# Project: Follow-Me

## Intro

In this project, a deep neural network was trained to identify and track a target in simulation. This is an example of so-called “follow me” application like this are key to many fields of robotics and the very same techniques you apply here could be extended to scenarios like advanced cruise control in autonomous vehicles or human-robot collaboration in industry.

Since this task requires spatial information understanding, a fully convolutional network (FCN) arthitecture appears to be a good approach.

# Neural Network Building Blocks

The network build in this project consists of 3 main building blocks:

1. **Encoders** consist of *separable convolution* and *batch normalization* layers. Encoders map input space to a compressed future representations at the cost of "depth" (additional dimensionalities) which works vell for images taking relatevely a lot of memory in "raw" form and where what is we usually interested in spans over multiple nearby pixels. Convolutions are responsbile for building the compressed futures while batch normalization extracts the "mean" value of nearby data points.

```
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
```

2. **Decoders** in this project are represented by *upsample* and *concatenation* layers. Decoders are responsible for re-projection of hidden input layers to the output to re-construct the compressed input into the original space. The upsampler layer accepts a "weighted average" of pixel data from the adjacent pixels to return a more useful value. The concatenation layer stacks the upsample and large input layers together. Finally, the output layer of decoder performs another separable convolution to preserve the spatial information.

Since encoders extract features and preserve their spatial context within an image, decoders can take outputs of the encoders, upscale it back to the original image size and perform pixel-wise classification on the upscaled feature map. However, during the encoding step, resolution is lost, due to images being compressed into a smaller spatial space (with each layer), so the final output has lower resolution compared to the original image.

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

    # TODO Upsample the small input layer using the bilinear_upsample() function.
    upsampled = bilinear_upsample(small_ip_layer)

    # TODO Concatenate the upsampled and large input layers using layers.concatenate
    concat_layer = layers.concatenate([upsampled, large_ip_layer])

    # TODO Add some number of separable convolution layers
    conv_l1 = separable_conv2d_batchnorm(concat_layer, filters)
    output_layer = separable_conv2d_batchnorm(conv_l1, filters)

    return output_layer
```

3. **1x1 convolutions** are used to reduce the dimensionality while retaining the information. In its essence, the 1x1 convolution layer behaves as a linear coordinate-dependent transformation in the filter space. Additionally, its usage is a function of kernel size and results in less over-fitting (especially when using stochastic gradient descent). 1x1 convolutions, while mathematically equivalent to Fully Connected Layers, are more flexible. 1x1 convolutions help increase model accuracy while allowing new parameters and non-linearity.

The usual placement of those blocks is:

Input -> Encoders (>1) -> 1x1 conv -> Decoders (>1) -> Output Layer

# Optimizations

FCN frequently use *skip connections* to improve the gradient flow through the network and protect themselves from vanishing gradients. This effectively increases the capacity without increasing the number of parameters via injecting one of previous layers outputs into one of the deeper layers so it could utlized the information from it as well as inputs from the previous layer.

# Input data

I decided I'll try to reach the required model score of 40% without any additional data because of two reasons:

- in real world it is not unfrequent situation that it is hard to get more data
- the data I would add may still differ in its distribution from the data set used to estimate the score.

This appears to me a really serious reason as I may spend a lot of time getting more train and validation data which simply is not from the same distribution as the evaluation data set. This would provide no benefit to the final score.

# Final model

To build the model in addition to hyper parameter selection (which will be described later) I tried to experiement with 2-3 hidden levels of encoders and decoders with filter counts of 32 and 64 for the first layer.

Here is the code for the model getting the required score:

```
def fcn_model(inputs, num_classes):
    f = 32
    # TODO Add Encoder Blocks.
    # Remember that with each encoder layer, the depth of your model (the number of filters) increases.
    encoder1 = encoder_block(inputs, f, 2)
    encoder2 = encoder_block(encoder1, f*2 , 2)
    encoder3 = encoder_block(encoder2, f*4 , 2)

    # TODO Add 1x1 Convolution layer using conv2d_batchnorm().
    conv = conv2d_batchnorm(encoder3, f*8, kernel_size=1, strides=1)

    # TODO: Add the same number of Decoder Blocks as the number of Encoder Blocks
    decoder1 = decoder_block(conv, encoder2, f*4)
    decoder2 = decoder_block(decoder1, encoder1, f*2)
    x = decoder_block(decoder2, inputs, f)

    # 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)
```

which translates into the following model:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         (None, 160, 160, 3)       0
_________________________________________________________________
separable_conv2d_keras_31 (S (None, 80, 80, 32)        155
_________________________________________________________________
batch_normalization_35 (Batc (None, 80, 80, 32)        128
_________________________________________________________________
separable_conv2d_keras_32 (S (None, 40, 40, 64)        2400
_________________________________________________________________
batch_normalization_36 (Batc (None, 40, 40, 64)        256
_________________________________________________________________
separable_conv2d_keras_33 (S (None, 20, 20, 128)       8896
_________________________________________________________________
batch_normalization_37 (Batc (None, 20, 20, 128)       512
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 20, 20, 256)       33024
_________________________________________________________________
batch_normalization_38 (Batc (None, 20, 20, 256)       1024
_________________________________________________________________
bilinear_up_sampling2d_11 (B (None, 40, 40, 256)       0
_________________________________________________________________
concatenate_11 (Concatenate) (None, 40, 40, 320)       0
_________________________________________________________________
separable_conv2d_keras_34 (S (None, 40, 40, 128)       43968
_________________________________________________________________
```

```
batch_normalization_39 (Batc (None, 40, 40, 128)       512
_________________________________________________________________
separable_conv2d_keras_35 (S (None, 40, 40, 128)       17664
_________________________________________________________________
batch_normalization_40 (Batc (None, 40, 40, 128)       512
_________________________________________________________________
bilinear_up_sampling2d_12 (B (None, 80, 80, 128)       0
_________________________________________________________________
concatenate_12 (Concatenate) (None, 80, 80, 160)       0
_________________________________________________________________
separable_conv2d_keras_36 (S (None, 80, 80, 64)        11744
_________________________________________________________________
batch_normalization_41 (Batc (None, 80, 80, 64)        256
_________________________________________________________________
separable_conv2d_keras_37 (S (None, 80, 80, 64)        4736
_________________________________________________________________
batch_normalization_42 (Batc (None, 80, 80, 64)        256
_________________________________________________________________
bilinear_up_sampling2d_13 (B (None, 160, 160, 64)      0
_________________________________________________________________
concatenate_13 (Concatenate) (None, 160, 160, 67)      0
_________________________________________________________________
separable_conv2d_keras_38 (S (None, 160, 160, 32)      2779
_________________________________________________________________
batch_normalization_43 (Batc (None, 160, 160, 32)      128
_________________________________________________________________
separable_conv2d_keras_39 (S (None, 160, 160, 32)      1344
_________________________________________________________________
batch_normalization_44 (Batc (None, 160, 160, 32)      128
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 160, 160, 3)       99
=================================================================
Total params: 130,521
Trainable params: 128,665
Non-trainable params: 1,856
_________________________________________________________________
```

# Hyperparameters

Train hyperparameters choice is another critical element effecting model performance in additiona to network architecture. I manually adjusted the following params:

- **learning_rate** - so that training loss smothly decreases. If this value is too low the training will be slow. If it is too high, there is a risk to never reach the optimial training performance
- **batch_size** - the smaller is the batch the longer is the training. Big bath is hard to fit into memory. Just some value in between those limits works well here.
- **num_epochs** - enough of passes through the entire training data set to train the network. Cut off value is when the training and validation loss stop to make significant progress in decreasing.
- **steps_per_epoch** - just `train_length // batch_size + 1` so that each pass is done through all the

training data

- **validation_steps** - also just `val_length // batch_size + 1` so that all validation data is used
- **workers** - this appears to be Keras-related parameter which I bumped a bit to make the training faster as my host had enough of resources

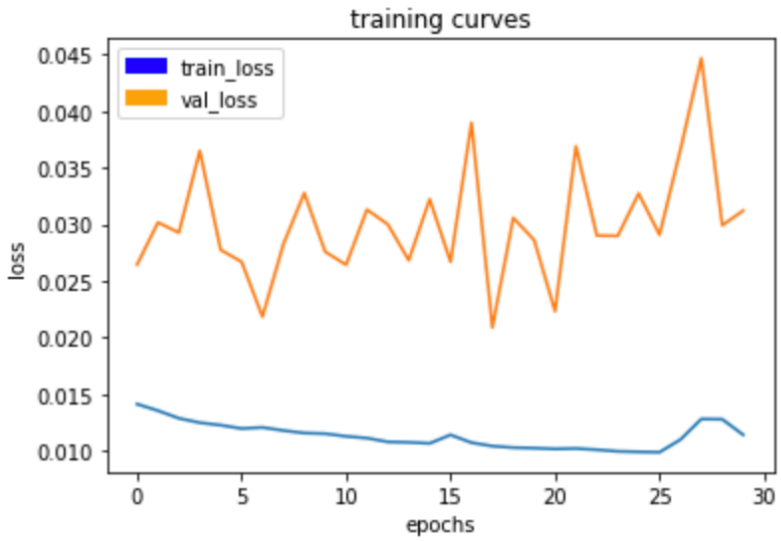
Final hyperparams:

```
train_length = 4132
val_length = 1184

learning_rate = 0.001
batch_size = 128
num_epochs = 30
steps_per_epoch = train_length // batch_size + 1
validation_steps = val_length // batch_size + 1
workers = 4
```

# Training

The learning curve is presented below:




What was suprising for me is that the learning loss got high from the beginning which is usually not the case. Anyway, the network was able to improve it.

Something else to note that validation loss is too spiky. I can't explain this for now.

Final values at epoch 30: loss: 0.0115, val_loss: 0.0312

# Results

Below are evaluation scores.

Scores for while the quad is following behind the target:

```
number of validation samples intersection over the union evaulated on 542
average intersection over union for background is 0.9951828668879676
average intersection over union for other people is 0.37432329004296483
average intersection over union for the hero is 0.9229315547094555
number true positives: 539, number false positives: 0, number false negatives: 0
```

Scores for images while the quad is on patrol and the target is not visable:

```
number of validation samples intersection over the union evaulated on 270
average intersection over union for background is 0.9868814348306688
average intersection over union for other people is 0.720158068232705
average intersection over union for the hero is 0.0
number true positives: 0, number false positives: 109, number false negatives: 0
```

Scores for images where the target may be far away:

```
number of validation samples intersection over the union evaulated on 322
average intersection over union for background is 0.9963686580727519
average intersection over union for other people is 0.4561040248301083
average intersection over union for the hero is 0.25298969243336833
number true positives: 144, number false positives: 3, number false negatives: 157
```

The final grade score is 0.4218246910706663. It is not a bad result above the required score. However, the model did not that well for patrol and target is far away situations having a lot of false positives or false negatives. However, although there is some room to improvement, I would say I won't be suprised if a human demonstrated a similar performance for those situations.

# Future Enhancements

The first apparent thing to improve the score would be getting more data which is still possible for this case since we have the simulator. However, it is required to ensure, the evaluation would be also performed on the newly acquired data to make the evaluation fair and representative.

If it would be required to detect any other object than what the model was trained for (i.e. dog, cat, car, etc.). It would be required to do additional training. Probably, transfer learning from the existing model would help a bit to reduce the training time and the number of required training data.