# Image Segmentation of Artificial Lunar Landscape Images



# Introduction

In this Project we going to use U-net Neural network architecture which is the best NN structure for the purpose of Image segmentation. This NN will be fed with an Artificial Lunar Landscape dataset, to segment the lunar surface to better predict the route a lunar rover should take to avoid system failure or accidents. This objective is very crucial for a space rover as it is a sensitive and expensive device on foreign space land, there is no possibility of repairs, so the best course of action for anyone in this situation is to avoid the failure in the first place. So for a rover, “ **Precaution is better than cure”,** is the only option.

This is a Deep learning - computer vision project, where I have used techniques like Transfer Learning, Callbacks, U-net model building and optimization from Vgg-16 Base as a backbone, Technical Documentation and plotting are done using matplotlib.

### I hope everyone gains value from this project, Happy Learning.

## 

## 

## We will be using iou score as metric to guage the performance of the model so what is Iou Score?

The Intersecction over Union(IoU) metric, also referred to as the jaccard index, is essentially a method to quantify the percent overlap between the target mask and our prediction output. This metric is closely related to the Dice Co-efficient which is often used as a loss function during training.

Quite simply, the IoU metric measures the number of pixels common between the target and prediction masks divided by the total number of pixels present across both masks.

## Our goal is to increase the val\_iou\_score as much as we can for this project using any method. The evaluation of our model will be based on the acquired val\_iou\_score.

Iou score on validation data is more valuable to us for obvious reason that model is testing it’s capabilities that it learned by trainininng on the training data that we fed to it, so it already has seen that data so its like a open book exam for it if we consider the iou score only for judging the performance and that will be not very practical.

### Some ways to increase the performance

* Increase the number of epochs
* Using SOTA high performance networks with transfer learning
* Using callbacks and carefully observing your model performance

***step:1 import the necessary Library***

import segmentation\_models as sm

import glob

import cv2

import os

import numpy as np

from matplotlib import pyplot as plt

import tensorflow as tf

import keras

from keras.callbacks import EarlyStopping

from keras.models import Model,load\_model

from sklearn.model\_selection import train\_test\_split

* Provide environment variable SM\_FRAMEWORK=keras / SM\_FRAMEWORK=tf.keras before import segmentation\_models
* Change framework sm.set\_framework('keras') / sm.set\_framework('tf.keras')

step:2 Setting framework environment

os.environ["SM\_FRAMEWORK"] = "tf.keras"

sm.set\_framework('tf.keras')

keras.backend.set\_image\_data\_format('channels\_last')

## 

## Step:3 Establishing Data Preprocessing Pipeline

H = 480

W = 480

*'''This function is used to return the list of path for images and masks in sorted order from the given directory respectively.'''*

*# function to return list of image paths and mask paths*

def process\_data(IMG\_DIR, MASK\_DIR):

images = [os.path.join(IMG\_DIR, x) for x **in** sorted(os.listdir(IMG\_DIR))]

masks = [os.path.join(MASK\_DIR, x) for x **in** sorted(os.listdir(MASK\_DIR))]

return images, masks

*'''This function is used to return splitted list of images and corresponding*

*mask paths in train and test by providing test size.'''*

*# function to load data and train test split*

def load\_data(IMG\_DIR, MASK\_DIR):

X, y = process\_data(IMG\_DIR, MASK\_DIR)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42)

return X\_train, X\_test, y\_train, y\_test

*'''This function is used to read images. It takes image path as input.*

*After reading image it is resized by width and height provide above(480 x 480).*

*Next normalization is done by dividing each values with 255. And the result is returned.'''*

*# function to read image*

def read\_image(x):

x = cv2.imread(x, cv2.IMREAD\_COLOR)

x = cv2.resize(x, (W, H))

x = x / 255.0

x = x.astype(np.float32)

return x

*'''This function is used to read masks.'''*

*# function to read mask*

def read\_mask(x):

x = cv2.imread(x, cv2.IMREAD\_GRAYSCALE)

x = cv2.resize(x, (W, H))

x = x.astype(np.int32)

return x

*'''This function is used to generate tensorflow data pipeline.*

*The tensorflow data pipeline is mapped to function ‘preprocess’ .'''*

*# function for tensorflow dataset pipeline*

def tf\_dataset(x, y, batch=8):

dataset = tf.data.Dataset.from\_tensor\_slices((x, y))

dataset = dataset.shuffle(buffer\_size=5000)

dataset = dataset.map(preprocess)

dataset = dataset.batch(batch)

dataset = dataset.repeat()

dataset = dataset.prefetch(2)

return dataset

*'''This function takes image and mask path.*

*It reads the image and mask as provided by paths.*

*Mask is one hot encoded for multi class segmentation (here 4 class).'''*

*# function to read image and mask amd create one hot encoding for mask*

def preprocess(x, y):

def f(x, y):

x = x.decode()

y = y.decode()

image = read\_image(x)

mask = read\_mask(y)

return image, mask

image, mask = tf.numpy\_function(f, [x, y], [tf.float32, tf.int32])

mask = tf.one\_hot(mask, 4, dtype=tf.int32)

image.set\_shape([H, W, 3])

mask.set\_shape([H, W, 4])

return image, mask

## 

## Step:4 Load the dataset

*'''RENDER\_IMAGE\_DIR\_PATH: ‘Path of image directory’*

*GROUND\_MASK\_DIR\_PATH: ‘Path of mask directory’*

*Here load\_data function is called. This will load the dataset paths and*

*split it into X\_train, X\_test, y\_train, y\_test '''*

RENDER\_IMAGE\_DIR\_PATH = '../input/artificial-lunar-rocky-landscape-dataset/images/render'

GROUND\_MASK\_DIR\_PATH = '../input/artificial-lunar-rocky-landscape-dataset/images/clean'

X\_train, X\_test, y\_train, y\_test = load\_data(RENDER\_IMAGE\_DIR\_PATH, GROUND\_MASK\_DIR\_PATH)

**Let’s see how our dataset is distributed**

print(f"Dataset:**\n** Train: **{**len(X\_train)**}** **\n** Test: **{**len(X\_test)**}**")

Dataset:

Train: 7812

Test: 1954

## Step:5 Generate tensorflow data pipeline

In [61]:

batch\_size = 16 *#batch size of 8 was given by default, I have increased it to 16 due to some results of reseasrch papers showing increase in batcch size increases the performance of the model. Also, if we increase batch size to some the maximum number that is 1 image per batch then there will be the case of over-fitting. reason for explaining this is that, there is a sweet spot of batch size that should be looked for by experimentations and will always be specific to that dataset only. there is no single batch size number which is best for every dataset.*

*'''Here the tf\_dataset function is called will generate the tensorflow data pipeline.'''*

*# calling tf\_dataset*

train\_dataset = tf\_dataset(X\_train, y\_train, batch=batch\_size)

valid\_dataset = tf\_dataset(X\_test, y\_test, batch=batch\_size)

## 

## 

## Creating U-net Architecture

Here we are going to use transfer learning to build U-net architecture using Vgg-16 backbone.

Transfer Leranring: Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks.

*We can implement unet architecture using the sequential API too.But that was not very efficient and got very un-satisfying results. Thus here i have implemented the Transfer learning technique using VGG-16 to build the unet architecture from the pre-trained vgg model.*

*#### Step 1: Creating a base model*

IMG\_SHAPE = (480, 480, 3)

*# include\_top specify that we don't want to use the top layer (classifier)*

base\_model = tf.keras.applications.VGG16(input\_shape=IMG\_SHAPE,

include\_top=False,

weights='imagenet')

*#### Step 2: Freezing the base*

*# It is important to freeze the convolutional base before you compile and train the model.*

*# Freezing prevents the weights in a given layer from being updated during training*

*# VGG16 has many layers, so setting the entire model's trainable flag to False will freeze all of them.*

base\_model.trainable = False

*# Let's take a look at the base model architecture*

base\_model.summary()

*#### Step 3: Adding the head*

*# inputs*

inputs = tf.keras.Input(shape=(480, 480, 3))

*# base with pretrained model*

x = base\_model(inputs, training=False)

*# head layers*

x = tf.keras.layers.GlobalAveragePooling2D()(x)

x = tf.keras.layers.Dropout(0.2)(x)

outputs = tf.keras.layers.Dense(2)(x)

*# model*

model = tf.keras.Model(inputs, outputs)

*# Let's take a look at the final model architecture*

model.summary()

Model: "vgg16"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

input\_4 (InputLayer) [(None, 480, 480, 3)] 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block1\_conv1 (Conv2D) (None, 480, 480, 64) 1792

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block1\_conv2 (Conv2D) (None, 480, 480, 64) 36928

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block1\_pool (MaxPooling2D) (None, 240, 240, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block2\_conv1 (Conv2D) (None, 240, 240, 128) 73856

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block2\_conv2 (Conv2D) (None, 240, 240, 128) 147584

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block2\_pool (MaxPooling2D) (None, 120, 120, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_conv1 (Conv2D) (None, 120, 120, 256) 295168

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_conv2 (Conv2D) (None, 120, 120, 256) 590080

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_conv3 (Conv2D) (None, 120, 120, 256) 590080

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_pool (MaxPooling2D) (None, 60, 60, 256) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv1 (Conv2D) (None, 60, 60, 512) 1180160

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv2 (Conv2D) (None, 60, 60, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv3 (Conv2D) (None, 60, 60, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_pool (MaxPooling2D) (None, 30, 30, 512) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv1 (Conv2D) (None, 30, 30, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv2 (Conv2D) (None, 30, 30, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv3 (Conv2D) (None, 30, 30, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_pool (MaxPooling2D) (None, 15, 15, 512) 0

=================================================================

Total params: 14,714,688

Trainable params: 0

Non-trainable params: 14,714,688

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Model: "model\_2"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

input\_5 (InputLayer) [(None, 480, 480, 3)] 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

vgg16 (Functional) (None, 15, 15, 512) 14714688

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

global\_average\_pooling2d\_1 ( (None, 512) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 512) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 2) 1026

=================================================================

Total params: 14,715,714

Trainable params: 1,026

Non-trainable params: 14,714,688

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*# using segmentation\_models to create U-net with vgg16 as a backbone*

*# and pretrained imagenet weights*

In [63]:

BACKBONE = 'vgg16'

input\_shape = (480, 480, 3)

n\_classes = 4

activation = 'softmax'

*# segmentation\_model basically will create a mirror image of our backbone as expansion path and add to the contraction path*

model = sm.Unet(backbone\_name = BACKBONE,

input\_shape = input\_shape,

classes = n\_classes,

activation = activation,

encoder\_weights = 'imagenet')

model.summary()

Model: "model\_3"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # Connected to

==================================================================================================

input\_6 (InputLayer) [(None, 480, 480, 3) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block1\_conv1 (Conv2D) (None, 480, 480, 64) 1792 input\_6[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block1\_conv2 (Conv2D) (None, 480, 480, 64) 36928 block1\_conv1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block1\_pool (MaxPooling2D) (None, 240, 240, 64) 0 block1\_conv2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block2\_conv1 (Conv2D) (None, 240, 240, 128 73856 block1\_pool[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block2\_conv2 (Conv2D) (None, 240, 240, 128 147584 block2\_conv1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block2\_pool (MaxPooling2D) (None, 120, 120, 128 0 block2\_conv2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_conv1 (Conv2D) (None, 120, 120, 256 295168 block2\_pool[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_conv2 (Conv2D) (None, 120, 120, 256 590080 block3\_conv1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_conv3 (Conv2D) (None, 120, 120, 256 590080 block3\_conv2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_pool (MaxPooling2D) (None, 60, 60, 256) 0 block3\_conv3[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv1 (Conv2D) (None, 60, 60, 512) 1180160 block3\_pool[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv2 (Conv2D) (None, 60, 60, 512) 2359808 block4\_conv1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv3 (Conv2D) (None, 60, 60, 512) 2359808 block4\_conv2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_pool (MaxPooling2D) (None, 30, 30, 512) 0 block4\_conv3[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv1 (Conv2D) (None, 30, 30, 512) 2359808 block4\_pool[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv2 (Conv2D) (None, 30, 30, 512) 2359808 block5\_conv1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv3 (Conv2D) (None, 30, 30, 512) 2359808 block5\_conv2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_pool (MaxPooling2D) (None, 15, 15, 512) 0 block5\_conv3[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

center\_block1\_conv (Conv2D) (None, 15, 15, 512) 2359296 block5\_pool[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

center\_block1\_bn (BatchNormaliz (None, 15, 15, 512) 2048 center\_block1\_conv[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

center\_block1\_relu (Activation) (None, 15, 15, 512) 0 center\_block1\_bn[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

center\_block2\_conv (Conv2D) (None, 15, 15, 512) 2359296 center\_block1\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

center\_block2\_bn (BatchNormaliz (None, 15, 15, 512) 2048 center\_block2\_conv[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

center\_block2\_relu (Activation) (None, 15, 15, 512) 0 center\_block2\_bn[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage0\_upsampling (UpSa (None, 30, 30, 512) 0 center\_block2\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage0\_concat (Concaten (None, 30, 30, 1024) 0 decoder\_stage0\_upsampling[0][0]

block5\_conv3[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage0a\_conv (Conv2D) (None, 30, 30, 256) 2359296 decoder\_stage0\_concat[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage0a\_bn (BatchNormal (None, 30, 30, 256) 1024 decoder\_stage0a\_conv[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage0a\_relu (Activatio (None, 30, 30, 256) 0 decoder\_stage0a\_bn[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage0b\_conv (Conv2D) (None, 30, 30, 256) 589824 decoder\_stage0a\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage0b\_bn (BatchNormal (None, 30, 30, 256) 1024 decoder\_stage0b\_conv[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage0b\_relu (Activatio (None, 30, 30, 256) 0 decoder\_stage0b\_bn[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage1\_upsampling (UpSa (None, 60, 60, 256) 0 decoder\_stage0b\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage1\_concat (Concaten (None, 60, 60, 768) 0 decoder\_stage1\_upsampling[0][0]

block4\_conv3[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage1a\_conv (Conv2D) (None, 60, 60, 128) 884736 decoder\_stage1\_concat[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage1a\_bn (BatchNormal (None, 60, 60, 128) 512 decoder\_stage1a\_conv[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage1a\_relu (Activatio (None, 60, 60, 128) 0 decoder\_stage1a\_bn[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage1b\_conv (Conv2D) (None, 60, 60, 128) 147456 decoder\_stage1a\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage1b\_bn (BatchNormal (None, 60, 60, 128) 512 decoder\_stage1b\_conv[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage1b\_relu (Activatio (None, 60, 60, 128) 0 decoder\_stage1b\_bn[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage2\_upsampling (UpSa (None, 120, 120, 128 0 decoder\_stage1b\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage2\_concat (Concaten (None, 120, 120, 384 0 decoder\_stage2\_upsampling[0][0]

block3\_conv3[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage2a\_conv (Conv2D) (None, 120, 120, 64) 221184 decoder\_stage2\_concat[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage2a\_bn (BatchNormal (None, 120, 120, 64) 256 decoder\_stage2a\_conv[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage2a\_relu (Activatio (None, 120, 120, 64) 0 decoder\_stage2a\_bn[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage2b\_conv (Conv2D) (None, 120, 120, 64) 36864 decoder\_stage2a\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage2b\_bn (BatchNormal (None, 120, 120, 64) 256 decoder\_stage2b\_conv[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage2b\_relu (Activatio (None, 120, 120, 64) 0 decoder\_stage2b\_bn[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage3\_upsampling (UpSa (None, 240, 240, 64) 0 decoder\_stage2b\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage3\_concat (Concaten (None, 240, 240, 192 0 decoder\_stage3\_upsampling[0][0]

block2\_conv2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage3a\_conv (Conv2D) (None, 240, 240, 32) 55296 decoder\_stage3\_concat[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage3a\_bn (BatchNormal (None, 240, 240, 32) 128 decoder\_stage3a\_conv[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage3a\_relu (Activatio (None, 240, 240, 32) 0 decoder\_stage3a\_bn[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage3b\_conv (Conv2D) (None, 240, 240, 32) 9216 decoder\_stage3a\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage3b\_bn (BatchNormal (None, 240, 240, 32) 128 decoder\_stage3b\_conv[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage3b\_relu (Activatio (None, 240, 240, 32) 0 decoder\_stage3b\_bn[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage4\_upsampling (UpSa (None, 480, 480, 32) 0 decoder\_stage3b\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage4a\_conv (Conv2D) (None, 480, 480, 16) 4608 decoder\_stage4\_upsampling[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage4a\_bn (BatchNormal (None, 480, 480, 16) 64 decoder\_stage4a\_conv[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage4a\_relu (Activatio (None, 480, 480, 16) 0 decoder\_stage4a\_bn[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage4b\_conv (Conv2D) (None, 480, 480, 16) 2304 decoder\_stage4a\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage4b\_bn (BatchNormal (None, 480, 480, 16) 64 decoder\_stage4b\_conv[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder\_stage4b\_relu (Activatio (None, 480, 480, 16) 0 decoder\_stage4b\_bn[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

final\_conv (Conv2D) (None, 480, 480, 4) 580 decoder\_stage4b\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

softmax (Activation) (None, 480, 480, 4) 0 final\_conv[0][0]

==================================================================================================

Total params: 23,752,708

Trainable params: 23,748,676

Non-trainable params: 4,032

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 

## 

## 

## 

## 

## Step:6 Load model and compile

In upcoming cells you will see you of many callbacks like ModelCheckpoint, ReducedLROnPlateau, and earlyStopping. These callbacks are used for the purpose of training our model efficiently. Callbacks control parameters like learning rate or stopping the training process altogether if performance of the model does not increase and beauty of callbacks they control the model during the training run automatically as per the arguments provided by us.

For our model, I have used IOU score for guaging the performance. IOU score is between 0 and 1, Higher the score better is the performance. For these perpuose we will monitor Val\_iou score to see if our model is estimating as per our expectation.

In [64]:

*# importing libraries*

from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping *#importing the necessary modules for callbacks*

from segmentation\_models.metrics import iou\_score

import datetime, os

*""" Defining Hyperparameters """*

img\_shape = (480, 480, 3) *#input shapes, default one with notebook (256,256,3) to presently this (480,480,3). This is done to accomadate the height and width increase in the input image that we have changed in the previous cells above.*

num\_classes = 4

lr = 1e-5 *#decreased learning rate gives better generalization*

batch\_size = 16 *#increasing the batch size can provide better performance*

epochs = 30 *#Increased epochs give better scores*

*""" Model building and compiling """*

*# metrics for result validation*

metrics = [sm.metrics.IOUScore(threshold=0.5), sm.metrics.FScore(threshold=0.5)]

*# compiling the model*

model.compile(loss = 'categorical\_crossentropy',

optimizer = tf.keras.optimizers.Adam(lr, epsilon=1e-8, decay=1e-6), *#epsilon and decay are added as a way to increase the performance. epsilon is a very small number to prevent any division by zero in the implementation. decay helps us to increase performance by reducing the momentum of the optimizer. Adam uses Momentum and Adaptive Learning Rates to converge faster.*

metrics = metrics)

train\_steps = len(X\_train)//batch\_size

valid\_steps = len(X\_test)//batch\_size

*""" Callbacks """*

*#A callback is an object that can perform actions at various stages of training (e.g. at the start or end of an epoch, before or after a single batch, etc). You can use callbacks to: Write TensorBoard logs after every batch of training to monitor your metrics, Periodically save your model to disk, Do early stopping, Get a view on internal states and statistics of a model during training.*

current\_datetime = datetime.datetime.now().strftime("%Y%m**%d**-%H%M%S")

callbacks = [

tf.keras.callbacks.ModelCheckpoint(filepath=f'models/LunarModel.h5', monitor='val\_iou\_score', verbose=1, mode='max', save\_best\_only=True), *# Create a callback that saves the model periodically as training moves along the number of epochs.*

tf.keras.callbacks.ReduceLROnPlateau(monitor="val\_iou\_score", mode='max', patience=4, factor=0.1, verbose=1, min\_lr=1e-6), *#This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.*

tf.keras.callbacks.EarlyStopping(monitor="val\_iou\_score", patience=5, verbose=1, mode='max'), *#Stop training when a monitored metric has stopped improving.*

]

## 

## Step:7 Training the model

In [65]:

*'''model.fit is used to train the model'''*

model\_history = model.fit(train\_dataset,

steps\_per\_epoch=train\_steps,

validation\_data=valid\_dataset,

validation\_steps=valid\_steps,

epochs=epochs,

callbacks=callbacks

)

## Step:8 Using our model for segmentation of rocks and boulders in some of the unknown test cases of Lunar Surface images.

from skimage.io import imread

from skimage.transform import resize

*# function to predict result*

def predict\_image(img\_path, mask\_path, model):

H = 480

W = 480

num\_classes = 4

img = imread(img\_path)

img = img[:480, :480, :]

img = img / 255.0

img = img.astype(np.float32)

*## Read mask*

mask = imread(mask\_path, as\_gray = True)

mask = mask[:480, :480]

*## Prediction*

pred\_mask = model.predict(np.expand\_dims(img, axis=0))

pred\_mask = np.argmax(pred\_mask, axis=-1)

pred\_mask = pred\_mask[0]

*# calculating IOU score*

inter = np.logical\_and(mask, pred\_mask)

union = np.logical\_or(mask, pred\_mask)

iou = inter.sum() / union.sum()

return img, mask, pred\_mask, iou

img\_path = '../input/artificial-lunar-rocky-landscape-dataset/images/render/render0042.png'

mask\_path = '../input/artificial-lunar-rocky-landscape-dataset/images/clean/clean0042.png'

img, mask, pred\_mask, iou = predict\_image(img\_path, mask\_path, model)

fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize = (15, 10))

ax1.set\_title("Input Image")

ax1.imshow(img)

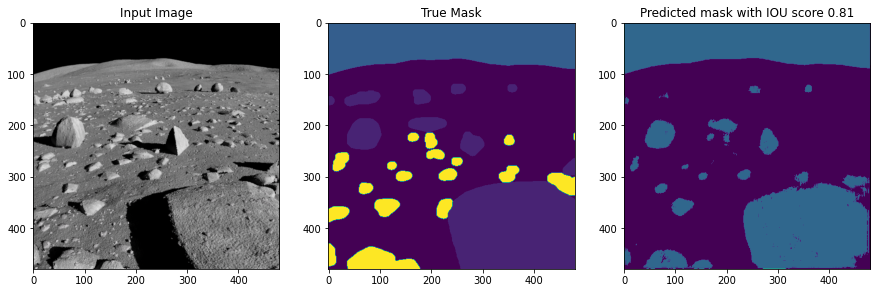
ax2.set\_title("True Mask")

ax2.imshow(mask)

ax3.set\_title("Predicted mask with IOU score **%.2f**"%(iou))

ax3.imshow(pred\_mask)

plt.show()



# 

# 

# Step:8 Documentation and plottings

*# visulization code*

import matplotlib.pyplot as plt

def Plotting(history1, history2):

plt.plot(history1['val\_iou\_score'])

plt.plot(history2['val\_iou\_score'])

plt.title('Model Performance')

plt.xlabel('epochs')

plt.ylabel('val\_iou\_score')

plt.legend(['Val\_iou\_score of 1st observation', 'Val\_iou\_score of 2st observation'])

plt.show()

*# Code block to create tables in Python*

*# import module*

from tabulate import tabulate

def Table\_to\_compare(history1,history2):

*# assign data*

mydata = [

[history1, history2]

]

*# create header*

head = ["Before Improvement", "After Improvement"]

*# display table*

print(tabulate(mydata, headers=head, tablefmt="grid"))

Note: each Documentation is for a effective improvement during my experiments only, not every little twik i did was so good but i have added the summary table at the end of the article for all the things i tested so check it out to quich your thrust of curiosity.

# 

# 

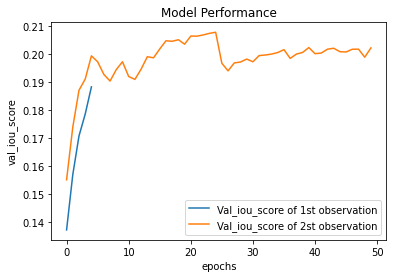
# 1. Plotting for impact of epoch increment on basic u-net's val\_iou\_score

In [72]:

Plotting(history\_1,history\_4)

print()

Table\_to\_compare(history\_1["val\_iou\_score"][-1],history\_4["val\_iou\_score"][-1])



+----------------------+---------------------+

| Before Improvement | After Improvement |

+======================+=====================+

| 0.188267 | 0.202188 |

+----------------------+---------------------+

From this comparison of two training runs 1st(history\_1) with 5 epochs and 2nd(history\_4) one under considration is with 50 epochs it can be seen that there is increase in val\_iou\_score slightly but it is there. I Didn't increased the number of epochs from 50 as you can see from the graph there is a no real benefit to increasinf the epochs anymore to improve performance

# 

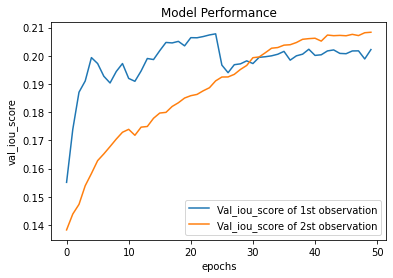
# 2. Plotting for impact of learning rate decrement on basic u-net's val\_iou\_score

In [73]:

Plotting(history\_4,history\_5)

print()

Table\_to\_compare(history\_4["val\_iou\_score"][-1],history\_5["val\_iou\_score"][-1])



+----------------------+---------------------+

| Before Improvement | After Improvement |

+======================+=====================+

| 0.202188 | 0.208336 |

+----------------------+---------------------+

As it is evident from the plot there is a stable increase with each epoch with lesser learning rate and also a slight increase in val\_iou\_score

# 

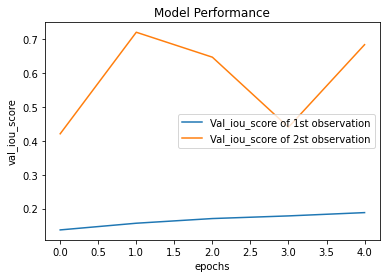
# 3. Plotting for impact of using transfer learning technique to build unet on val\_iou\_score

In [74]:

Plotting(history\_1,history\_9)

print()

Table\_to\_compare(history\_1["val\_iou\_score"][-1],history\_9["val\_iou\_score"][-1])



+----------------------+---------------------+

| Before Improvement | After Improvement |

+======================+=====================+

| 0.188267 | 0.684124 |

+----------------------+---------------------+

There is great boost in performance by using transfer learning technique to build u-net from a base model Vgg-16, transfer learning is so beneficial that even from the 1st epoch we get a considerable amount of performance as compared to just basic u-net model trained on our dataset.

# 

# 4. Plotting to show impact of inceasing epochs on our updated model

Improvements that are included in the training run under consideration

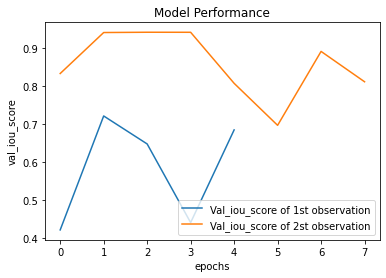
increased to epochs = 30 but stopped after 8 epochs ,Some parameters that i specified in adam optimizer after internet research epsilon=1e-8, decay=1e-6. Also used ReducedLRonplateau with min lr = 1e-6 and reduction in lr by factor 0.1 and patience = 5. other than these changes everything is same as previous experiment

In [75]:

Plotting(history\_9,history\_10)

print()

Table\_to\_compare(history\_9["val\_iou\_score"][-1],history\_10["val\_iou\_score"][-1])



+----------------------+---------------------+

| Before Improvement | After Improvement |

+======================+=====================+

| 0.684124 | 0.810828 |

+----------------------+---------------------+

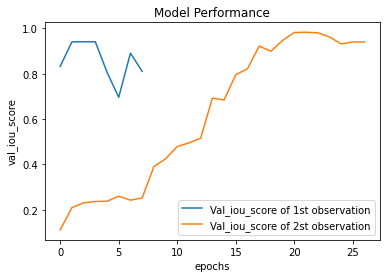
# 5. Plotting to show impact of Decreasing learning rate on our updated model in 11th training run

In [76]:

Plotting(history\_10,history\_11)

print()

Table\_to\_compare(history\_10["val\_iou\_score"][-1],history\_11["val\_iou\_score"][-1])



+----------------------+---------------------+

| Before Improvement | After Improvement |

+======================+=====================+

| 0.810828 | 0.939933 |

+----------------------+---------------------+

overall increase in val\_iou score with steady increase in performance every epoch. We can infer from this that with higher learning rate we get unstable model and improvement over epoch is lost because even after 5 epochs there was not a little bit of increase in val\_iou score, But on the other hand, we see that with less learning rate we are obtaining a steady growth over every passing epoch which is desirable.

# 

# 

# 

# 

# 

# 

# FINAL IMPROVEMENT SUMMARY TABLE

| Experiment# | Description | Increase in val\_iou from | Increase in val\_iou to (considering val\_iou score of final epoch of the corresponding training run) |
| --- | --- | --- | --- |
| 1 | default notebook with no improvements | NA | 0.1882 |
| 2 | Increased epoch = 5, which was given in the deafult notebook to epoch = 10 | 0.1882 | 0.2052 |
| 3 | Increased epoch = 10 to epoch = 20 | 0.2052 | 0.2058 |
| 4 | Increased epoch = 20 to epoch = 50 | 0.2058 | 0.2021 |
| 5 | Kept the epoch = 50 as earlier training run but decreased the learning rate of lr = 1e-4 to lr = 1e-5 | 0.2021 | 0.2083 |
| 6 | Tried experimenting with batch size. I used lr = 1e-4, decreased the to batch\_size = 8 and used epochs = 30 | 0.2083 | 0.1902 |
| 7 | After intensive internet research and things taught to us in the sessions for optimization implemented some callbacks like earlystopping and cmodel checkpointing. Used paramters listed here lr = 7e-4 batch\_size = 4 epochs = 30(with early stopped at 13th epoch,because model failed to increase val\_iou after 13 epochs) | 0.1902 | 0.1777 |
| 8 | Just used some callbacks and carried out ad-hoc testing of results with the listed parameters lr = 7e-4 batch\_size = 16 epochs = 5. No real improvements in results | 0.1777 | 0.1894 |
| 9 | from here on out tranfer learning technique was implemented and tremendous performance increase was attained with the listed parameters, lr = 1e-4 batch\_size = 16 epochs = 5.Also, one major change i made in terms of data provided to the model is that i increased the Height and width of the image that is going to be fed to the model as shown in the here(H = 480, W = 480 instead of H,W = 256 that was givven to us in the default notebook), This was very necessary as lunar dataset had images of 720 \* 480 dimensions and as we downscale the H,W values, our model get less pixel data as compared to original image to train on. | 0.1894 | 0.6841 |
| 10 | increased to epochs = 30 but stopped after 8 epochs ,Some parameters that i specified in adam optimizer after internet research epsilon=1e-8, decay=1e-6. Also used ReducedLRonplateau with min lr = 1e-6 and reduction in lr by factor 0.1 and patience = 5. other than these changes everything is same as previous experiment | 0.6841 | 0.8108 |
| 11 | Just decreased the learning rate to lr = 1e-5,other than this change everything is same as previous experiment. During training i used epochs = 30 but stopped after 27 epochs due to early stopping | 0.8108 | 0.9399 |
| 12 | no changes were made from previous training run, wanted to be sure if model reproduces same result, which is true as there is difference in iou score of just 0.0001 | 0.9399 | 0.9401 |

# This is collection of model histories of all the experiments that I ran with some characteristics shown alongside with it corresponding to that specific training run.

# Thank you for checking out my Project Article, If you like it please Follow 😊 and do share your feedback and suggestions in the comments, I will love to hear from the community🙌