Homework 2: Image Segmentation

Introduction

Our focus was to build a model to segment pictures from the BipBip-Haricot dataset.
To tackle the problem we, firstly, implement an encoder-decoder structure using VGG16 as encoder with frozen weights trained on Imagenet dataset. This network required as input 256x256 properly resized images and produced as output 256x256 sized masks, which needed to be resized to match the original test image dimension, producing poor results.

Therefore we had to face two critical aspects:

1. The encoder-decoder structure behaves poorly.
2. Excessive resizing of images leads to mediocre performances over the test set.

In particular to solve the second issue we tried to train the network with an higher dimensional input. However we stumbled across a major inconvenience, due to a lack of computational power.

Network Design

We finally decided to use a version of the Unet; the scheme of the actual implementation is available at the end of the file.

Data Preparation

Training data were modified so as to match the dataset generation procedure exploited in lab notebooks, creating text files to split images into train and validation .
Input images were colored, rectangular and size-variable.
Validation sets were extracted randomly from this structure (pc=0.15), with fixed random seed in order to compare different experiments.
After trying to analyze just raw data, we chose to apply the following data augmentation techniques in order to achieve a better generalization and reduce overfitting:

- rotation range of 10°
- shift range and height shift equal to 10
- zoom range 0.3
- horizontal and vertical flip
- reflect fill mode

The key procedure to face the resize issue was to crop images into patches of size 256x256. In this way the original picture is divided into various patches and the network is trained upon all of them. Finally the output masks are obtained reassembling the mask patches according to the superposition rule of the specific images.

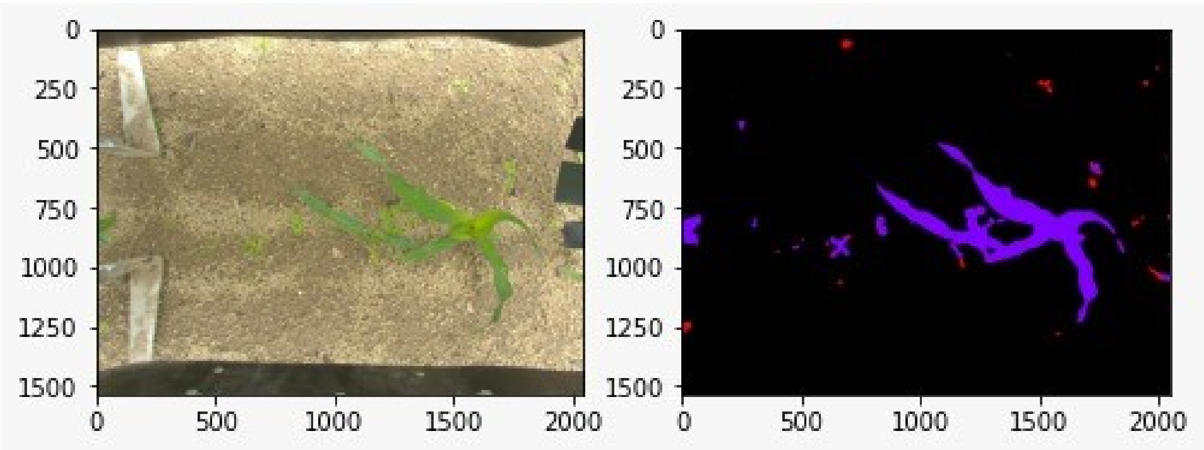
To avoid losing information related to specific shapes at borders, due to the cut, we decided to overlap patches , retaining in each patch pixels of its neighbours. During prediction, to deal with overlapped values we resort to a maximum rule: we choose the predicted class with highest label. Actually, if two superposed predictions were 0 (background) and 1 (weed) we would want to assign greater importance to the less likely label; the same holds for 1 and 2.

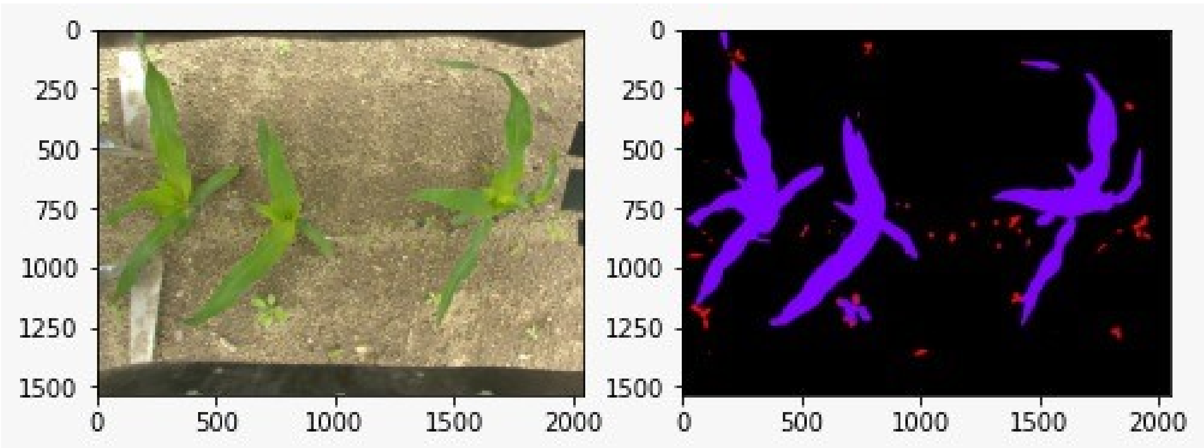
Loss

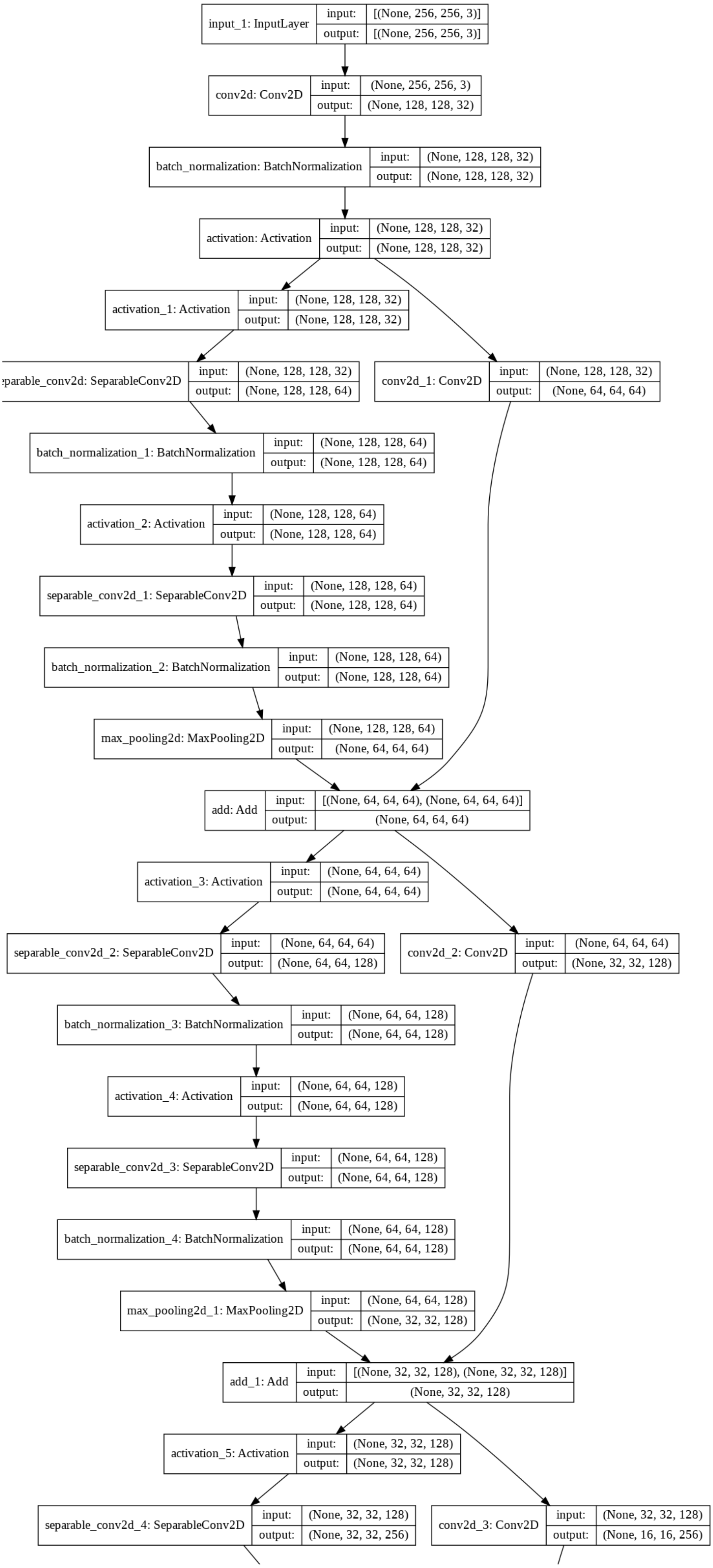
To deal with unbalanced classes we built a weighted loss.

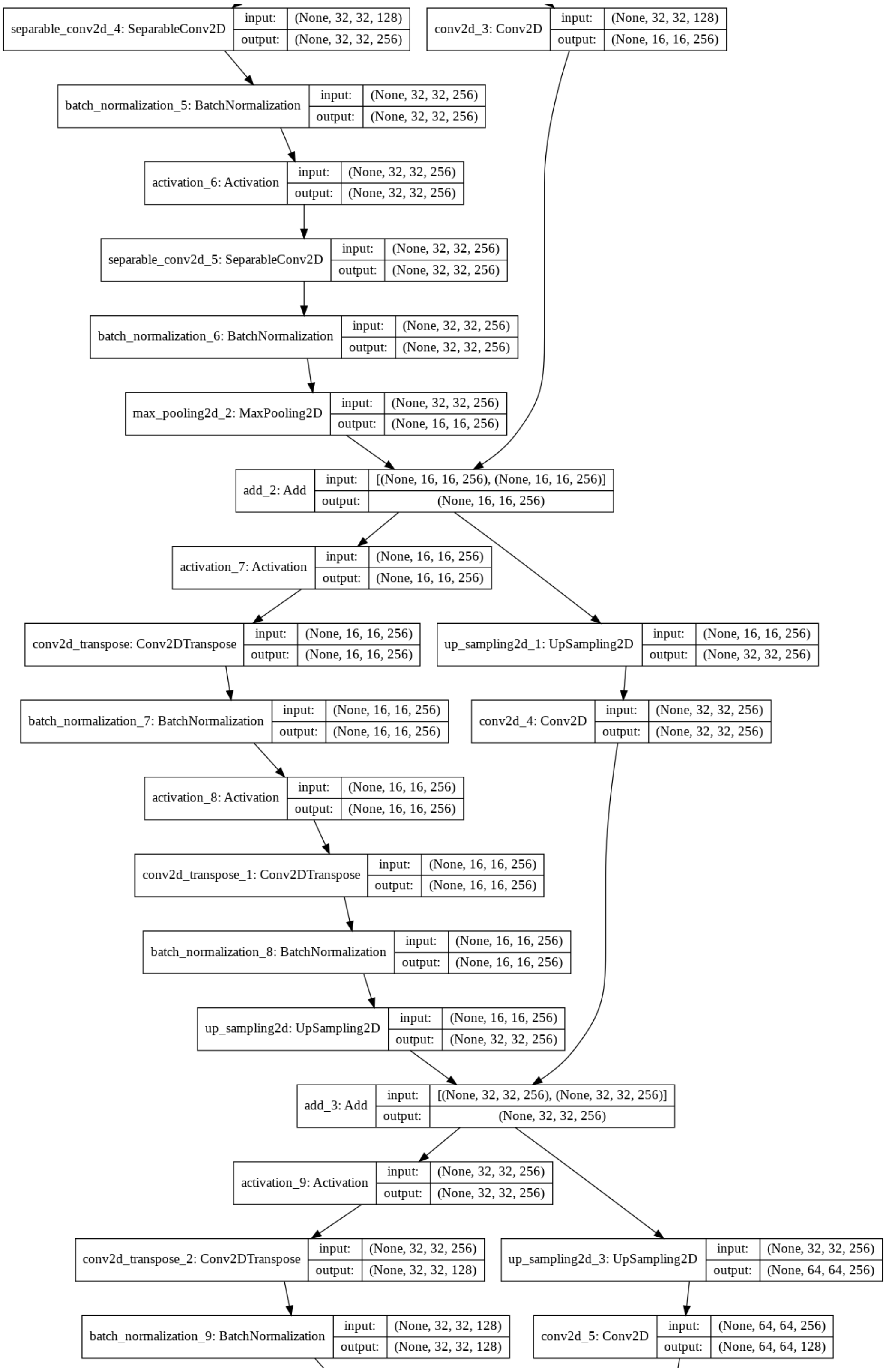
Conclusions

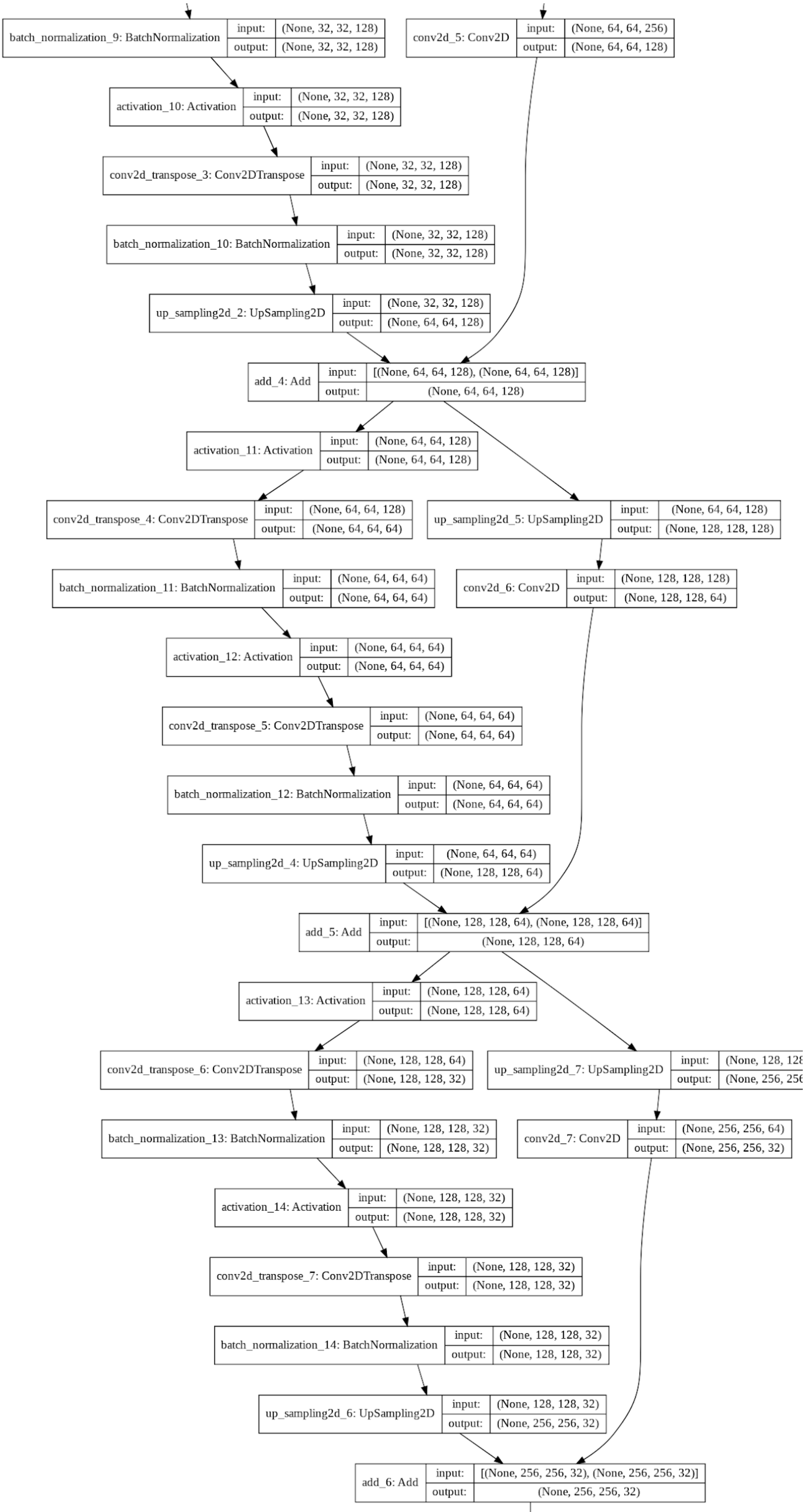
We are satisfied with the superposing patches method, which can be seen as the starting point of a more precise and organic analysis; for instance possible paths of research may be varying the crop size or the reassembling mechanism (maybe via smoothing splines…). Finally it's interesting to note that although the model was trained only on BipBip-Haricot dataset, it performed well enough also on BipBip-Mais and Weedelec datasets, showing a considerable level of generalization. Thus we can argue that training the model on the whole datasets (having proper resources) will likely lead to an efficiency improvement. Below, two obtained masks from the test set are attached.






input_1: InputLayer
input: [(None, 256, 256, 3)]
output: [(None, 256, 256, 3)]
conv2d: Conv2D
input: (None, 256, 256, 3)
output: (None, 128, 128, 32)
batch_normalization: BatchNormalization
input: (None, 128, 128, 32)
output: (None, 128, 128, 32)
activation: Activation
input: (None, 128, 128, 32)
output: (None, 128, 128, 32)
activation_1: Activation
input: (None, 128, 128, 32)
output: (None, 128, 128, 32)
eparable_conv2d: SeparableConv2D
input: (None, 128, 128, 32)
output: (None, 128, 128, 64)
conv2d_1: Conv2D
input: (None, 128, 128, 32)
output: (None, 64, 64, 64)
batch_normalization_1: BatchNormalization
input: (None, 128, 128, 64)
output: (None, 128, 128, 64)
activation_2: Activation
input: (None, 128, 128, 64)
output: (None, 128, 128, 64)
separable_conv2d_1: SeparableConv2D
input: (None, 128, 128, 64)
output: (None, 128, 128, 64)
batch_normalization_2: BatchNormalization
input: (None, 128, 128, 64)
output: (None, 128, 128, 64)
max_pooling2d: MaxPooling2D
input: (None, 128, 128, 64)
output: (None, 64, 64, 64)
add: Add
input: [(None, 64, 64, 64), (None, 64, 64, 64)]
output: (None, 64, 64, 64)
activation_3: Activation
input: (None, 64, 64, 64)
output: (None, 64, 64, 64)
separable_conv2d_2: SeparableConv2D
input: (None, 64, 64, 64)
output: (None, 64, 64, 128)
conv2d_2: Conv2D
input: (None, 64, 64, 64)
output: (None, 32, 32, 128)
batch_normalization_3: BatchNormalization
input: (None, 64, 64, 128)
output: (None, 64, 64, 128)
activation_4: Activation
input: (None, 64, 64, 128)
output: (None, 64, 64, 128)
separable_conv2d_3: SeparableConv2D
input: (None, 64, 64, 128)
output: (None, 64, 64, 128)
batch_normalization_4: BatchNormalization
input: (None, 64, 64, 128)
output: (None, 64, 64, 128)
max_pooling2d_1: MaxPooling2D
input: (None, 64, 64, 128)
output: (None, 32, 32, 128)
add_1: Add
input: [(None, 32, 32, 128), (None, 32, 32, 128)]
output: (None, 32, 32, 128)
activation_5: Activation
input: (None, 32, 32, 128)
output: (None, 32, 32, 128)
separable_conv2d_4: SeparableConv2D
input: (None, 32, 32, 128)
output: (None, 32, 32, 256)
conv2d_3: Conv2D
input: (None, 32, 32, 128)
output: (None, 16, 16, 256)


separable_conv2d_4: SeparableConv2D
input: (None, 32, 32, 128)
output: (None, 32, 32, 256)
conv2d_3: Conv2D
input: (None, 32, 32, 128)
output: (None, 16, 16, 256)
batch_normalization_5: BatchNormalization
input: (None, 32, 32, 256)
output: (None, 32, 32, 256)
activation_6: Activation
input: (None, 32, 32, 256)
output: (None, 32, 32, 256)
separable_conv2d_5: SeparableConv2D
input: (None, 32, 32, 256)
output: (None, 32, 32, 256)
batch_normalization_6: BatchNormalization
input: (None, 32, 32, 256)
output: (None, 32, 32, 256)
max_pooling2d_2: MaxPooling2D
input: (None, 32, 32, 256)
output: (None, 16, 16, 256)
add_2: Add
input: [(None, 16, 16, 256), (None, 16, 16, 256)]
output: (None, 16, 16, 256)
activation_7: Activation
input: (None, 16, 16, 256)
output: (None, 16, 16, 256)
conv2d_transpose: Conv2DTranspose
input: (None, 16, 16, 256)
output: (None, 16, 16, 256)
up_sampling2d_1: UpSampling2D
input: (None, 16, 16, 256)
output: (None, 32, 32, 256)
batch_normalization_7: BatchNormalization
input: (None, 16, 16, 256)
output: (None, 16, 16, 256)
conv2d_4: Conv2D
input: (None, 32, 32, 256)
output: (None, 32, 32, 256)
activation_8: Activation
input: (None, 16, 16, 256)
output: (None, 16, 16, 256)
conv2d_transpose_1: Conv2DTranspose
input: (None, 16, 16, 256)
output: (None, 16, 16, 256)
batch_normalization_8: BatchNormalization
input: (None, 16, 16, 256)
output: (None, 16, 16, 256)
up_sampling2d: UpSampling2D
input: (None, 16, 16, 256)
output: (None, 32, 32, 256)
add_3: Add
input: [(None, 32, 32, 256), (None, 32, 32, 256)]
output: (None, 32, 32, 256)
activation_9: Activation
input: (None, 32, 32, 256)
output: (None, 32, 32, 256)
conv2d_transpose_2: Conv2DTranspose
input: (None, 32, 32, 256)
output: (None, 32, 32, 128)
up_sampling2d_3: UpSampling2D
input: (None, 32, 32, 256)
output: (None, 64, 64, 256)
batch_normalization_9: BatchNormalization
input: (None, 32, 32, 128)
output: (None, 32, 32, 128)
conv2d_5: Conv2D
input: (None, 64, 64, 256)
output: (None, 64, 64, 128)



batch_normalization_9: BatchNormalization
input: (None, 32, 32, 128)
output: (None, 32, 32, 128)
conv2d_5: Conv2D
input: (None, 64, 64, 256)
output: (None, 64, 64, 128)
activation_10: Activation
input: (None, 32, 32, 128)
output: (None, 32, 32, 128)
conv2d_transpose_3: Conv2DTranspose
input: (None, 32, 32, 128)
output: (None, 32, 32, 128)
batch_normalization_10: BatchNormalization
input: (None, 32, 32, 128)
output: (None, 32, 32, 128)
up_sampling2d_2: UpSampling2D
input: (None, 32, 32, 128)
output: (None, 64, 64, 128)
add_4: Add
input: [(None, 64, 64, 128), (None, 64, 64, 128)]
output: (None, 64, 64, 128)
activation_11: Activation
input: (None, 64, 64, 128)
output: (None, 64, 64, 128)
conv2d_transpose_4: Conv2DTranspose
input: (None, 64, 64, 128)
output: (None, 64, 64, 64)
up_sampling2d_5: UpSampling2D
input: (None, 64, 64, 128)
output: (None, 128, 128, 128)
batch_normalization_11: BatchNormalization
input: (None, 64, 64, 64)
output: (None, 64, 64, 64)
conv2d_6: Conv2D
input: (None, 128, 128, 128)
output: (None, 128, 128, 64)
activation_12: Activation
input: (None, 64, 64, 64)
output: (None, 64, 64, 64)
conv2d_transpose_5: Conv2DTranspose
input: (None, 64, 64, 64)
output: (None, 64, 64, 64)
batch_normalization_12: BatchNormalization
input: (None, 64, 64, 64)
output: (None, 64, 64, 64)
up_sampling2d_4: UpSampling2D
input: (None, 64, 64, 64)
output: (None, 128, 128, 64)
add_5: Add
input: [(None, 128, 128, 64), (None, 128, 128, 64)]
output: (None, 128, 128, 64)
activation_13: Activation
input: (None, 128, 128, 64)
output: (None, 128, 128, 64)
conv2d_transpose_6: Conv2DTranspose
input: (None, 128, 128, 64)
output: (None, 128, 128, 32)
up_sampling2d_7: UpSampling2D
batch_normalization_13: BatchNormalization
input: (None, 128, 128, 32)
output: (None, 128, 128, 32)
conv2d_7: Conv2D
input: (None, 256, 256, 64)
output: (None, 256, 256, 32)
activation_14: Activation
input: (None, 128, 128, 32)
output: (None, 128, 128, 32)
conv2d_transpose_7: Conv2DTranspose
input: (None, 128, 128, 32)
output: (None, 128, 128, 32)
batch_normalization_14: BatchNormalization
input: (None, 128, 128, 32)
output: (None, 128, 128, 32)
up_sampling2d_6: UpSampling2D
input: (None, 128, 128, 32)
output: (None, 256, 256, 32)
add_6: Add
input: [(None, 256, 256, 32), (None, 256, 256, 32)]
output: (None, 256, 256, 32)