Semantic Segmentation Lab

In this lab, you will build a deep learning network that locates a particular human target within an image. The premise is that a quadcopter (simulated) is searching for a target, and then will follow the target once found. It's not enough to simply say the target is present in the image in this case, but rather to know *where* in the image the target is, so that the copter can adjust its direction in order to follow.

Consequently, an image classification network is not enough to solve the problem. Intead, a semantic segmentation network is needed so that the target can be specifically located within the image.

You can click on any of the following to quickly jump to that part of this notebook:

Data Collection

We have provided you with the dataset for this lab. If you haven't already downloaded the training and validation datasets, you can check out the README for this lab's repo for instructions as well.

In [113]:
```
import os
import glob
import sys
import tensorflow as tf

from scipy import misc
import numpy as np

from tensorflow.contrib.keras.python import keras
from tensorflow.contrib.keras.python.keras import layers, models

from tensorflow import image

from utils import scoring_utils
from utils.separable_conv2d import SeparableConv2DKeras, BilinearUpSampling2D
from utils import data_iterator
from utils import plotting_tools
from utils import model_tools
```

FCN Layers

In the Classroom, we discussed the different layers that constitute a fully convolutional network. The following code will intoduce you to the functions that you will be using to build out your model.

Separable Convolutions

The Encoder for your FCN will essentially require separable convolution layers. Below we have implemented two functions - one which you can call upon to build out separable convolutions or regular convolutions. Each with batch normalization and with the ReLU activation function applied to the layers.

While we recommend the use of separable convolutions thanks to their advantages we covered in the Classroom, some of the helper code we will present for your model will require the use for regular convolutions. But we encourage you to try and experiment with each as well!

The following will help you create the encoder block and the final model for your architecture.

```python
In [114]:  def separable_conv2d_batchnorm(input_layer, filters, strides=1):
               output_layer = SeparableConv2DKeras(filters=filters,kernel_size=3, strides=strides,
                                   padding='same', activation='relu')(input_layer)

               output_layer = layers.BatchNormalization()(output_layer)
               return output_layer

           def conv2d_batchnorm(input_layer, filters, kernel_size=3, strides=1):
               output_layer = layers.Conv2D(filters=filters, kernel_size=kernel_size, strides=strides,
                                   padding='same', activation='relu')(input_layer)

               output_layer = layers.BatchNormalization()(output_layer)
               return output_layer
```

Bilinear Upsampling

The following helper function will help implement the bilinear upsampling layer. Currently, upsampling by a factor of 2 is recommended but you can try out different factors as well. You will use this to create the decoder block later!

```python
In [115]:  def bilinear_upsample(input_layer):
               output_layer = BilinearUpSampling2D((2,2))(input_layer)
               return output_layer
```

Build the Model

In the following cells, we will cover how to build the model for the task at hand.

- We will first create an Encoder Block, where you will create a separable convolution layer using an input layer and the size(depth) of the filters as your inputs.
- Next, you will create the Decoder Block, where you will create an upsampling layer using bilinear upsampling, followed by a layer concatentaion, and some separable convolution layers.
- Finally, you will combine the above two and create the model. In this step you will be able to experiment with different number of layers and filter sizes for each to build your model.

Let's cover them individually below.

Encoder Block

Below you will create a separable convolution layer using the separable_conv2d_batchnorm() function. The `filters` parameter defines the size or depth of the output layer. For example, 32 or 64.

```
In [116]: def encoder_block(input_layer, filters, strides):

              # TODO Create a separable convolution layer using the separable_conv2d_bat
          chnorm() function.
              output_layer = separable_conv2d_batchnorm( input_layer, filters, strides)

              return output_layer
```

Decoder Block

The decoder block, as covered in the Classroom, comprises of three steps -

- A bilinear upsampling layer using the upsample_bilinear() function. The current recommended factor for upsampling is set to 2.
- A layer concatenation step. This step is similar to skip connections. You will concatenate the upsampled small_ip_layer and the large_ip_layer.
- Some (one or two) additional separable convolution layers to extract some more spatial information from prior layers.

In [117]:
```python
def decoder_block(small_ip_layer, large_ip_layer, filters):

    # TODO Upsample the small input layer using the bilinear_upsample() functi
on.
    output_upsample = bilinear_upsample( small_ip_layer)
    # TODO Concatenate the upsampled and large input layers using layers.conca
tenate
    output_concatenate = layers.concatenate( large_ip_layer, output_upsample )
    # TODO Add some number of separable convolution layers
    separable_conv2d_batchnorm( output_concatenate, filters )
    return output_layer
```

Model

Now that you have the encoder and decoder blocks ready, you can go ahead and build your model architecture!

There are three steps to the following:

- Add encoder blocks to build out initial set of layers. This is similar to how you added regular convolutional layers in your CNN lab.
- Add 1x1 Convolution layer using conv2d_batchnorm() function. Remember that 1x1 Convolutions require a kernel and stride of 1.
- Add decoder blocks for upsampling and skip connections.

In [118]:
```python
def fcn_model(inputs, num_classes):

    # TODO Add Encoder Blocks.
    # Remember that with each encoder layer, the depth of your model (the numb
er of filters) increases.
    encode_out= encoder_block(inputs,16,1)
    # TODO Add 1x1 Convolution Layer using conv2d_batchnorm().
    conv_out = conv2d_batchnorm(encode_out, 16)
    # TODO: Add the same number of Decoder Blocks as the number of Encoder Blo
cks
    x = decoder_block( inputs, encode_out, 16 )

    # The function returns the output layer of your model. "x" is the final la
yer obtained from the last decoder_block()
    return layers.Conv2D(num_classes, 3, activation='softmax', padding='same')
(x)
```

Training

The following cells will utilize the model you created and define an ouput layer based on the input and the number of classes.Following that you will define the hyperparameters to compile and train your model!

In [119]:

```python
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""

image_hw = 128
image_shape = (image_hw, image_hw, 3)
inputs = layers.Input(image_shape)
num_classes = 3

# Call fcn_model()
output_layer = fcn_model(inputs, num_classes)
```

```
---------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-119-c5c53411be0d> in <module>()
      9
     10 # Call fcn_model()
---> 11 output_layer = fcn_model(inputs, num_classes)


<ipython-input-118-a0a923557da3> in fcn_model(inputs, num_classes)
      7     conv_out = conv2d_batchnorm(encode_out, 16)
      8     # TODO: Add the same number of Decoder Blocks as the number of En
coder Blocks
----> 9     x = decoder_block( inputs, encode_out, 16 )
     10
     11     # The function returns the output layer of your model. "x" is the
 final layer obtained from the last decoder_block()


<ipython-input-117-bd914a2c1e12> in decoder_block(small_ip_layer, large_ip_la
yer, filters)
      4     output_upsample = bilinear_upsample( small_ip_layer)
      5     # TODO Concatenate the upsampled and large input layers using lay
ers.concatenate
----> 6     output_concatenate = layers.concatenate( large_ip_layer, output_u
psample )
      7     # TODO Add some number of separable convolution layers
      8     separable_conv2d_batchnorm( output_concatenate, filters )


~\AppData\Local\conda\conda\envs\RoboND\lib\site-packages\tensorflow\contrib
\keras\python\keras\layers\merge.py in concatenate(inputs, axis, **kwargs)
    536         A tensor, the concatenation of the inputs alongside axis
`axis`.
    537     """
--> 538     return Concatenate(axis=axis, **kwargs)(inputs)
    539
    540


~\AppData\Local\conda\conda\envs\RoboND\lib\site-packages\tensorflow\contrib
\keras\python\keras\engine\topology.py in __call__(self, inputs, **kwargs)
    430
    431         # Actually call the layer (optionally building it).
--> 432         output = super(Layer, self).__call__(inputs, **kwargs)
    433
    434         # Handle mask computation.


~\AppData\Local\conda\conda\envs\RoboND\lib\site-packages\tensorflow\python\l
ayers\base.py in __call__(self, inputs, *args, **kwargs)
    432             input_shapes = [x.get_shape() for x in input_list]
    433             if len(input_shapes) == 1:
--> 434                 self.build(input_shapes[0])
    435             else:
    436                 self.build(input_shapes)


~\AppData\Local\conda\conda\envs\RoboND\lib\site-packages\tensorflow\contrib
\keras\python\keras\layers\merge.py in build(self, input_shape)
    288         # Used purely for shape validation.
    289         if not isinstance(input_shape, list):
--> 290             raise ValueError('`Concatenate` layer should be called '
    291                              'on a list of inputs')
```

```
292          if all([shape is None for shape in input_shape]):
```

ValueError: `Concatenate` layer should be called on a list of inputs

Hyperparameters

Define and tune your hyperparameters.

- **batch_size**: number of training samples/images that get propagated through the network in a single pass.
- **num_epochs**: number of times the entire training dataset gets propagated through the network.
- **steps_per_epoch**: number of batches of training images that go through the network in 1 epoch. We have provided you with a default value. One recommended value to try would be based on the total number of images in training dataset divided by the batch_size.
- **validation_steps**: number of batches of validation images that go through the network in 1 epoch. This is similar to steps_per_epoch, except validation_steps is for the validation dataset. We have provided you with a default value for this as well.
- **workers**: maximum number of processes to spin up. This can affect your training speed and is dependent on your hardware. We have provided a recommended value to work with.

```
In [120]:  learning_rate = 0.1
           batch_size = 16
           num_epochs = 5
           steps_per_epoch = 200
           validation_steps = 50
           workers = 2
```

In [121]:
```python
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
# Define the Keras model and compile it for training
model = models.Model(inputs=inputs, outputs=output_layer)

model.compile(optimizer=keras.optimizers.Adam(learning_rate), loss='categorical_crossentropy')

# Data iterators for loading the training and validation data
train_iter = data_iterator.BatchIteratorSimple(batch_size=batch_size,
                                                data_folder=os.path.join('..',
'data', 'train'),
                                                image_shape=image_shape,
                                                shift_aug=True)

val_iter = data_iterator.BatchIteratorSimple(batch_size=batch_size,
                                                data_folder=os.path.join('..', 'd
ata', 'validation'),
                                                image_shape=image_shape)

logger_cb = plotting_tools.LoggerPlotter()
callbacks = [logger_cb]

model.fit_generator(train_iter,
                    steps_per_epoch = steps_per_epoch, # the number of batches
 per epoch,
                    epochs = num_epochs, # the number of epochs to train for,
                    validation_data = val_iter, # validation iterator
                    validation_steps = validation_steps, # the number of batch
es to validate on
                    callbacks=callbacks,
                    workers = workers)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-121-a650e0202de2> in <module>()
      3 """
      4 # Define the Keras model and compile it for training
----> 5 model = models.Model(inputs=inputs, outputs=output_layer)
      6
      7 model.compile(optimizer=keras.optimizers.Adam(learning_rate), loss='c
ategorical_crossentropy')

NameError: name 'output_layer' is not defined
```

```
In [76]:  # Save your trained model weights
          weight_file_name = 'model_weights'
          model_tools.save_network(model, weight_file_name)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-76-55423d204f23> in <module>()
      1 # Save your trained model weights
      2 weight_file_name = 'model_weights'
----> 3 model_tools.save_network(model, weight_file_name)

NameError: name 'model' is not defined
```

Prediction

```
In [77]:  # If you need to load a model which you previously trained you can uncomment t
          he codeline that calls the function below.

          # weight_file_name = 'model_weights'
          # restored_model = model_tools.load_network(weight_file_name)
```

```
In [78]:  # generate predictions, save in the runs, directory.
          run_number = 'run1'
          validation_path, output_path = model_tools.write_predictions_grade_set(model,r
          un_number,'validation')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-78-fc625de637a5> in <module>()
      1 # generate predictions, save in the runs, directory.
      2 run_number = 'run1'
----> 3 validation_path, output_path = model_tools.write_predictions_grade_se
t(model,run_number,'validation')

NameError: name 'model' is not defined
```

In [79]:
```
# take a look at predictions
# validation_path = 'validation'
im_files = plotting_tools.get_im_file_sample(run_number,validation_path)
for i in range(3):
    im_tuple = plotting_tools.load_images(im_files[i])
    plotting_tools.show_images(im_tuple)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-79-76bf55c454df> in <module>()
      1 # take a look at predictions
      2 # validation_path = 'validation'
----> 3 im_files = plotting_tools.get_im_file_sample(run_number,validation_pa
th)
      4 for i in range(3):
      5     im_tuple = plotting_tools.load_images(im_files[i])

NameError: name 'validation_path' is not defined
```

Evaluation

Let's evaluate your model!

In [80]:
```
scoring_utils.score_run(validation_path, output_path)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-80-3307a410b6a8> in <module>()
----> 1 scoring_utils.score_run(validation_path, output_path)

NameError: name 'validation_path' is not defined
```

In [ ]: