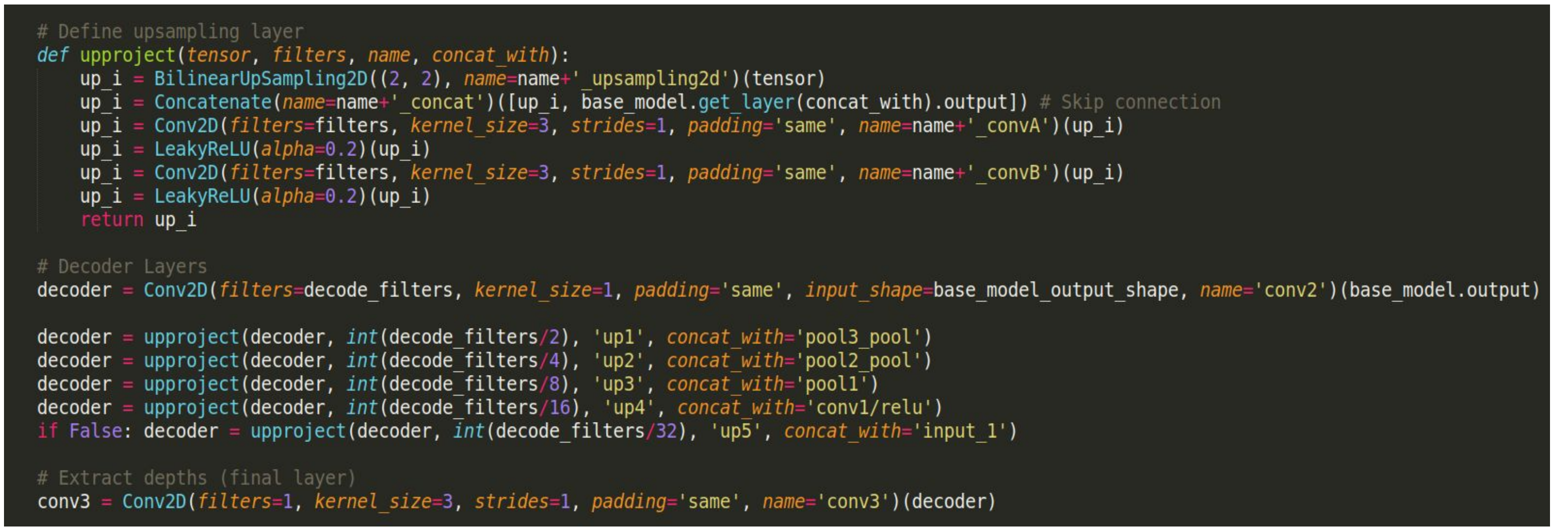
A QUICK UPDATE

30/12/2020

The assigned task right now is to implement the MBM algorithm in place of multiplication in the Convolution Operation.

The following is the code snippet that is used to actually stack layers in the decoder part of the architecture.



```python
# Define upsampling layer
def upproject(tensor, filters, name, concat_with):
    up_i = BilinearUpSampling2D((2, 2), name=name+'_upsampling2d')(tensor)
    up_i = Concatenate(name=name+'_concat')([up_i, base_model.get_layer(concat_with).output]) # Skip connection
    up_i = Conv2D(filters=filters, kernel_size=3, strides=1, padding='same', name=name+'_convA')(up_i)
    up_i = LeakyReLU(alpha=0.2)(up_i)
    up_i = Conv2D(filters=filters, kernel_size=3, strides=1, padding='same', name=name+'_convB')(up_i)
    up_i = LeakyReLU(alpha=0.2)(up_i)
    return up_i

# Decoder Layers
decoder = Conv2D(filters=decode_filters, kernel_size=1, padding='same', input_shape=base_model_output_shape, name='conv2')(base_model.output)

decoder = upproject(decoder, int(decode_filters/2), 'up1', concat_with='pool3_pool')
decoder = upproject(decoder, int(decode_filters/4), 'up2', concat_with='pool2_pool')
decoder = upproject(decoder, int(decode_filters/8), 'up3', concat_with='pool1')
decoder = upproject(decoder, int(decode_filters/16), 'up4', concat_with='conv1/relu')
if False: decoder = upproject(decoder, int(decode_filters/32), 'up5', concat_with='input_1')

# Extract depths (final layer)
conv3 = Conv2D(filters=1, kernel_size=3, strides=1, padding='same', name='conv3')(decoder)
```

Each function call of these layers is actually a call to the constructors of classes with the same name. (for instance Conv2D is a call to the constructor of class Conv2D)

So, in order to make a custom Convolutional layer (in our case, a custom convolutional layer where the MBM multiplication algo has to be applied), a subclass has to be created inheriting from the appropriate base class.

From the documentation of layers in tensorflow 2.0/ keras:
Conv2D <----- Conv <----- Layer
The above mentioned is the manner in which the Conv2D class is inherited from the Layer (base class) and from which attributes are shared.

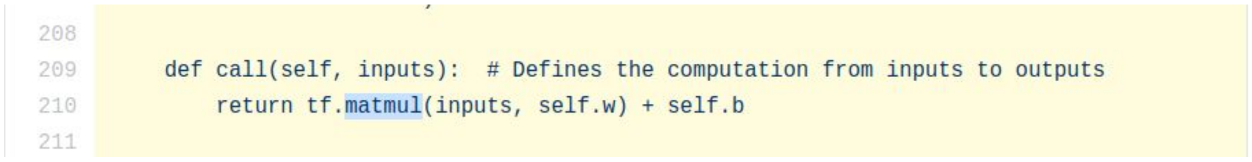
Github source codes: (for classes)

1) https://github.com/tensorflow/tensorflow/blob/v2.4.0/tensorflow/python/keras/layers/convolutional.py#L516-L671 - Source code for Conv class from which Conv2D is sub-classed
2) https://github.com/tensorflow/tensorflow/blob/v2.4.0/tensorflow/python/keras/engine/base_layer.py#L114-L3103 - Source code for Layer class

In order to make changes in the way multiplication operation is done, I assume changes have to be made at the appropriate base class… in this case the Layer class.

Reason:-
From the documentation, it seems like the basic matrix multiplication operation - tf.matmul() is done at the Layer class.
Though I'm not very sure because what this function does is matrix multiplication as opposed to dot product in usual convolution operation.

Below is an excerpt from the source code of Layer class where multiplication operation is performed.



```
    def call(self, inputs):  # Defines the computation from inputs to outputs
        return tf.matmul(inputs, self.w) + self.b
```

So my idea is to replace the function here with the MBM algo and give it a try.

Other Limitations:-
In tensorflow, tensors (the basic data-units) do not support all arithmetic operations directly and need to be done by using functions such as tf.matmul().
I am looking through them as well.

All of this is pretty new to me and I feel there must be some other easier way to go about it! Any suggestions would be really helpful.

Thank You