', 'mask_percentage', 'labels'], dtype='object')
```

In [ ]:
```
mask_df['labels'].value_counts()
```

Out[51]:
```
0    9378
1    2669
Name: labels, dtype: int64
```

In [ ]:
```
len(final_data), len(mask_df)
```

Out[52]:
```
(12047, 12047)
```

Model without weighting

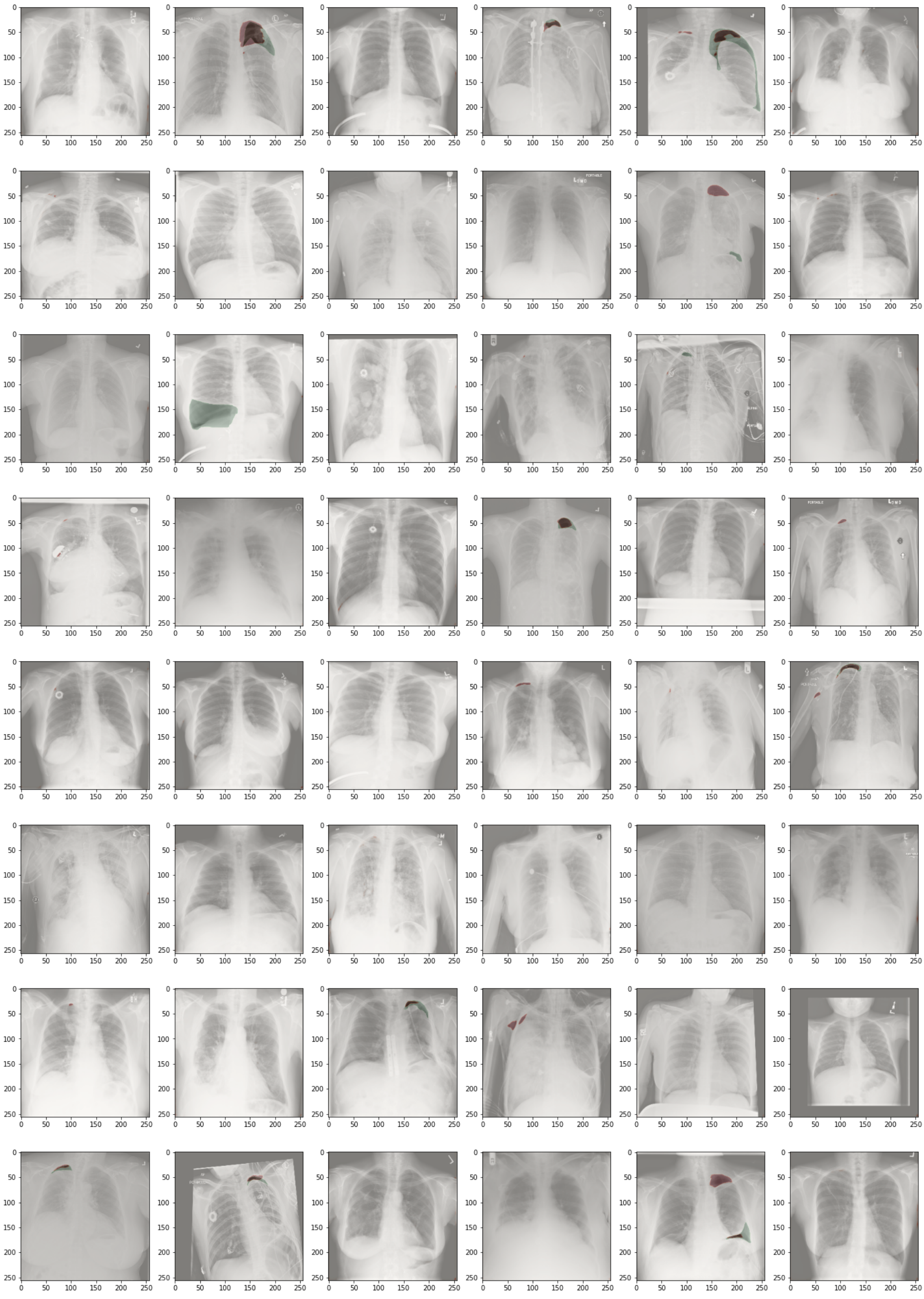
In [ ]:
```
max_images = 48
grid_width = 6
grid_height = int(max_images / grid_width)
fig, axs = plt.subplots(grid_height, grid_width, figsize=(grid_width*4, grid_height*4))
fig.subplots_adjust(bottom = 0.05)

model.load_weights( '/content/drive/MyDrive/27_Case_study_2/best_Double_Unet.hdf5')

for i in range(max_images):
    img = final_data['ImagePath'].values[i]
    msk = final_data['MaskPath'].values[i]
    image_orig, image_mask_orig, predict_mask = Predict(img, msk)

    ax = axs[int(i / grid_width), i % grid_width]
    ax.imshow(image_orig)
    ax.imshow(image_mask_orig[:,:,0], cmap='Greens', alpha = 0.3)
    ax.imshow(np.squeeze(predict_mask[:,:,:,1]), cmap='Reds', alpha = 0.3)
    # ax.axis('off')
plt.suptitle("Red: Predicted Mask, Green: Original Mask", fontsize=20)
plt.show()
```

Red: Predicted Mask, Green: Original Mask



Model after weighted metric

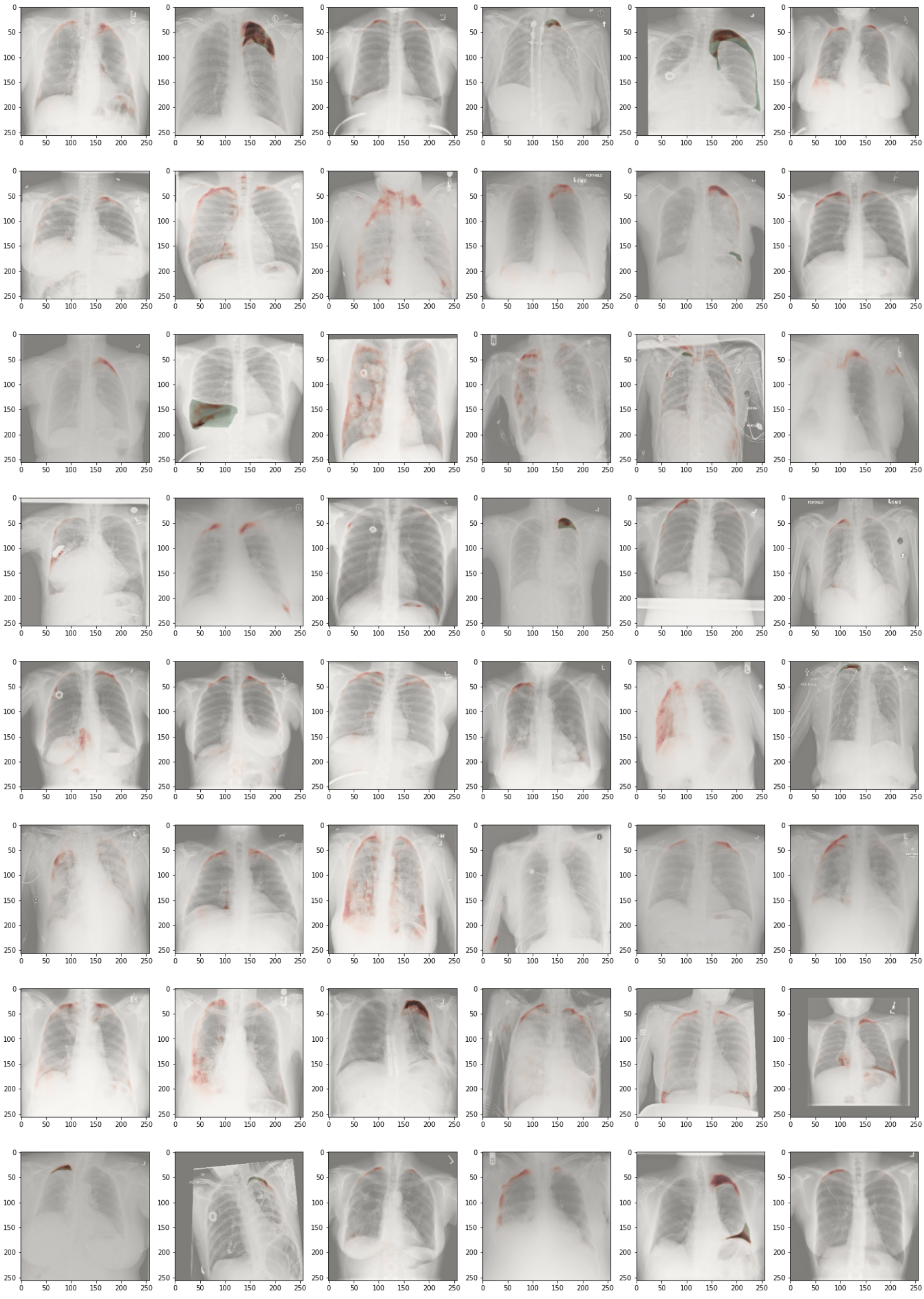
In [ ]:
```
max_images = 48
grid_width = 6
grid_height = int(max_images / grid_width)
fig, axs = plt.subplots(grid_height, grid_width, figsize=(grid_width*4, grid_height*4))
fig.subplots_adjust(bottom = 0.05)

model.load_weights('/content/drive/MyDrive/27_Case_study_2/best_Double_Unet_weighted_metric.hdf5')

for i in range(max_images):
    img = final_data['ImagePath'].values[i]
    msk = final_data['MaskPath'].values[i]
    image_orig, image_mask_orig, predict_mask = Predict(img, msk)

    ax = axs[int(i / grid_width), i % grid_width]
    ax.imshow(image_orig)
    ax.imshow(image_mask_orig[:,:,0], cmap='Greens', alpha = 0.3)
    ax.imshow(np.squeeze(predict_mask[:,:,:,1]), cmap='Reds', alpha = 0.3)
    # ax.axis('off')
plt.suptitle("Red: Predicted Mask, Green: Original Mask", fontsize=20)
plt.show()
```

Red: Predicted Mask, Green: Original Mask



Let's analyze the outputs for all datapoints

Weighted Metric

In [ ]:
```
def get_dice_for_each_image(final_data):
    dice_coefs = []
    for i in tqdm(range(len(final_data))):
        img = final_data['ImagePath'].values[i]
        msk = final_data['MaskPath'].values[i]
        image_orig, image_mask_orig, predict_mask = Predict(img, msk)
        image_mask_orig = np.expand_dims(image_mask_orig, axis=0)
        image_mask_orig = Concatenate(axis=0)([image_mask_orig, image_mask_orig])
        y_true = K.flatten(image_mask_orig)
        y_pred = K.flatten(predict_mask)
        d_c = w_dice_coef(y_true, y_pred, w_0=0.1, w_1=4.)
        # print(d_c)
        dice_coefs.append(d_c)
    print("Done")
    return dice_coefs
```

In [ ]:
```
model.load_weights( '/content/drive/MyDrive/27_Case_study_2/best_Double_Unet.hdf5')
print('loaded saved model {best_Double_Unet.hdf5}')
dice_coefs_old = get_dice_for_each_image(final_data)
dice_coefs_old = [dc.numpy() for dc in dice_coefs_old]
model.load_weights( '/content/drive/MyDrive/27_Case_study_2/best_Double_Unet_weighted_metric.hdf5')
print('loaded saved model {best_Double_Unet_weighted_metric.hdf5}')
dice_coefs_new = get_dice_for_each_image(final_data)
dice_coefs_new = [dc.numpy() for dc in dice_coefs_new]
```

In [ ]:
```
output_df = final_data.copy()
output_df['dice_coef_old'] = dice_coefs_old
output_df['dice_coef_new'] = dice_coefs_new
output_df.head()
```

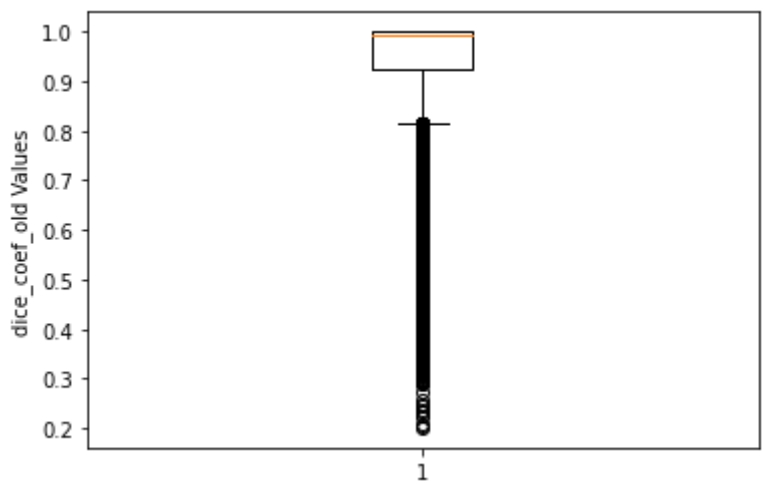
Out[61]:

	ImagePath	MaskPath	dice_coef_old	dice_coef_new
0	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...	0.999505	0.980972
1	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...	0.438927	0.480529
2	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...	0.998065	0.993456
3	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...	0.780020	0.886426
4	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...	0.426541	0.436540

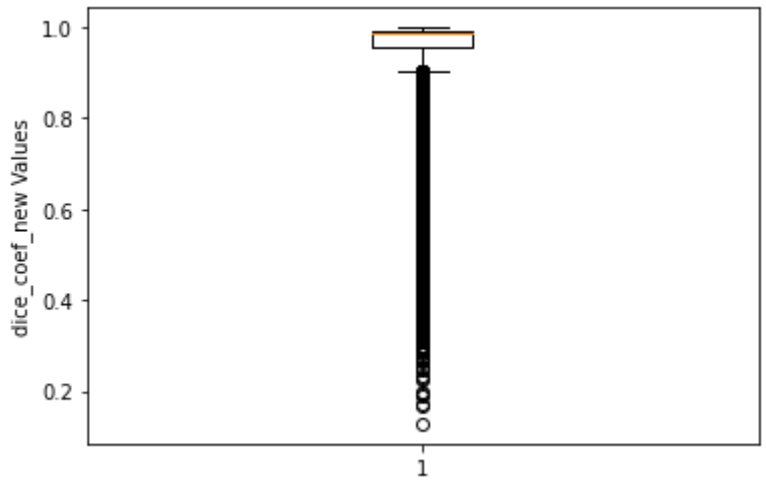
In [ ]:
```
print(output_df['dice_coef_old'].describe())
print('*'*25)
print(output_df['dice_coef_new'].describe())
```

```
count    12047.000000
mean         0.927662
std          0.131354
min          0.201312
25%          0.925691
50%          0.991132
75%          0.999310
max          0.999834
Name: dice_coef_old, dtype: float64
*************************
count    12047.000000
mean         0.936873
std          0.117173
min          0.128267
25%          0.956147
50%          0.985283
75%          0.991909
max          0.999309
Name: dice_coef_new, dtype: float64
```

In [ ]:
```
plt.boxplot(dice_coefs_old)
plt.ylabel("dice_coef_old Values")
plt.show()
```



In [ ]:
```
plt.boxplot(dice_coefs_new)
plt.ylabel("dice_coef_new Values")
plt.show()
```



Normal Metric

In [46]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

In [47]:
```
smooth = 1.
def dice_coef(y_true, y_pred):
    y_true = tf.keras.layers.Flatten()(y_true)
    y_pred = tf.keras.layers.Flatten()(y_pred)
    intersection = tf.reduce_sum(y_true * y_pred)
    return (2. * intersection + smooth) / (tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) + smooth)

def get_dice_for_each_image(final_data):
    dice_coefs = []
    for i in tqdm(range(len(final_data))):
        img = final_data['ImagePath'].values[i]
        msk = final_data['MaskPath'].values[i]
        image_orig, image_mask_orig, predict_mask = Predict(img, msk)
        image_mask_orig = np.expand_dims(image_mask_orig, axis=0)
        image_mask_orig = Concatenate(axis=0)([image_mask_orig, image_mask_orig])
        y_true = K.flatten(image_mask_orig)
        y_pred = K.flatten(predict_mask)
        d_c = dice_coef(y_true, y_pred)
        # print(d_c)
        dice_coefs.append(d_c)
    print("Done")
    return dice_coefs
```

In [51]:
```
model.load_weights( '/content/drive/MyDrive/27_Case_study_2/best_Double_Unet.hdf5')
print('loaded saved model {best_Double_Unet.hdf5}')
dice_coefs_old = get_dice_for_each_image(X_val)
dice_coefs_old = [dc.numpy() for dc in dice_coefs_old]
model.load_weights( '/content/drive/MyDrive/27_Case_study_2/best_Double_Unet_weighted_metric.hdf5')
print('loaded saved model {best_Double_Unet_weighted_metric.hdf5}')
dice_coefs_new = get_dice_for_each_image(X_val)
dice_coefs_new = [dc.numpy() for dc in dice_coefs_new]
```

```
  0%|          | 0/1808 [00:00<?, ?it/s]
loaded saved model {best_Double_Unet.hdf5}
100%|██████████| 1808/1808 [02:16<00:00, 13.20it/s]
Done
  0%|          | 2/1808 [00:00<02:15, 13.34it/s]
loaded saved model {best_Double_Unet_weighted_metric.hdf5}
100%|██████████| 1808/1808 [02:10<00:00, 13.87it/s]
Done
```

In [52]:
```
output_df = X_val.copy()
output_df['dice_coef_old'] = dice_coefs_old
output_df['dice_coef_new'] = dice_coefs_new
output_df.to_csv('dice_coef_X_val.csv', index=False)
output_df.head()
```

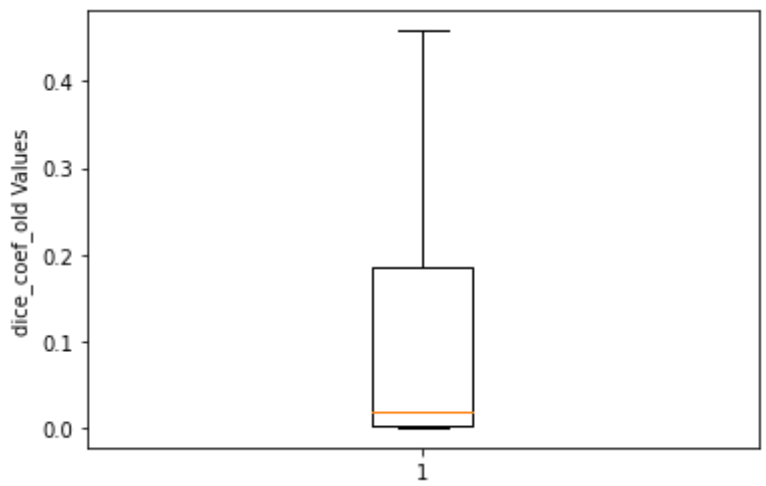
Out[52]:

	ImagePath	MaskPath	dice_coef_old	dice_coef_new
3842	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...	0.427148	0.024903
6110	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...	0.019273	0.011696
10079	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...	0.382858	0.025336
896	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...	0.010993	0.011360
8317	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...	0.057103	0.008682

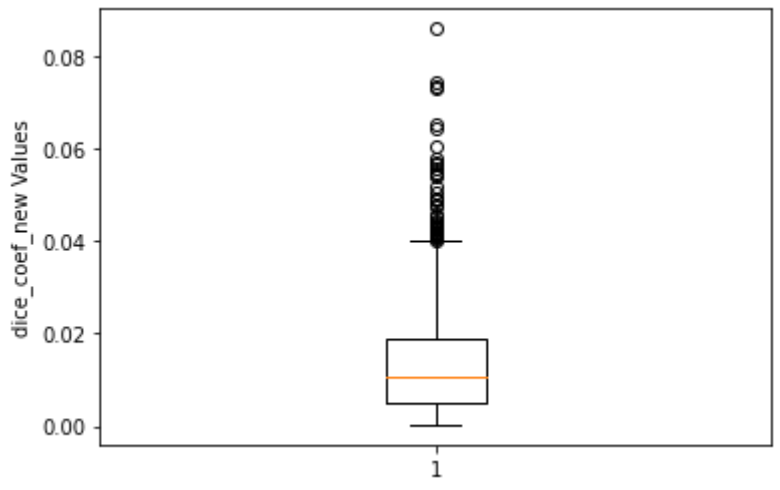
In [53]:
```
print(output_df['dice_coef_old'].describe())
print('*'*25)
print(output_df['dice_coef_new'].describe())
```

```
count    1808.000000
mean        0.100401
std         0.136088
min         0.000074
25%         0.002470
50%         0.018627
75%         0.185465
max         0.458416
Name: dice_coef_old, dtype: float64
*************************
count    1808.000000
mean        0.013343
std         0.010787
min         0.000255
25%         0.005048
50%         0.010702
75%         0.019079
max         0.086043
Name: dice_coef_new, dtype: float64
```

In [54]:
```
plt.boxplot(dice_coefs_old)
plt.ylabel("dice_coef_old Values")
plt.show()
```



In [55]:
```
plt.boxplot(dice_coefs_new)
plt.ylabel("dice_coef_new Values")
plt.show()
```



In [56]:
```
fore_ground = []
back_ground = []


for m in tqdm(X_val['MaskPath']):
    mask = read_mask(m)
    # print(type(mask))
    # print(mask.shape)
    f_mask = K.flatten(mask).numpy()
    zero_count = list(f_mask).count(0.0)
    back_ground.append(zero_count)
    one_count = list(f_mask).count(1.0)
    fore_ground.append(one_count)
    # print(list(f_mask).count(1.0))
    # break

val_pixel_df = pd.DataFrame({'fore_ground':fore_ground, 'back_ground':back_ground})
val_pixel_df.to_csv('val_pixel_count_df.csv', index=False)
val_pixel_df.head()
```

```
100%|██████████| 1808/1808 [05:57<00:00,  5.06it/s]
```

Out[56]:

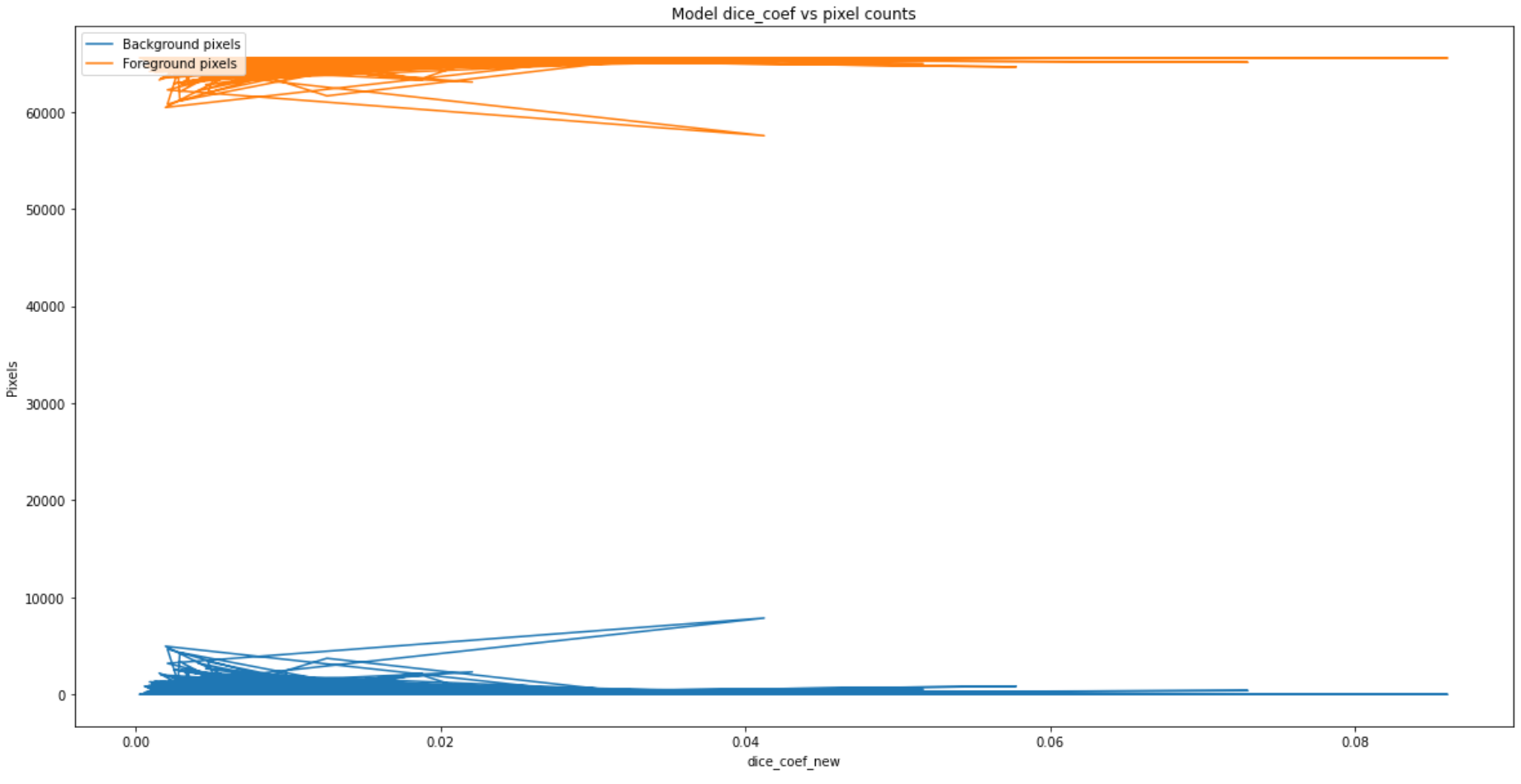
	fore_ground	back_ground
0	0	65536
1	0	65536
2	0	65536
3	0	65536
4	0	65536

In [57]:
```
val_pixel_df = pd.read_csv('val_pixel_count_df.csv')
print(val_pixel_df.columns)
val_dice_df = pd.read_csv('dice_coef_X_val.csv')
print(val_dice_df.columns)
```

```
Index(['fore_ground', 'back_ground'], dtype='object')
Index(['ImagePath', 'MaskPath', 'dice_coef_old', 'dice_coef_new'], dtype='object')
```

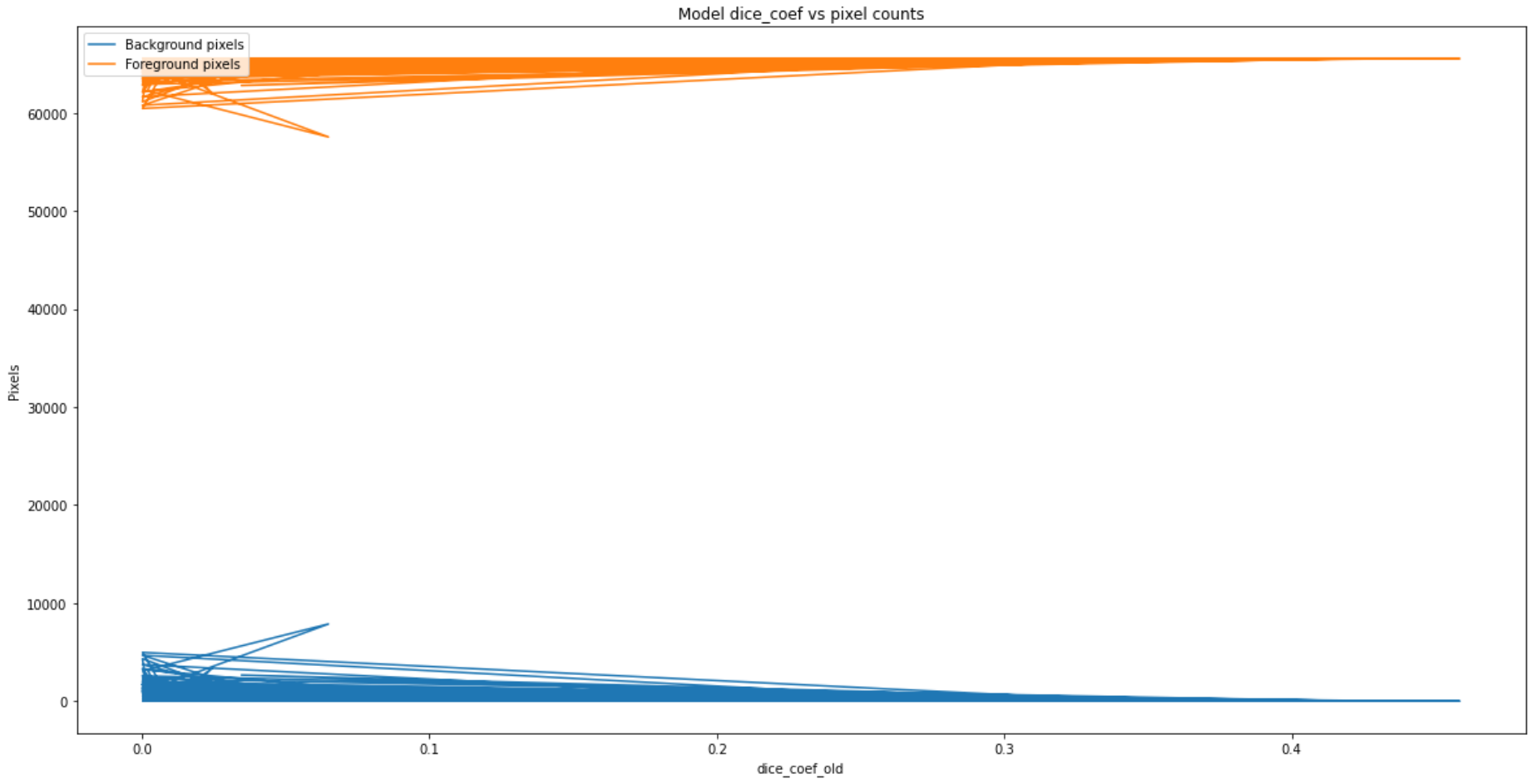
In [71]:
```
plt.figure(figsize=(20, 10))
# plt.subplot(121)
plt.plot(val_dice_df['dice_coef_new'], val_pixel_df['fore_ground'])
# plt.subplot(122)
plt.plot(val_dice_df['dice_coef_new'], val_pixel_df['back_ground'])
plt.title('Model dice_coef vs pixel counts')
plt.ylabel('Pixels')
plt.xlabel('dice_coef_new')
plt.legend(['Background pixels', 'Foreground pixels'], loc='upper left')
```

Out[71]: <matplotlib.legend.Legend at 0x7f9067876a58>



In [72]:
```
plt.figure(figsize=(20, 10))
# plt.subplot(121)
plt.plot(val_dice_df['dice_coef_old'], val_pixel_df['fore_ground'])
# plt.subplot(122)
plt.plot(val_dice_df['dice_coef_old'], val_pixel_df['back_ground'])
plt.title('Model dice_coef vs pixel counts')
plt.ylabel('Pixels')
plt.xlabel('dice_coef_old')
plt.legend(['Background pixels', 'Foreground pixels'], loc='upper left')
```

Out[72]: <matplotlib.legend.Legend at 0x7f90677669e8>



This shows that lower the foreground pixels higher is the dice score.

Observation:

From above information and plots, I can see the model trained previously i.e. {best_Double_Unet.hdf5} is performing better than the later one.

The average dice coefficient for this model is 0.100401 and the average dice coefficient for other model is 0.013343. And this can be seen in the image plots I have plotted above for both models

These are still low scores but we can train it for more epochs so as to get better results like 300, 500 or 1000 epochs. I have trained the model only for 30 epochs due to time constraints