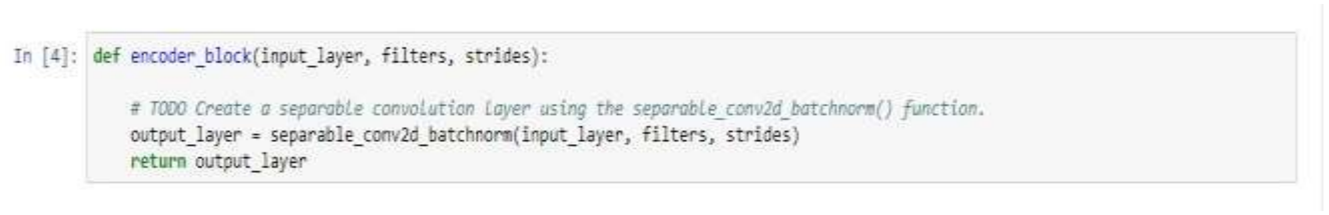
Follow me

Encoder Block –

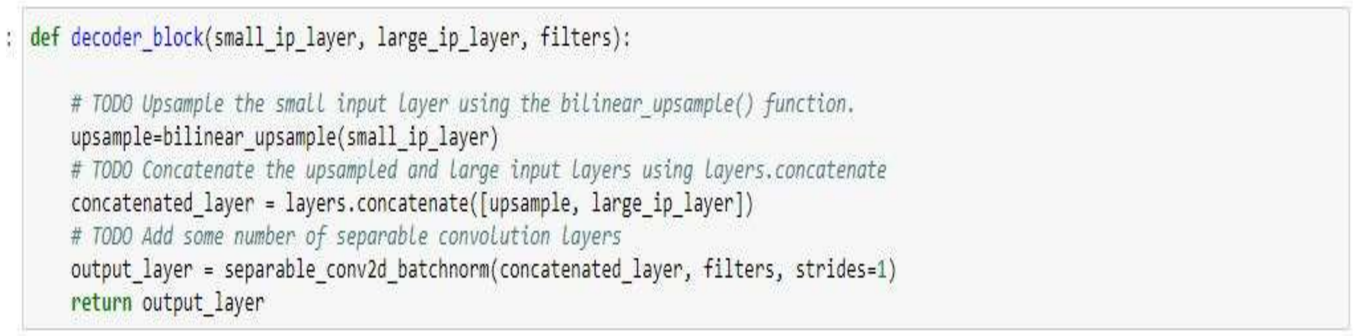


```
In [4]: def encoder_block(input_layer, filters, strides):

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

To increase the efficiency of encoder network, we have to reduce the parameters needed and to do so we create a separable convolution layer in the encoder block using separable_conv2d_batchnorm () function. We have applied batch normalization after the separable convolution layer in order to train network faster, provide a bit of regularization and allow us to use much higher learning rates, which further increases the speed at which networks train.

Decoder Block –



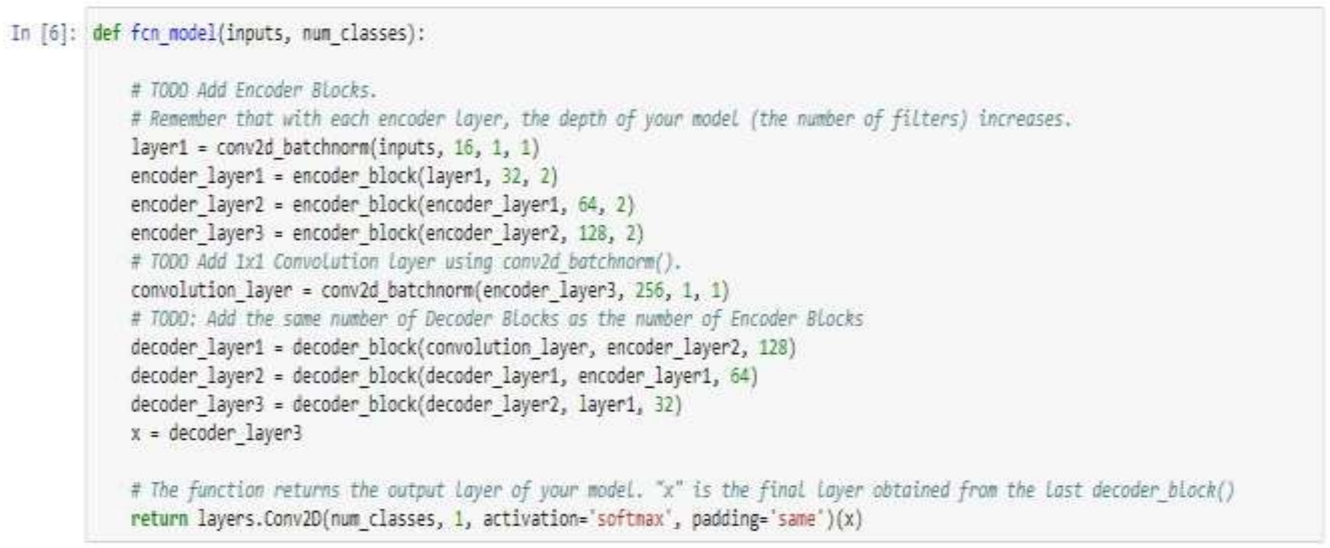
```
: def decoder_block(small_ip_layer, large_ip_layer, filters):

      # TODO Upsample the small input layer using the bilinear_upsample() function.
      upsample=bilinear_upsample(small_ip_layer)
      # TODO Concatenate the upsampled and large input layers using layers.concatenate
      concatenated_layer = layers.concatenate([upsample, large_ip_layer])
      # TODO Add some number of separable convolution layers
      output_layer = separable_conv2d_batchnorm(concatenated_layer, filters, strides=1)
      return output_layer
```

In the decoder block, we perform 3 steps. First is bilinear upsampling to speed up performance. Second is to concatenate the upsampled and large input layer which will help to simplify the network and third is to add some separable convolution layers which help to learn those finer spatial details from the previous layers better.

Model-

FCN architecture-

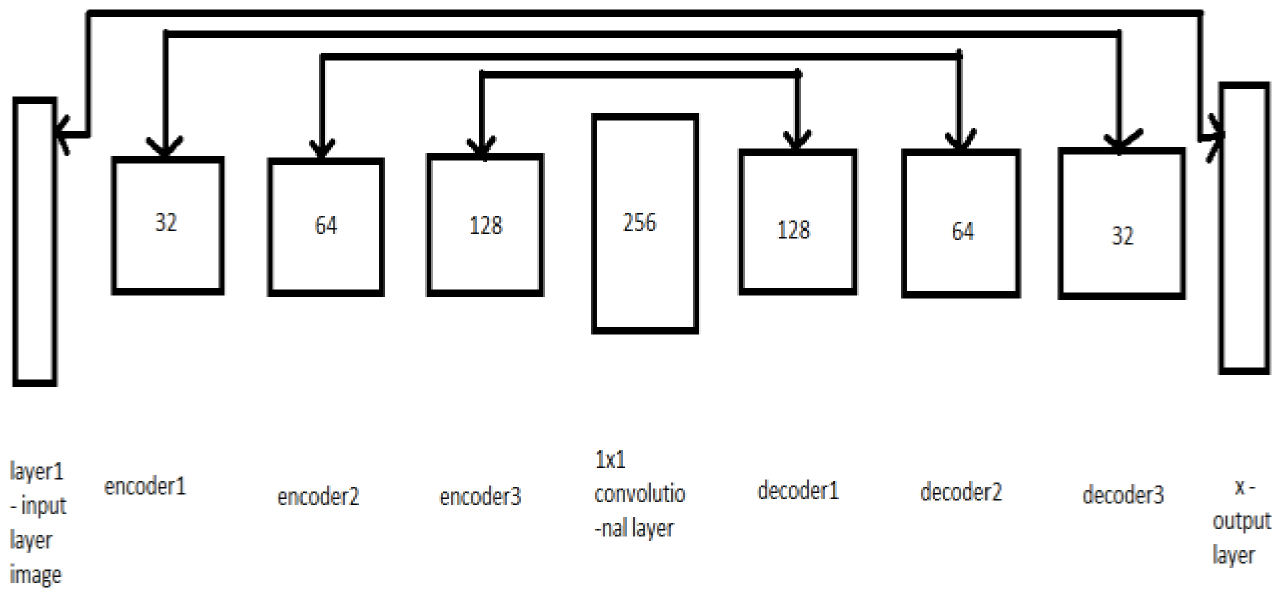


```
In [6]: def fcn_model(inputs, num_classes):

            # TODO Add Encoder Blocks.
            # Remember that with each encoder layer, the depth of your model (the number of filters) increases.
            layer1 = conv2d_batchnorm(inputs, 16, 1, 1)
            encoder_layer1 = encoder_block(layer1, 32, 2)
            encoder_layer2 = encoder_block(encoder_layer1, 64, 2)
            encoder_layer3 = encoder_block(encoder_layer2, 128, 2)
            # TODO Add 1x1 Convolution layer using conv2d_batchnorm().
            convolution_layer = conv2d_batchnorm(encoder_layer3, 256, 1, 1)
            # TODO: Add the same number of Decoder Blocks as the number of Encoder Blocks
            decoder_layer1 = decoder_block(convolution_layer, encoder_layer2, 128)
            decoder_layer2 = decoder_block(decoder_layer1, encoder_layer1, 64)
            decoder_layer3 = decoder_block(decoder_layer2, layer1, 32)
            x = decoder_layer3

            # The function returns the output layer of your model. "x" is the final layer obtained from the last decoder_block()
            return layers.Conv2D(num_classes, 1, activation='softmax', padding='same')(x)
```

Fully connected layers don’t preserve spatial information which makes it ideal for classification task but the fully convolutional network is used to preserve the spatial information of the recognized object in images throughout the network which makes it better suited for this task. The network consists of 3 main stages, the encoding stage, the implementation of 1x1 convolutional layer, and the decoding stage.



Encoder layer -

An encoding layer is implemented to extract the features in the image. In layer 1 we take in R-G-B images and pass through 3 encoder layers. First encoder layer we implemented is of size 32, second encoder layer has size of 64 and third encoder layer is of size 128.

1x1 Convolutions -

1x1 convolution is used to reduce the dimensionality of the layer while preserving the feature of the encoded layer and make the model deeper and have more parameter. Here we applied 1x1 convolution layer of filter size of 256.

Decoder layer -

Decoding layers are implemented at the end of the neural network model to upscale the output of the encoder such that the output layer is of the same size as the input image which results in segmenting each and every pixel of the input images. The first decoding layer is convoluted with an encoder filter of size 128, while the second decoding layer is convoluted with an encoder filter of size 64 and third decoding layer is convoluted with a filter size of 32.

Skip connection –

To retain the lost information in the encoding stage, skip connections are used. Skip connections are implemented by connecting output of one layer to a non-adjacent layer which allows the network to use information from multiple resolutions which consequently improves the recognition process or segmentation. Here three skip connections are used first is encoder1 to decoder3, second is encoder2 to decoder2 and third is encoder3 to decoder1.

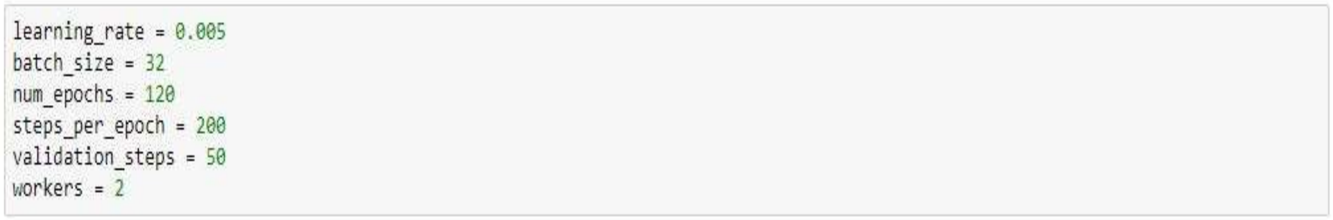
Hyper parameters –

- **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.

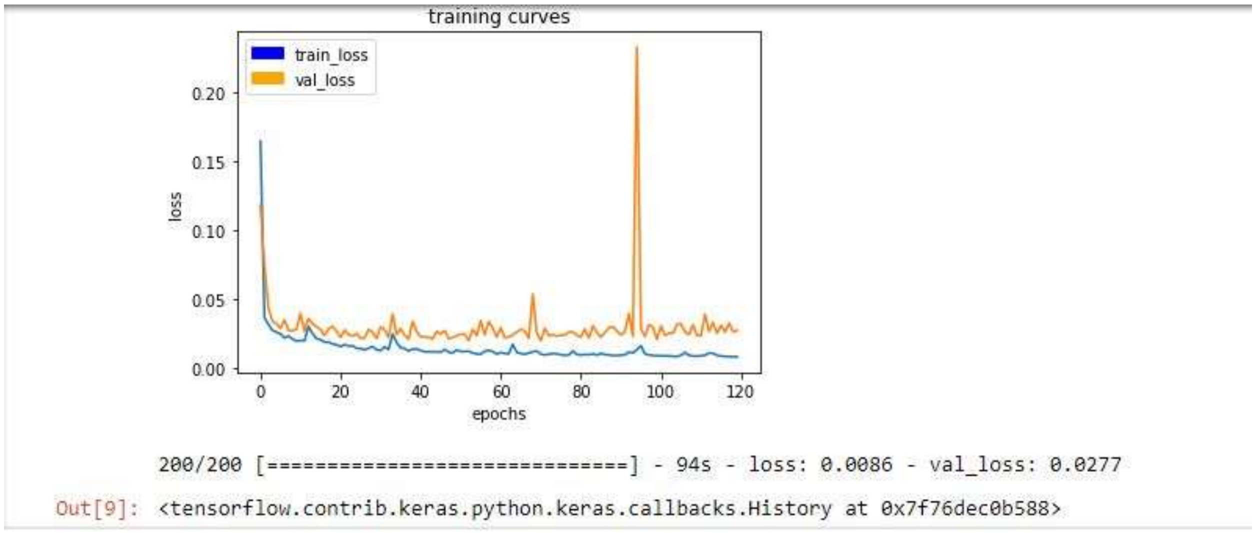
After experimenting with different values it was observed that after decreasing the learning rate the accuracy improves and after increasing the epoch the accuracy improves. The final values I used for this projects are –



```
learning_rate = 0.005
batch_size = 32
num_epochs = 120
steps_per_epoch = 200
validation_steps = 50
workers = 2
```

Results –

The below image shows the output of model trained in AWS with 120 epochs –



```
200/200 [==============================] - 94s - loss: 0.0086 - val_loss: 0.0277
Out[9]: <tensorflow.contrib.keras.python.keras.callbacks.History at 0x7f76dec0b588>
```

The final accuracy obtained using the above parameter is 44%.

Future Enhancements-

The model can further be improved by experimenting with different hyper parameters including trying different optimizer algorithms. Collecting and training on more data will also dramatically increase the accuracy of the network. By collecting more data for the target, we can also train the model to follow animals like cats and dogs. Since training neural network takes a lot of time, model checkpoints can be added to save the calculated weights to avoid loss of data if training stops abruptly.