Fully Convolutional Network (FCN)

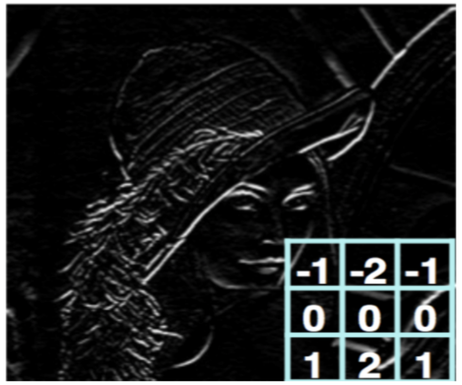
Industrial AI Lab.

Prof. Seungchul Lee

Deep Learning for Computer Vision: Review

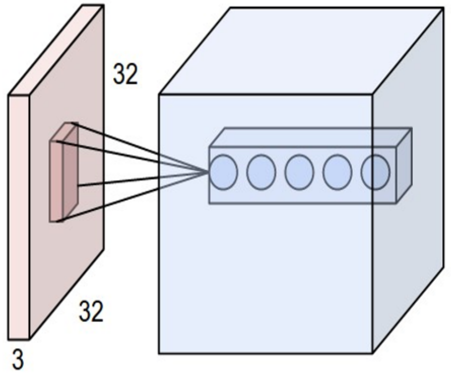
Foundations

- Why computer vision?
- Representing images
- Convolutions for feature extraction



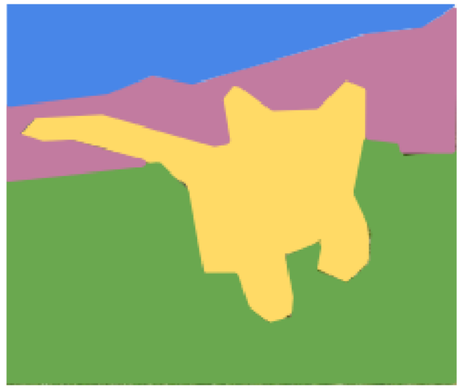
CNNs

- CNN architecture
- Application to classification: ImageNet



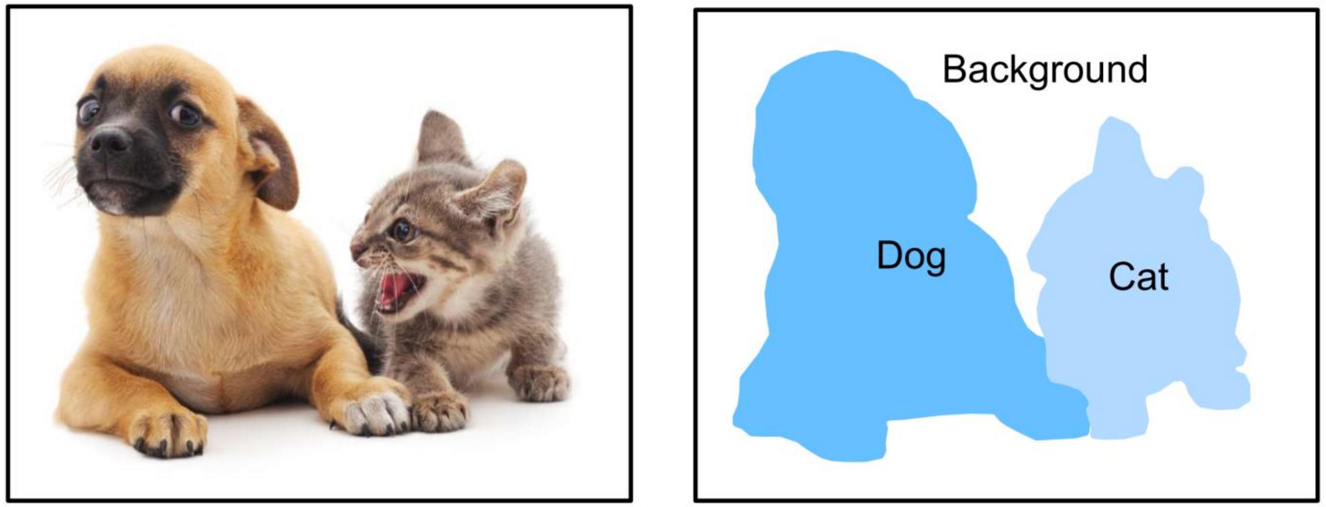
Applications

- Segmentation, object detection, image captioning
- Visualization



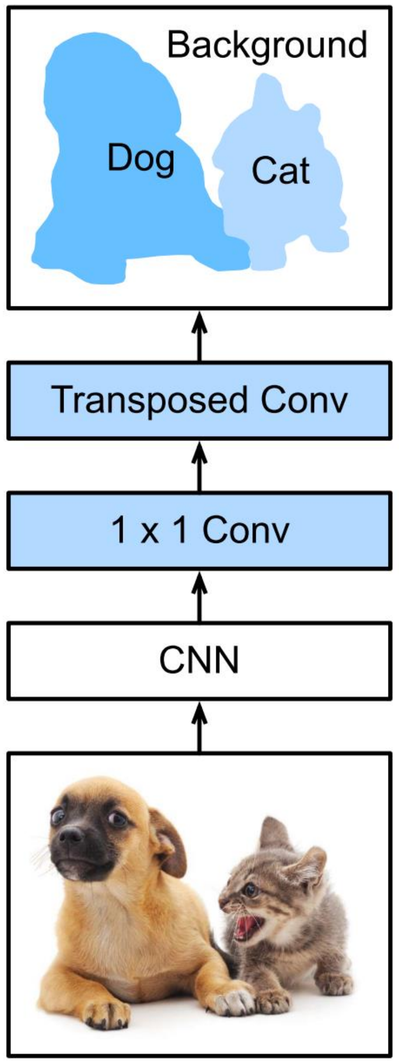
Segmentation

- Segmentation task is different from classification task because it requires predicting a class for each pixel of the input image, instead of only 1 class for the whole input.
- Classification needs to understand what is in the input (namely, the context).
- However, in order to predict what is in the input for each pixel, segmentation needs to recover not only what is in the input, but also where.
- Segment images into regions with different semantic categories. These semantic regions label and predict objects at the pixel level

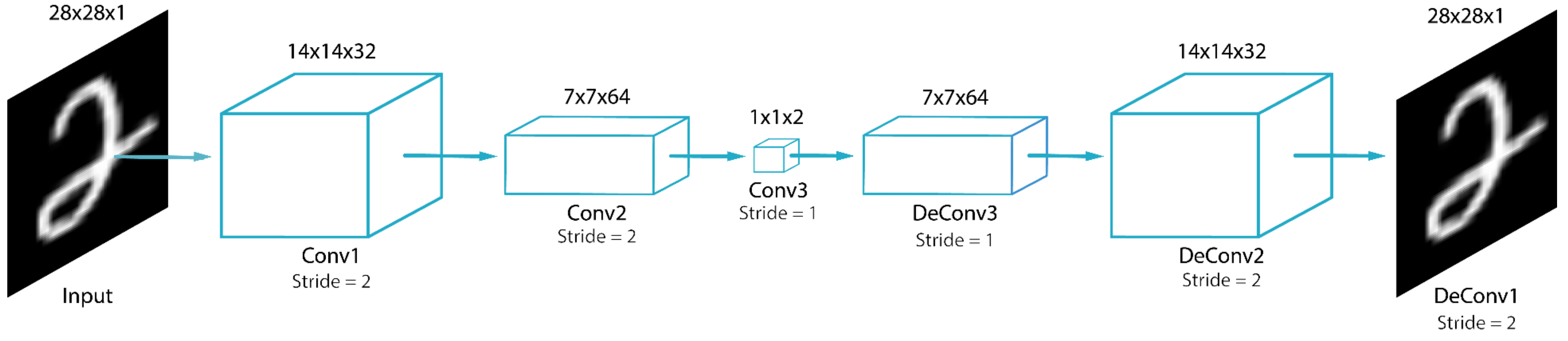


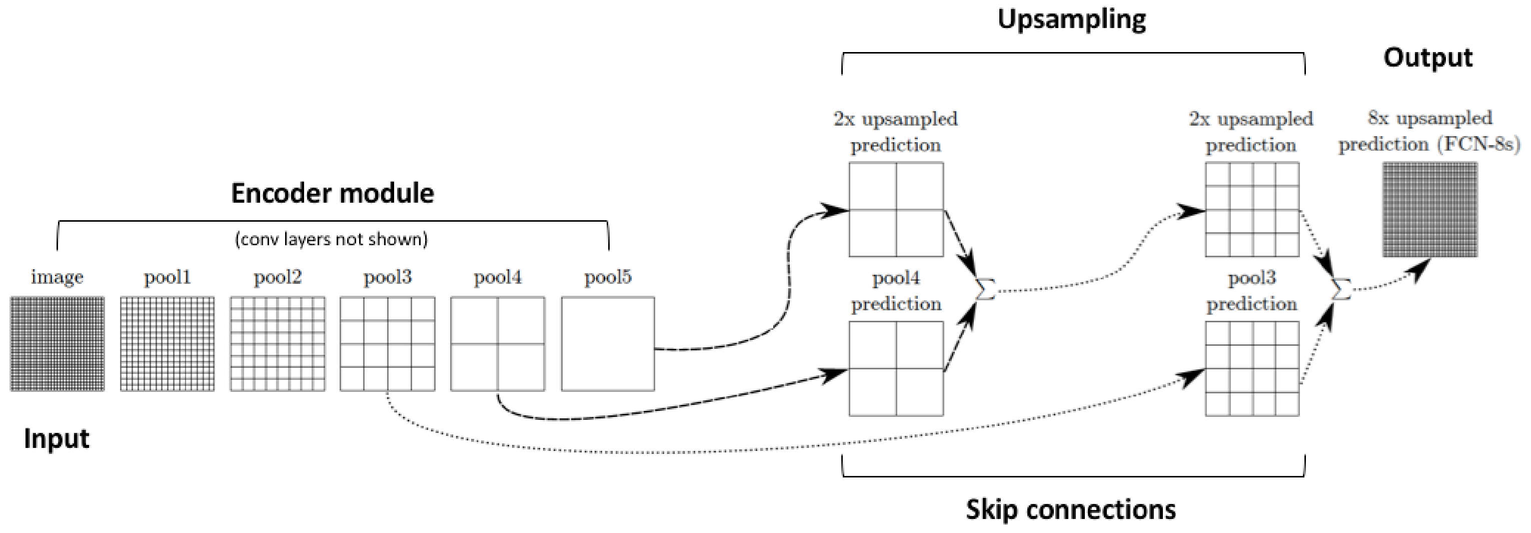
Semantic Segmentation: FCNs

- FCN uses a convolutional neural network to transform image pixels to pixel categories.
- Network designed with all convolutional layers, with down-sampling and up-sampling operations
- FCN transforms the height and width of the intermediate layer feature map back to the size of input image through the transposed convolution layer, so that the predictions have a one-to-one correspondence with input image in spatial dimension
- Given a position on the spatial dimension, the output of the channel dimension will be a category prediction of the pixel corresponding to the location.



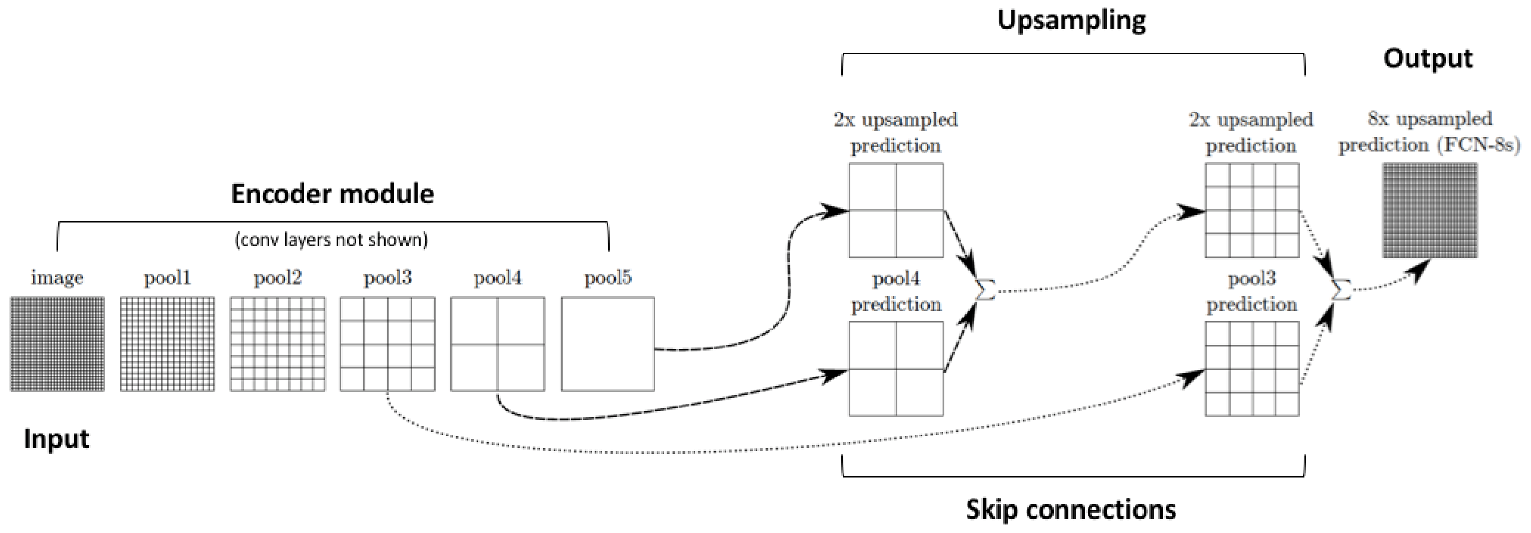
From CAE to FCN





Fully Convolutional Networks (FCNs)

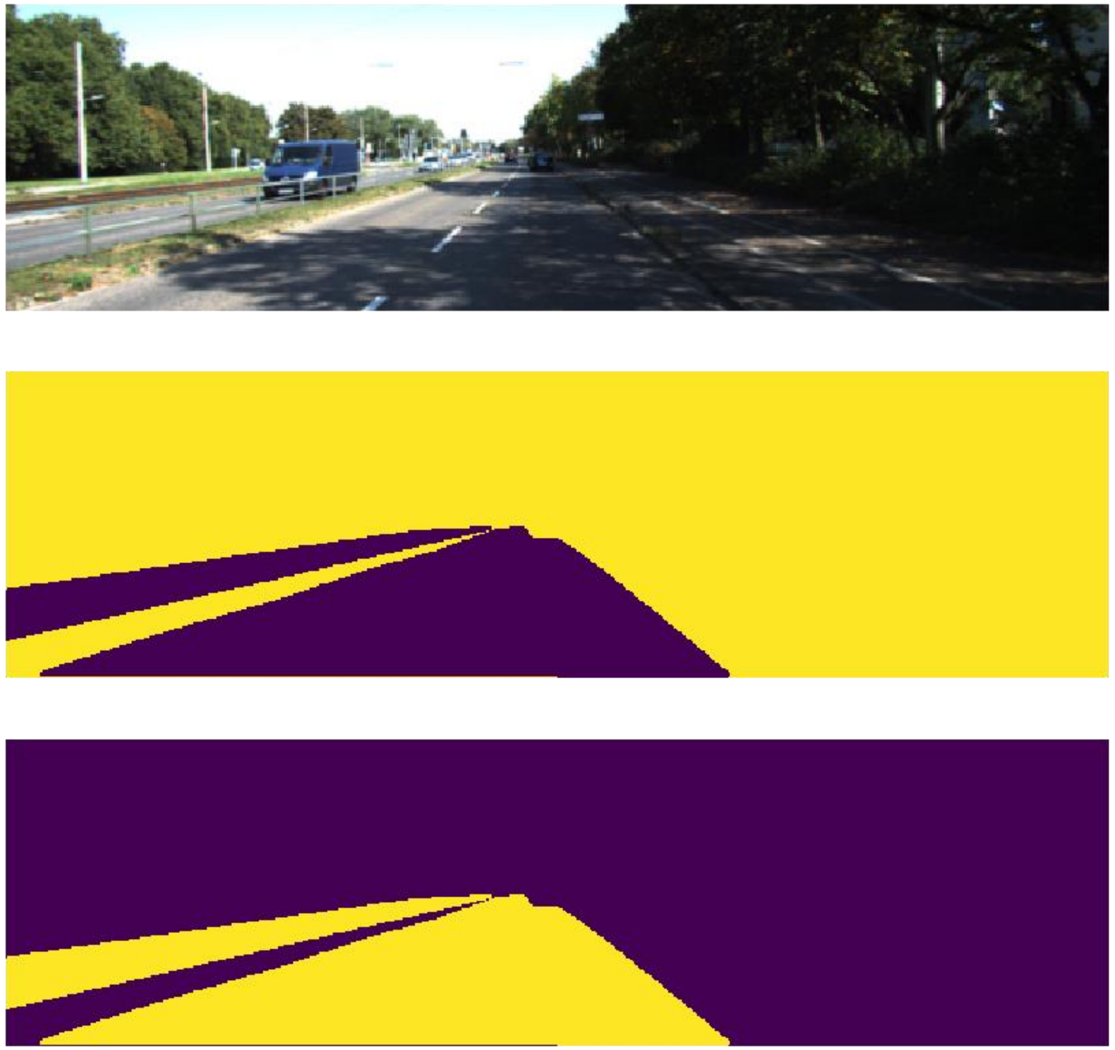
- To obtain a segmentation map (output), segmentation networks usually have 2 parts
  - Downsampling path: capture semantic/contextual information
  - Upsampling path: recover spatial information
- The downsampling path is used to extract and interpret the context (what), while the upsampling path is used to enable precise localization (where).
- Furthermore, to fully recover the fine-grained spatial information lost in the pooling or downsampling layers, we often use skip connections.
- Network can work regardless of the original image size, without requiring any fixed number of units at any stage.



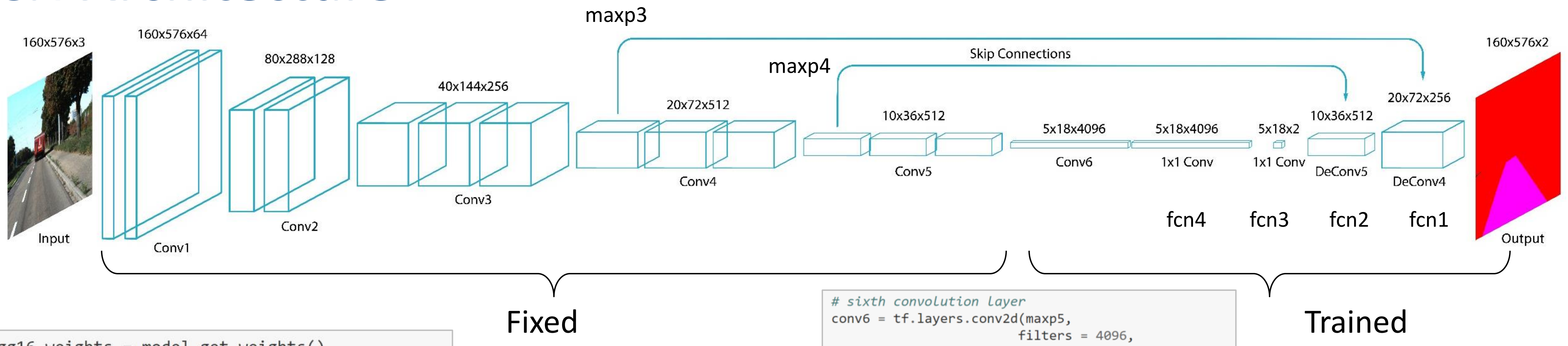
Skip Connection

- A skip connection is a connection that bypasses at least one layer.
- Here, it is often used to transfer local information by concatenating or summing feature maps from the downsampling path with feature maps from the upsampling path.
- Merging features from various resolution levels helps combining context information with spatial information.

Segmented (Labeled) Images



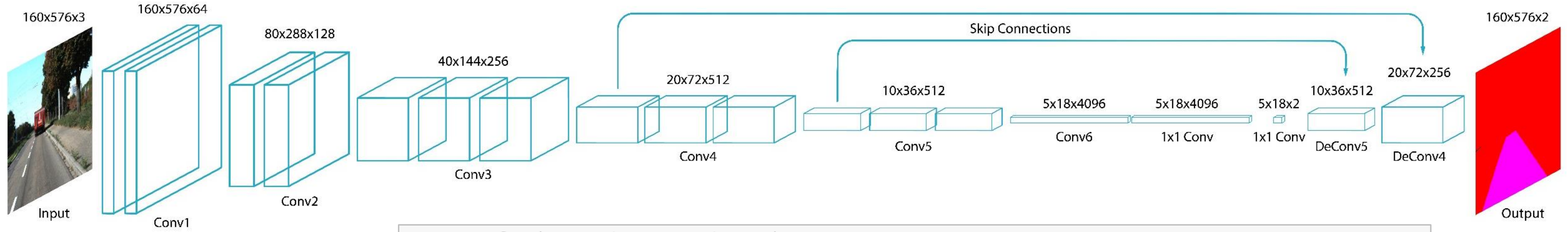
FCN Architecture



```
vgg16_weights = model.get_weights()

weights = {
    'conv1_1' : tf.constant(vgg16_weights[0]),
    'conv1_2' : tf.constant(vgg16_weights[2]),

    'conv2_1' : tf.constant(vgg16_weights[4]),
    'conv2_2' : tf.constant(vgg16_weights[6]),

    'conv3_1' : tf.constant(vgg16_weights[8]),
    'conv3_2' : tf.constant(vgg16_weights[10]),
    'conv3_3' : tf.constant(vgg16_weights[12]),

    'conv4_1' : tf.constant(vgg16_weights[14]),
    'conv4_2' : tf.constant(vgg16_weights[16]),
    'conv4_3' : tf.constant(vgg16_weights[18]),

    'conv5_1' : tf.constant(vgg16_weights[20]),
    'conv5_2' : tf.constant(vgg16_weights[22]),
    'conv5_3' : tf.constant(vgg16_weights[24]),
}
```

```
# sixth convolution layer
conv6 = tf.layers.conv2d(maxp5,
                        filters = 4096,
                        kernel_size = 7,
                        padding = 'SAME',
                        activation = tf.nn.relu)

# 1x1 convolution layers
fcn4 = tf.layers.conv2d(conv6,
                        filters = 4096,
                        kernel_size = 1,
                        padding = 'SAME',
                        activation = tf.nn.relu)
fcn3 = tf.layers.conv2d(fcn4,
                        filters = 2,
                        kernel_size = 1,
                        padding = 'SAME')

# Upsampling layers
fcn2 = tf.layers.conv2d_transpose(fcn3,
                                  filters = 512,
                                  kernel_size = 4,
                                  strides = (2, 2),
                                  padding = 'SAME')

fcn1 = tf.layers.conv2d_transpose(fcn2 + maxp4,
                                  filters = 256,
                                  kernel_size = 4,
                                  strides = (2, 2),
                                  padding = 'SAME')

output = tf.layers.conv2d_transpose(fcn1 + maxp3,
                                  filters = 2,
                                  kernel_size = 16,
                                  strides = (8, 8),
                                  padding = 'SAME')
```

Segmentation Result



```
pred = fcn(x, weights, biases)
logits = tf.reshape(pred, (-1, 2))
labels = tf.reshape(y, (-1, 2))

loss = tf.nn.softmax_cross_entropy_with_logits(logits = logits, labels = labels)
loss = tf.reduce_mean(loss)
```