Long, Jonathan, Evan Shelhamer, and Trevor Darrell.

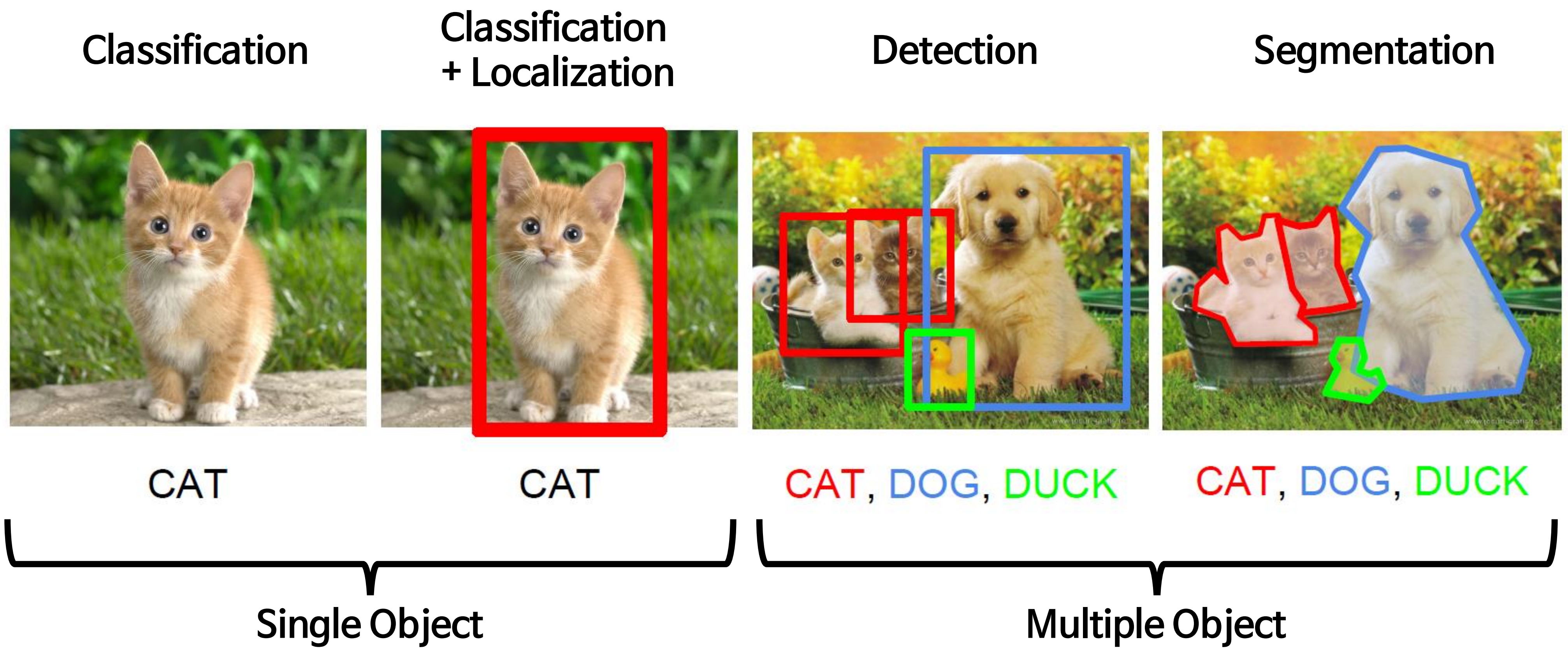
"Fully convolutional networks for semantic segmentation."

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

Introduction

Computer Vision

Computer Vision Task



Stanford cs231n 2016winter

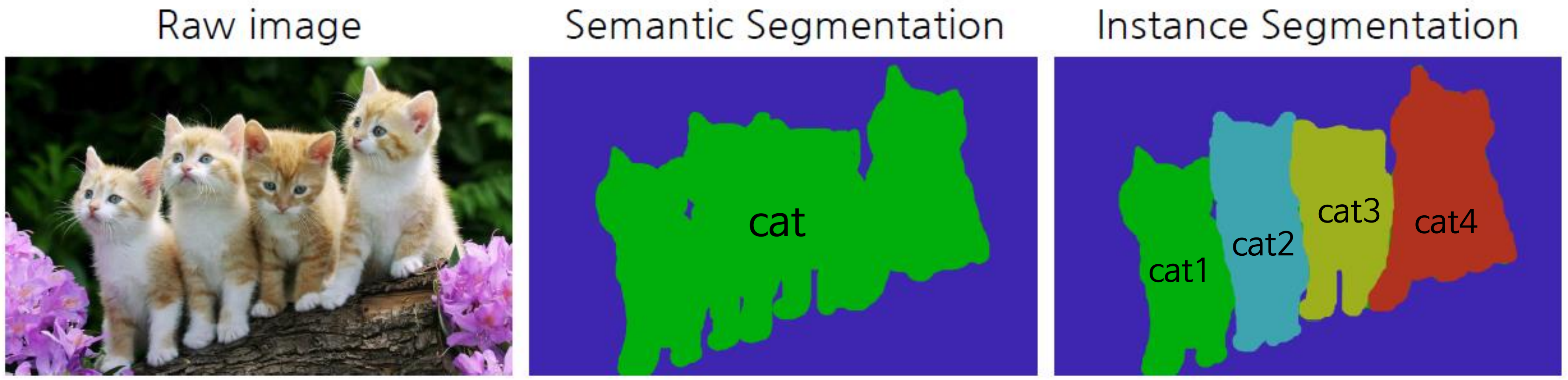
01 Introduction

Computer Vision

Segmentation

: 영상에서 의미있는 부분만을 구별해내는 기술

- Semantic Segmentation : 모든 픽셀에 label을 1개씩 할당
- Instance Segmentation : 한 class에 대하여 여러 개체를 구분하여 할당

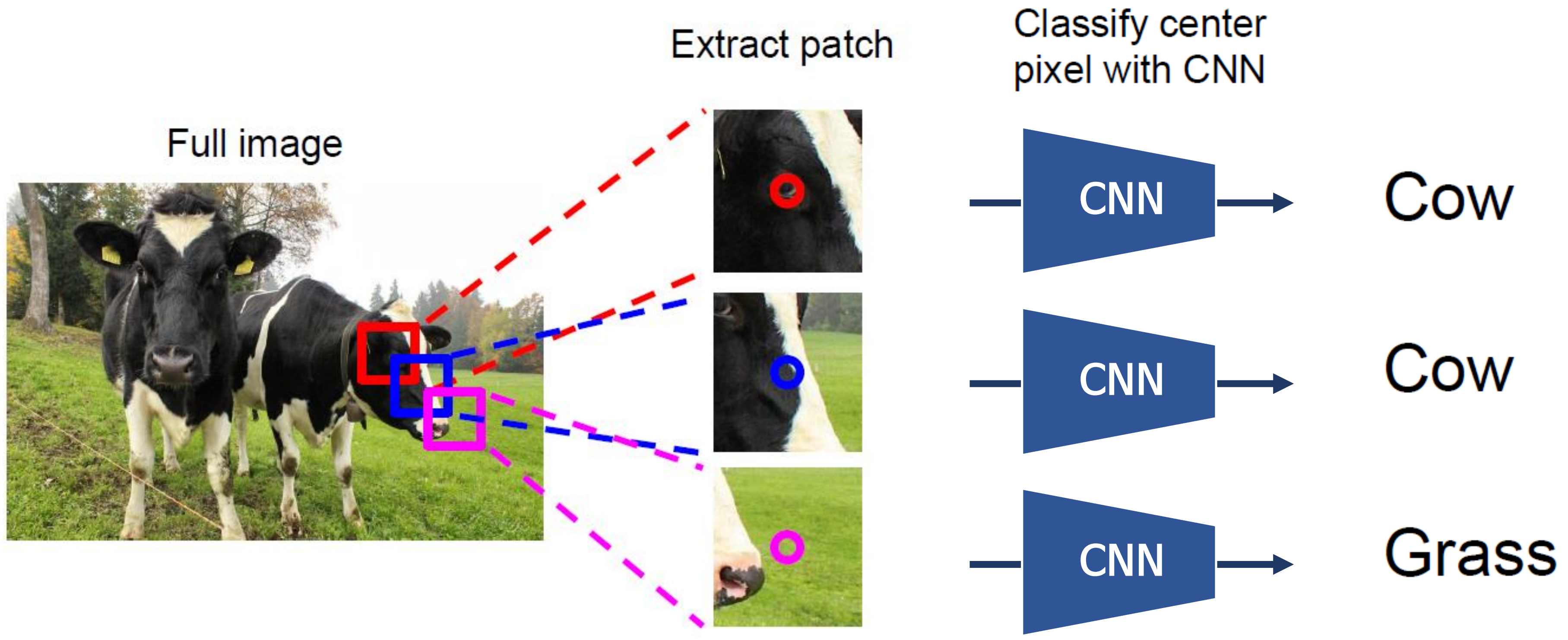


Problem : 영상 속 무엇(what)이 있는지를 확인하는 것(semantic)뿐만 아니라 어느 위치(where)에 있는지(location)까지 파악

→ Semantic과 Location은 성질상 지향하는 바가 다르기 때문에 적절히 조화롭게 해결해야 한다.

01 Introduction
Semantic Segmentation

Sliding Window



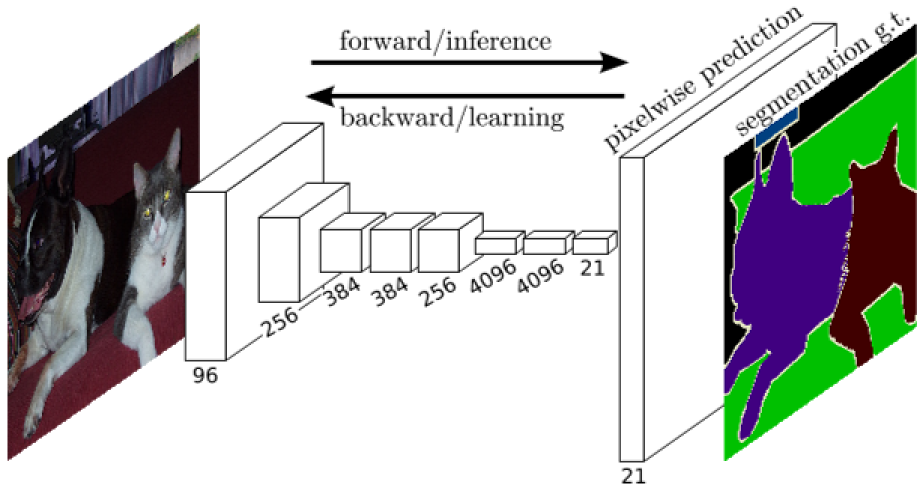
: Patch-wise segmentation은 매우 비효율적

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

01 Introduction

Semantic Segmentation

Summary

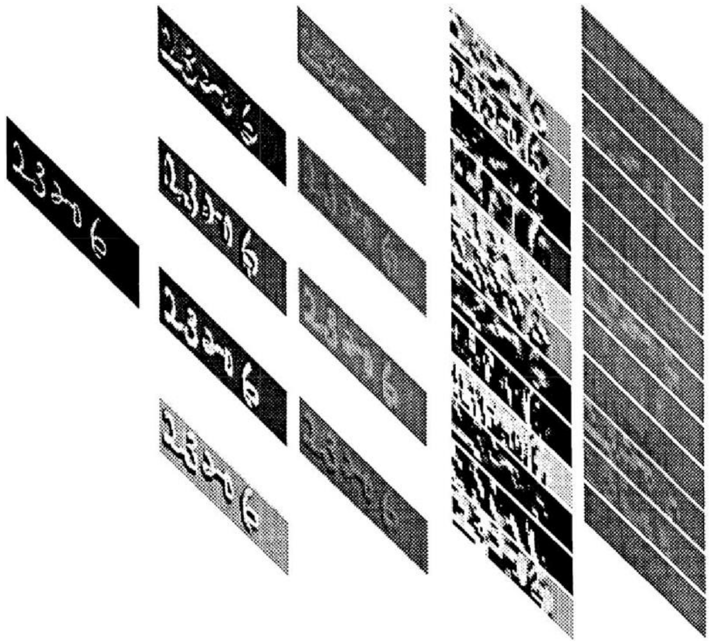


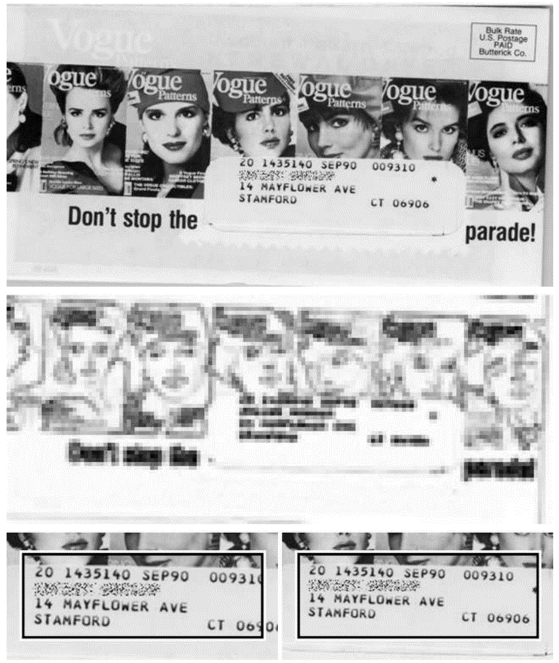
- "Fully convolutional" networks that take input of arbitrary size and produce correspondingly-sized output.
- To adapt contemporary classification networks into fully convolutional networks and transfer their learned representations by fine-tuning to the segmentation task.
- Skip architecture that combines semantic information from a deep, coarse layer with appearance information from a shallow, fine layer to produce aacurate and detailed segmentations.

01 Introduction

Semantic Segmentation

Related Work - Fully Convolutional Networks


1-dimensional input strings

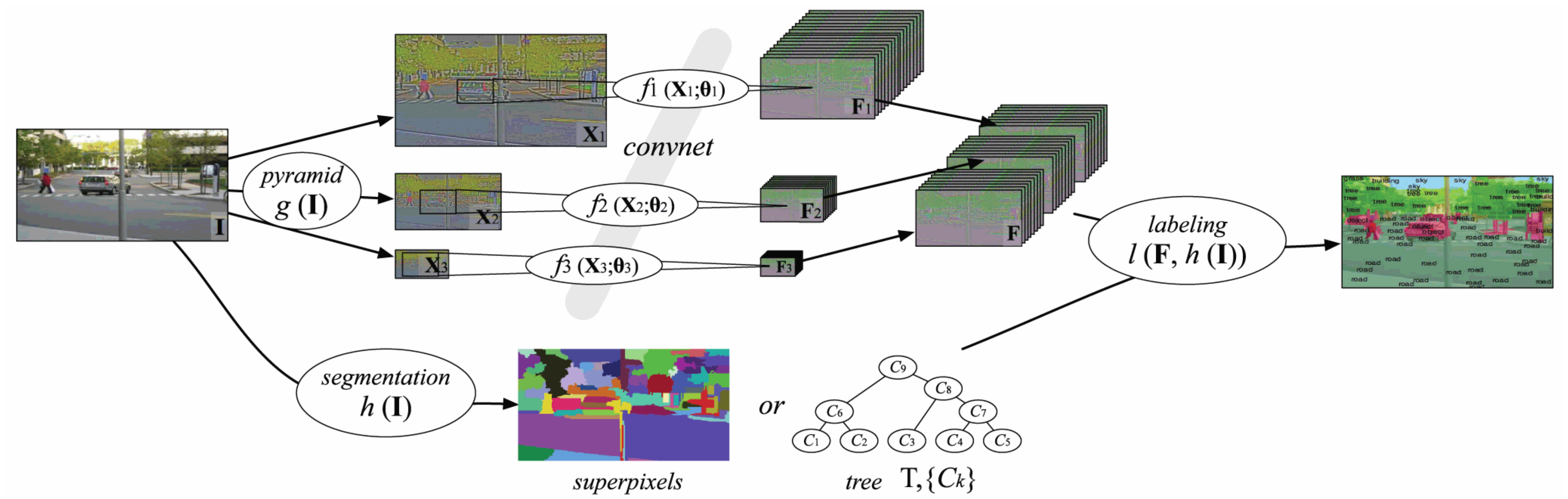

2-dimensional maps

Matan, Ofer, et al. "Multi-digit recognition using a space displacement neural network." Advances in neural information processing systems. 1992.
Wolf, Ralph, and John C. Platt. "Postal address block location using a convolutional locator network." Advances in Neural Information Processing Systems. 1994.

01 Introduction

Semantic Segmentation

Related Work - Dense prediction with convnets



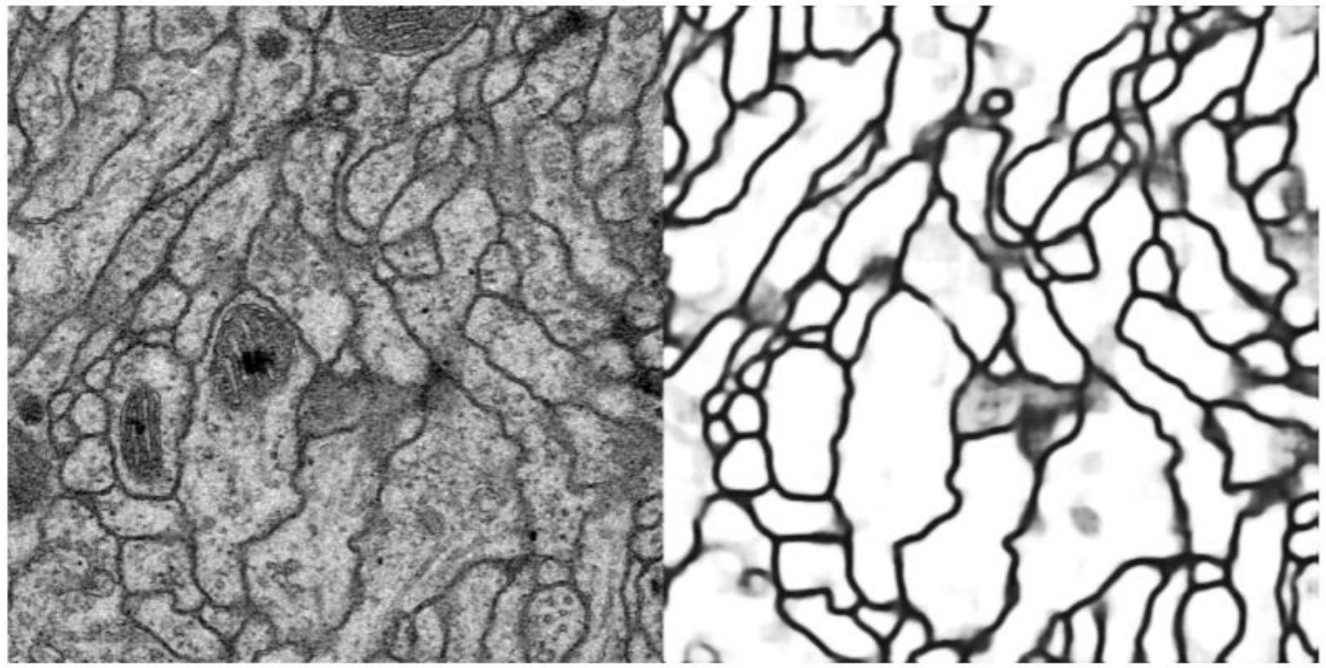
semantic segmentation

Farabet, Clement, et al. "Learning hierarchical features for scene labeling." IEEE transactions on pattern analysis and machine intelligence 35.8 (2013): 1915-1929.

01 Introduction

Semantic Segmentation

Related Work - Dense prediction with convnets



boundary prediction

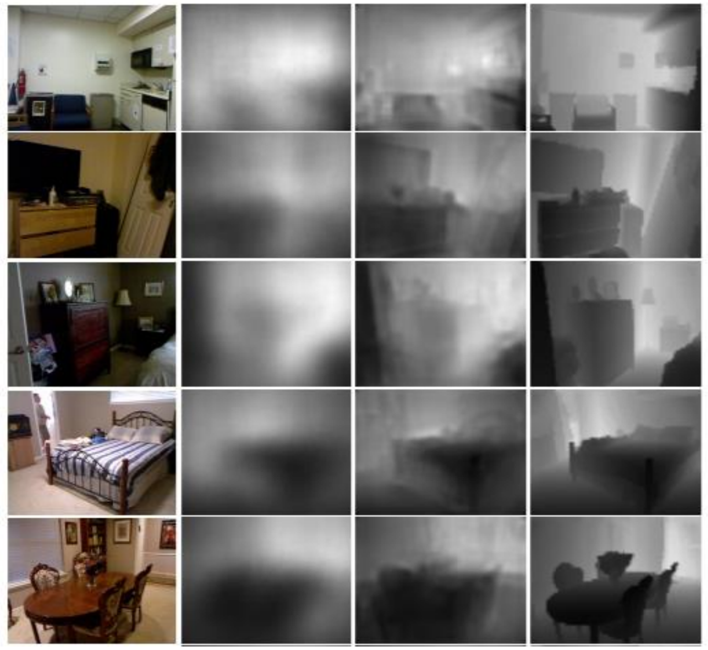
Ciresan, Dan, et al. "Deep neural networks segment neuronal membranes in electron microscopy images." Advances in neural information processing systems. 2012.

01 Introduction

Semantic Segmentation

Related Work - Dense prediction with convnets

image restoration



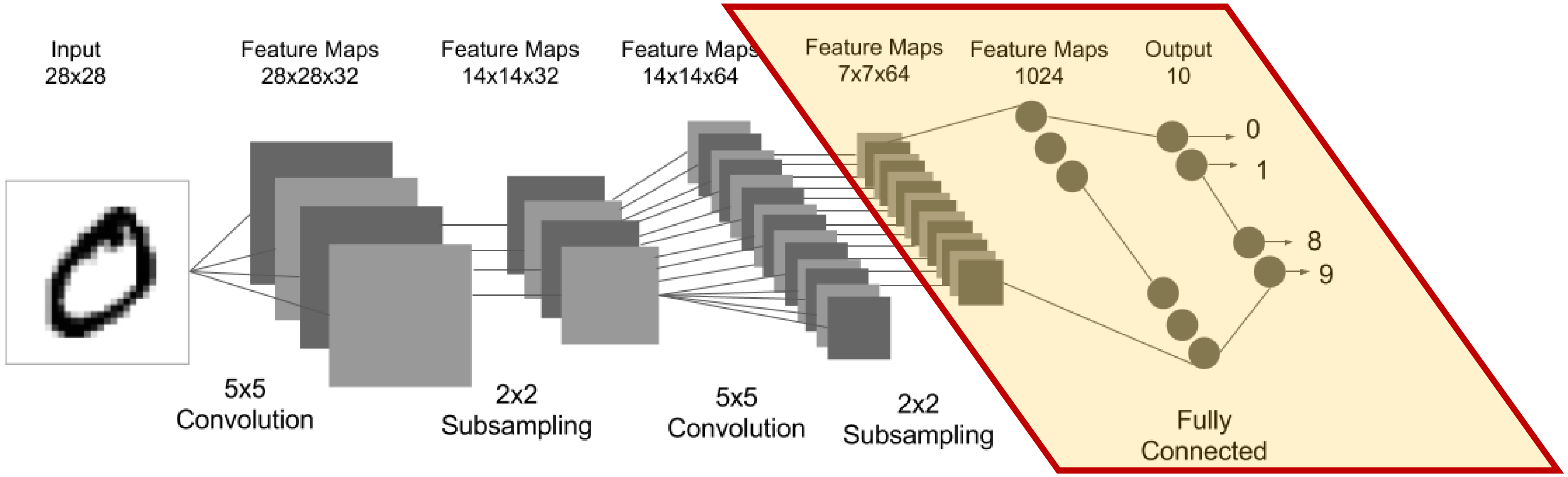
depth estimation

Eigen, David, Dilip Krishnan, and Rob Fergus. "Restoring an image taken through a window covered with dirt or rain." Proceedings of the IEEE International Conference on Computer Vision. 2013.
Eigen, David, Christian Puhrsch, and Rob Fergus. "Depth map prediction from a single image using a multi-scale deep network." Advances in neural information processing systems. 2014.

02 Main Text

Fully Convolutional Network

CNN (Fully Connected Layer) 단점



: 네트워크 뒷단에 분류를 위한 Fully connected layer는 고정된 크기만을 입력으로 받으며, 또한 위치에 대한 정보를 소실

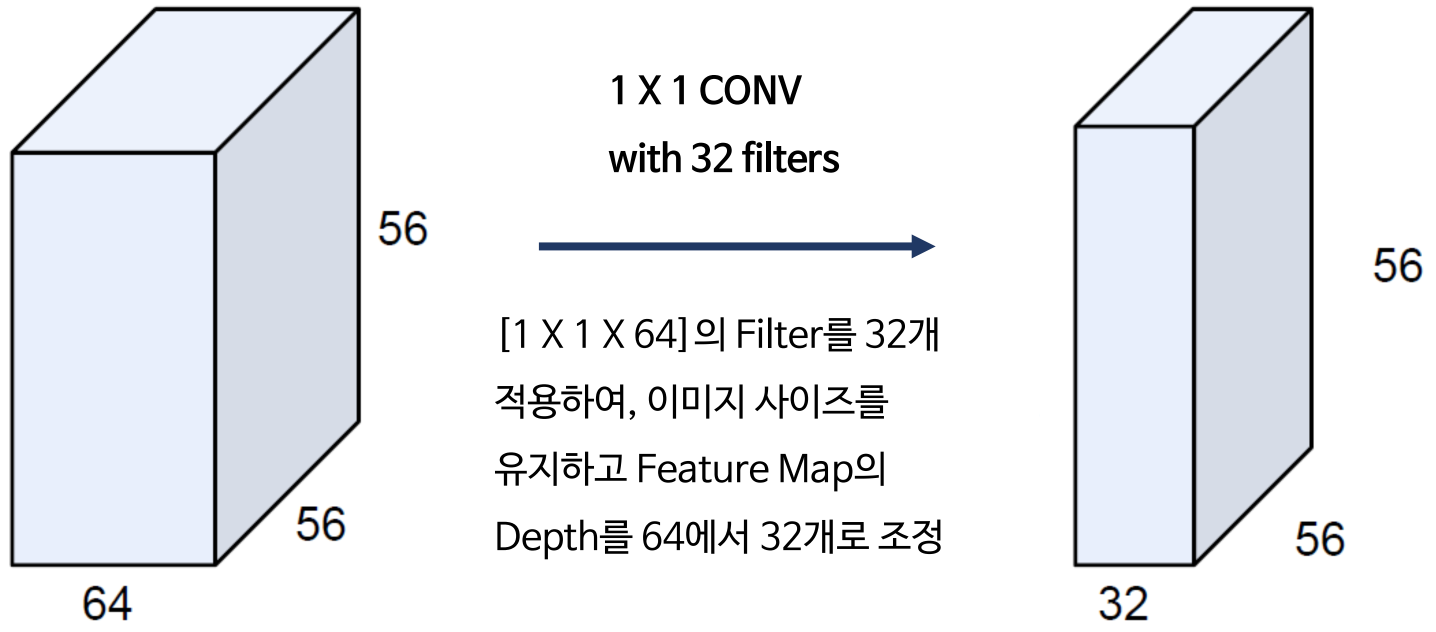
→ 1X1 Convolution Filter를 사용

02 Main Text

Fully Convolutional Network

1X1 Convolution

: [1 X 1 X Kernel] 크기의 Filter를 적용함으로써, 이미지 사이즈는 그대로 하고, Feature Map의 Depth를 조정

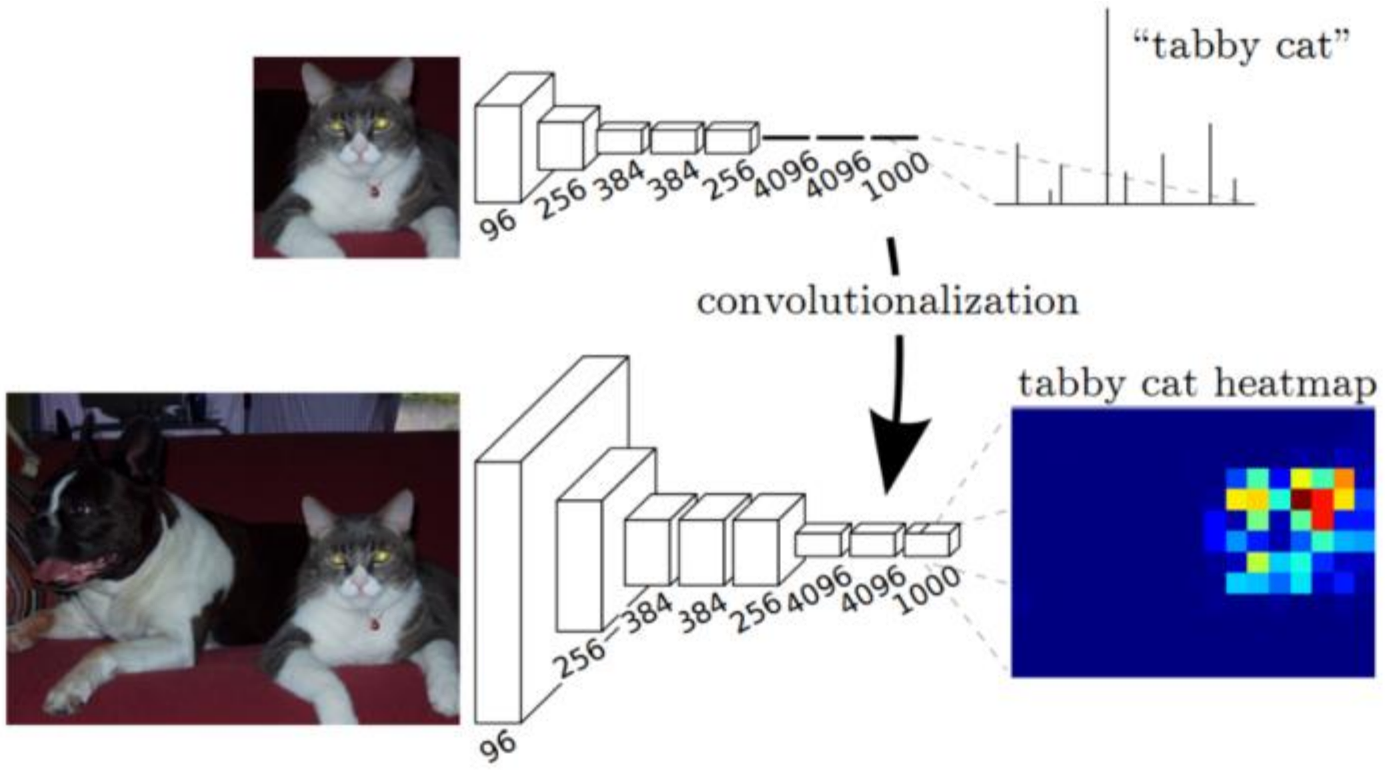


[1 X 1 X 64]의 Filter를 32개 적용하여, 이미지 사이즈를 유지하고 Feature Map의 Depth를 64에서 32개로 조정

Main Text

Fully Convolutional Network

Convolutionalization



: Fully connected Layer로 인해, 위치 정보 소실로 한 이미지에 대해서 하나의 class에 대한 확률을 output

: 1X1 convolution을 통해, 위치 정보를 유지하여 이미지 전체의 모든 픽셀에서 class에 대한 확률을 output

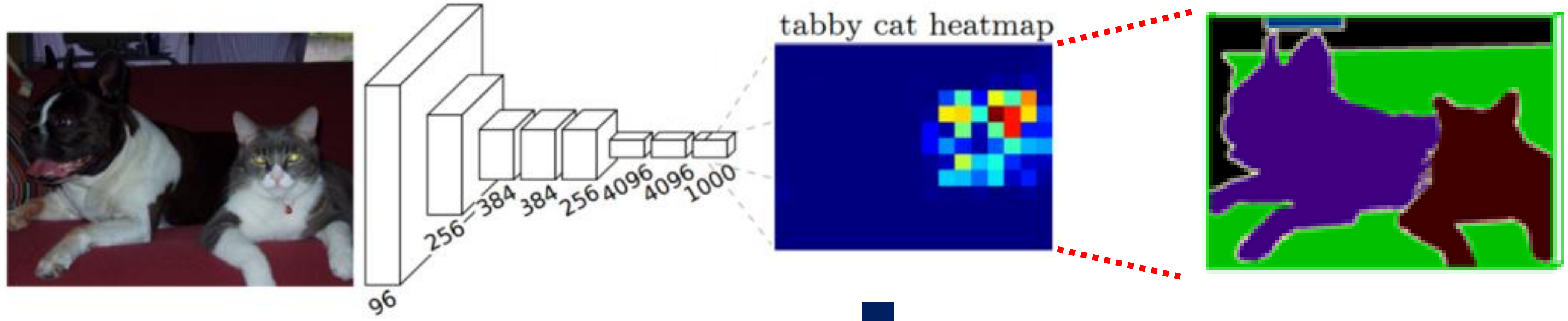
02 Main Text

Segmentation Architecture

Upsampling

정교한(fine) output을 출력할 수 있는 Upsampling 방법 도입

→ Skip Layer



- Problem

: Convolution Layer를 통과하면서,

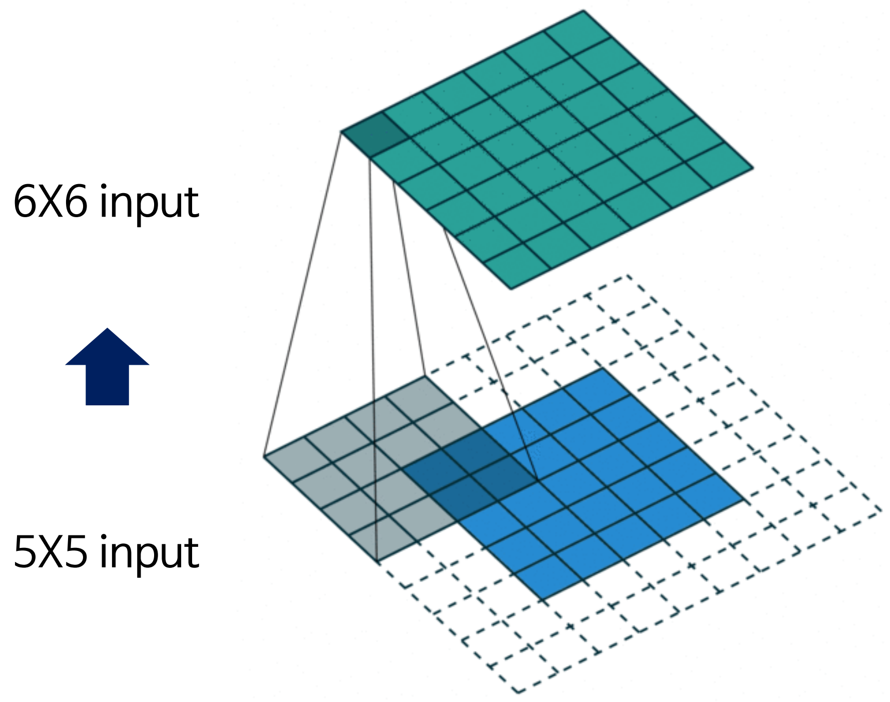
이미지의 픽셀 수가 작아지기 때문에 ouput이 거칠어짐(coarse)

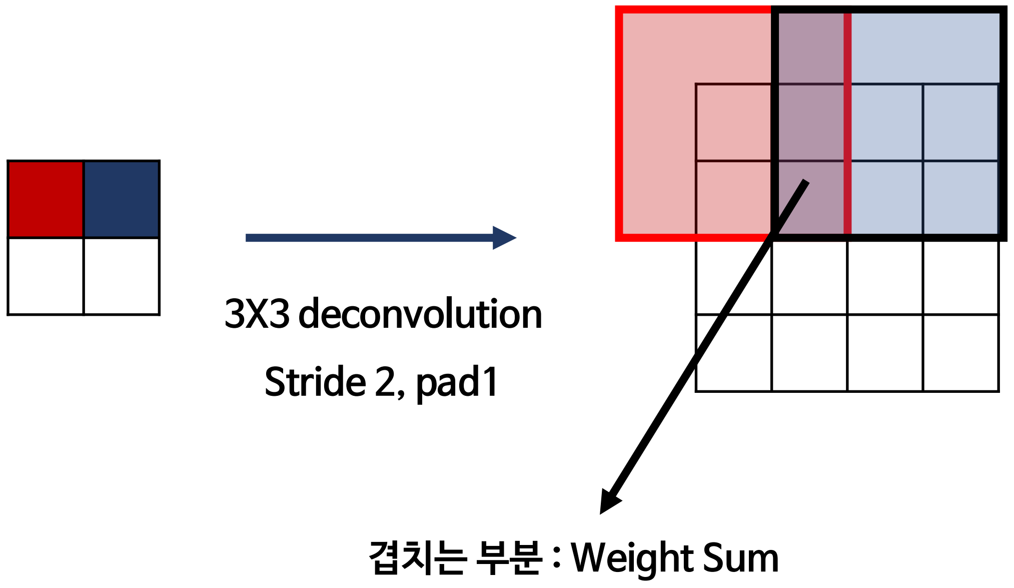
02 Main Text

Segmentation Architecture

Learnable Upsampling

: Convolution Layer를 거치면서 Feature map의 크기가 원래 영상의 크기보다 줄어들었는데, 픽셀 단위로 조밀한 예측을 위해 다시 원래 영상의 크기로 복원하는 과정으로, Deconvolution, Unpooling, Convolution Transpose라고 부르기도 한다.

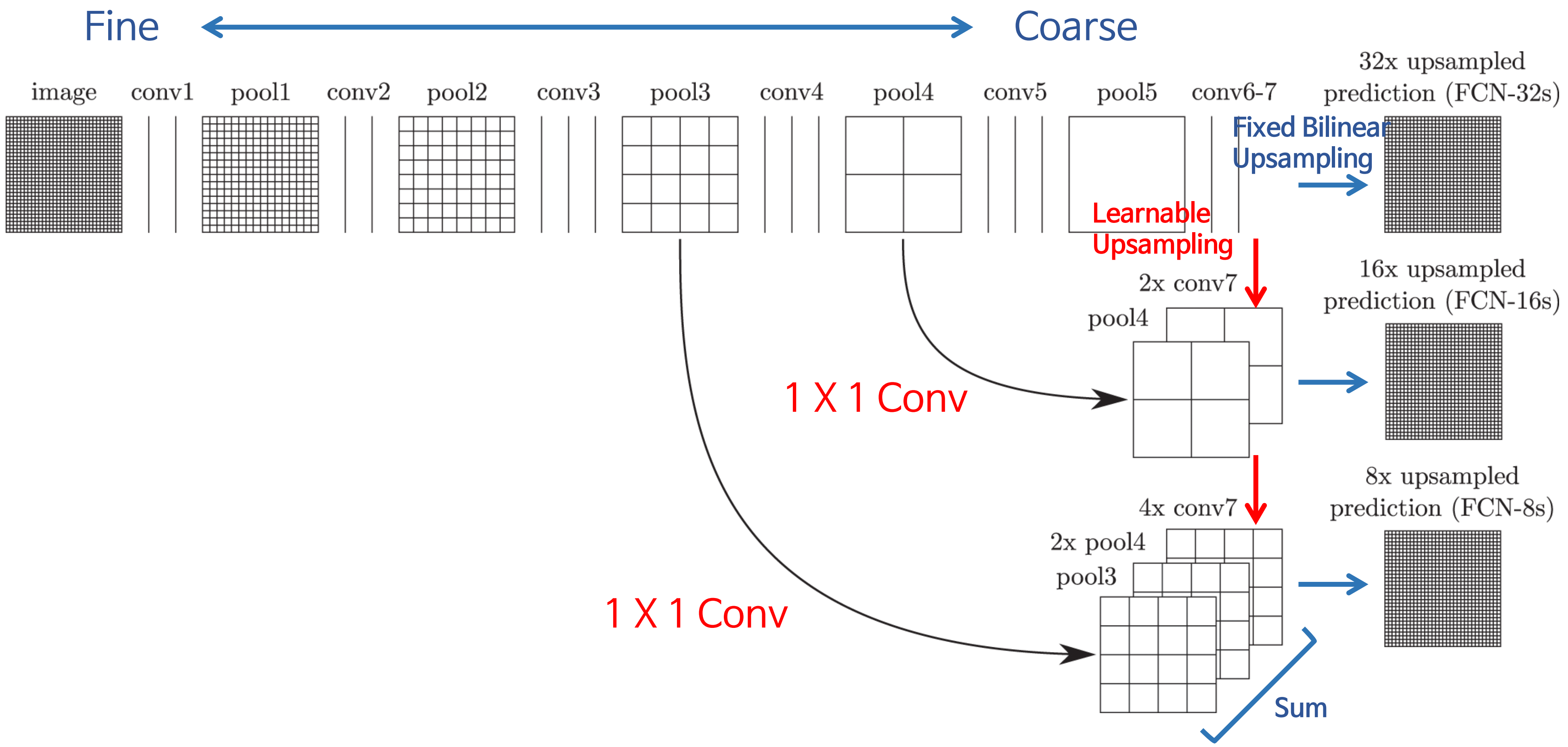




02 Main Text

Segmentation Architecture

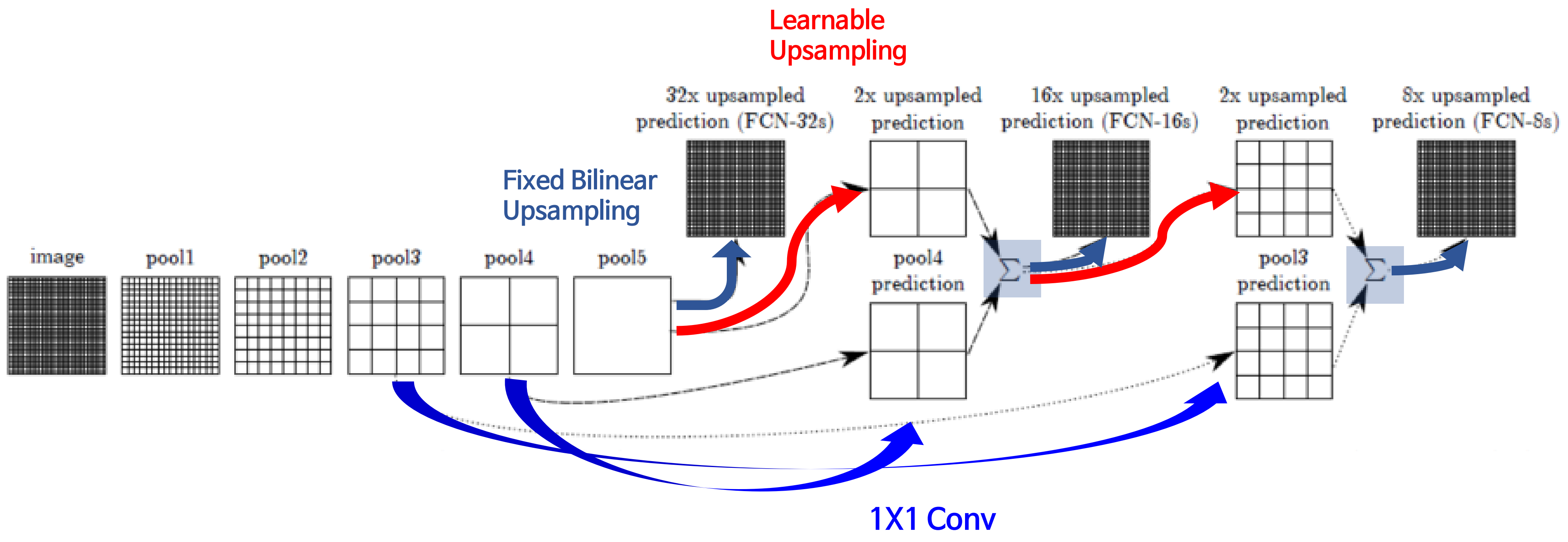
Skip Layer



02 Main Text

Segmentation Architecture

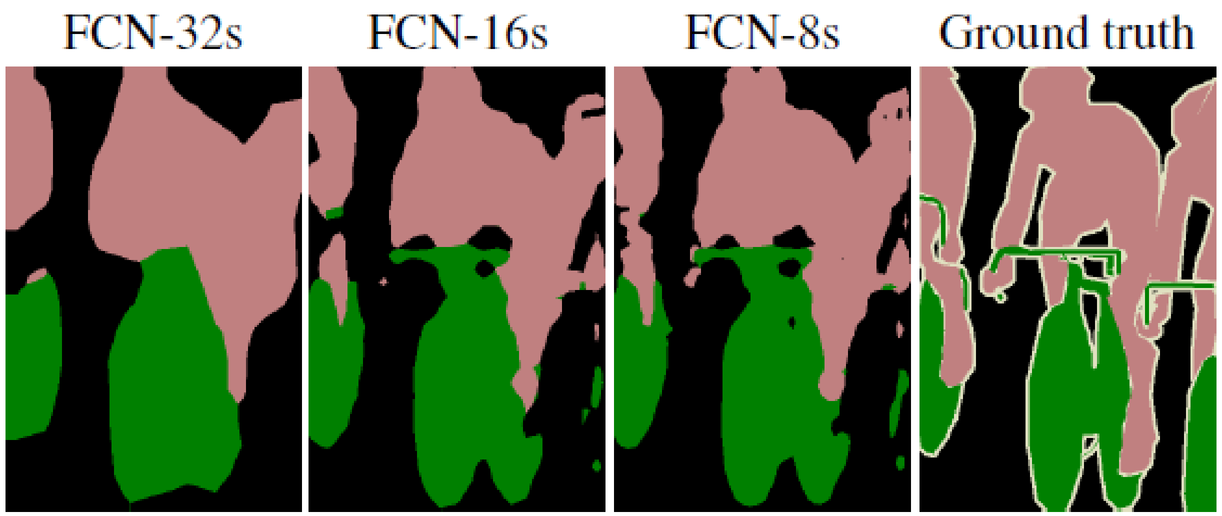
Skip Layer



02 Main Text

Segmentation Architecture

Skip Layer Result



	pixel acc.	mean acc.	mean IU	f.w. IU
FCN-32s-fixed	83.0	59.7	45.4	72.0
FCN-32s	89.1	73.3	59.4	81.4
FCN-16s	90.0	75.7	62.4	83.0
FCN-8s	**90.3**	**75.9**	**62.7**	**83.2**

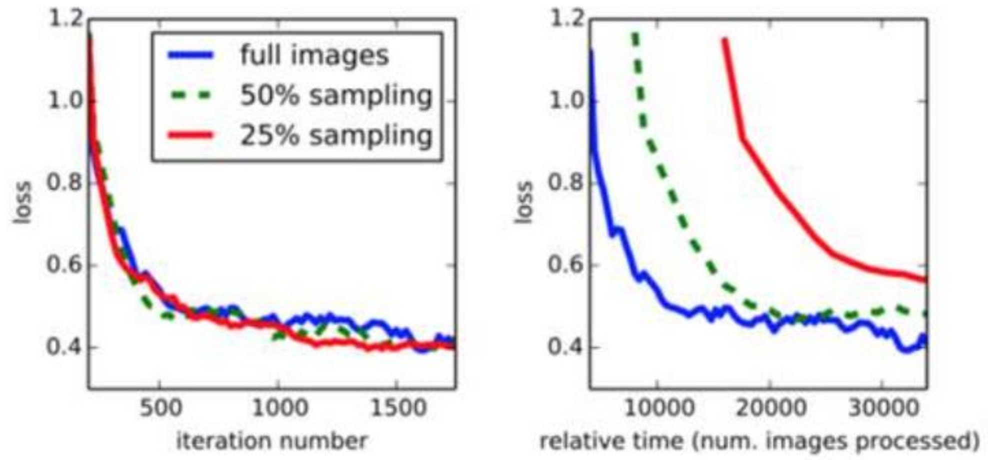
: Stride가 작을수록 (= Sallow Layer를 쓸수록) 더욱 Ground truth의 모양을 정밀하게 잡으며 성능이 좋음을 볼 수 있다.

03 Result

Experiment

Experiment

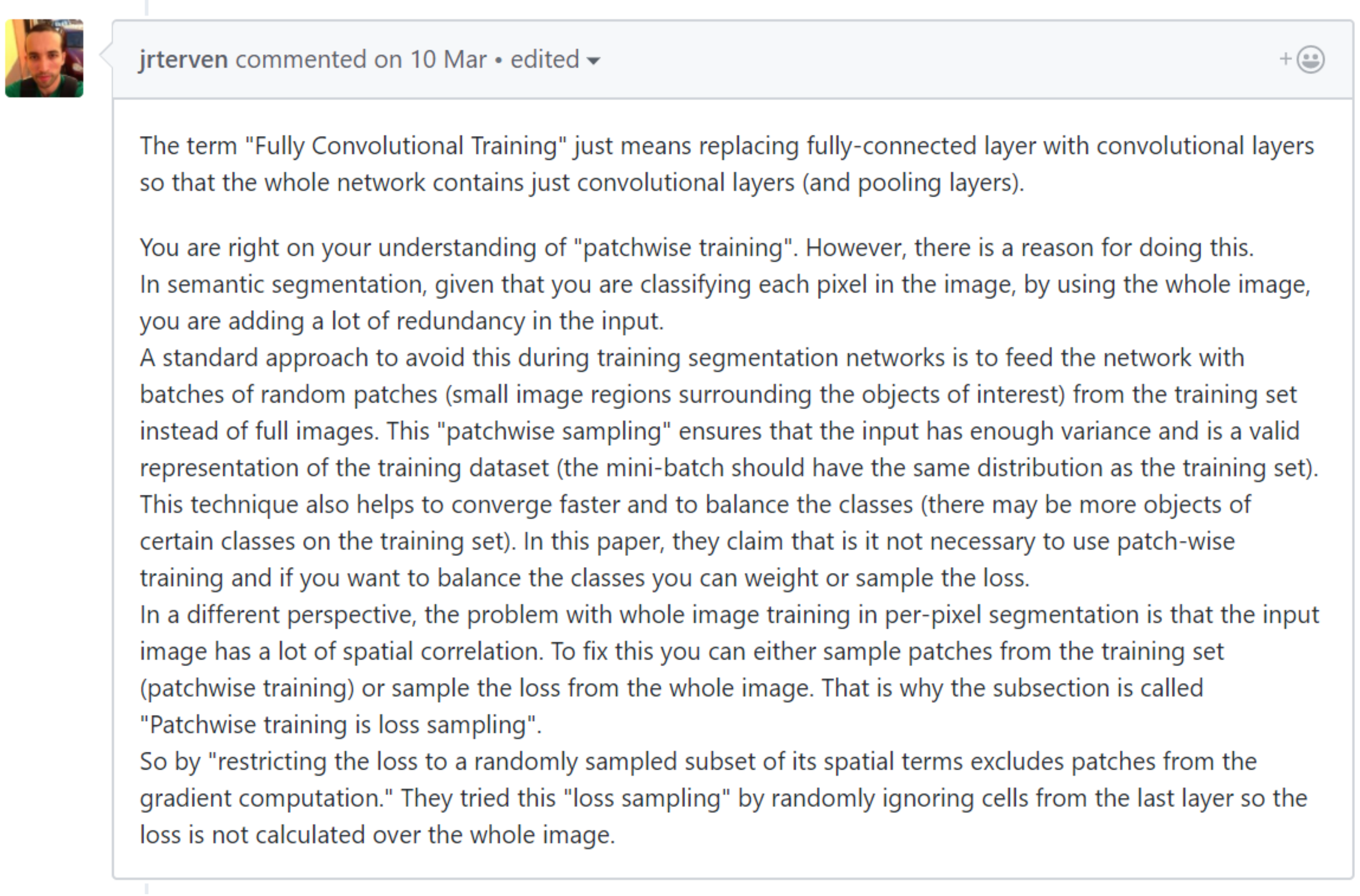
- 기존에 Classification에 효율이 좋은 모델인, AlexNet, VGG16, GoogLeNet을 사용
- Stochastic Gradient Descent with momentum 0.9
- Minibatch size of 20
- Learning rate : $10^{-3}, 10^{-4}, 10^{-5}$
- Patch Sampling을 하지 않은, Full image train



full image와 Sampling의 성능 차이가 거의 없으며,
Sampling을 하게 되면 시간이 오래 걸린다.

03 Result

Experiment



jrterven commented on 10 Mar • edited

The term "Fully Convolutional Training" just means replacing fully-connected layer with convolutional layers so that the whole network contains just convolutional layers (and pooling layers).

You are right on your understanding of "patchwise training". However, there is a reason for doing this.
In semantic segmentation, given that you are classifying each pixel in the image, by using the whole image, you are adding a lot of redundancy in the input.
A standard approach to avoid this during training segmentation networks is to feed the network with batches of random patches (small image regions surrounding the objects of interest) from the training set instead of full images. This "patchwise sampling" ensures that the input has enough variance and is a valid representation of the training dataset (the mini-batch should have the same distribution as the training set).
This technique also helps to converge faster and to balance the classes (there may be more objects of certain classes on the training set). In this paper, they claim that is it not necessary to use patch-wise training and if you want to balance the classes you can weight or sample the loss.
In a different perspective, the problem with whole image training in per-pixel segmentation is that the input image has a lot of spatial correlation. To fix this you can either sample patches from the training set (patchwise training) or sample the loss from the whole image. That is why the subsection is called "Patchwise training is loss sampling".
So by "restricting the loss to a randomly sampled subset of its spatial terms excludes patches from the gradient computation." They tried this "loss sampling" by randomly ignoring cells from the last layer so the loss is not calculated over the whole image.

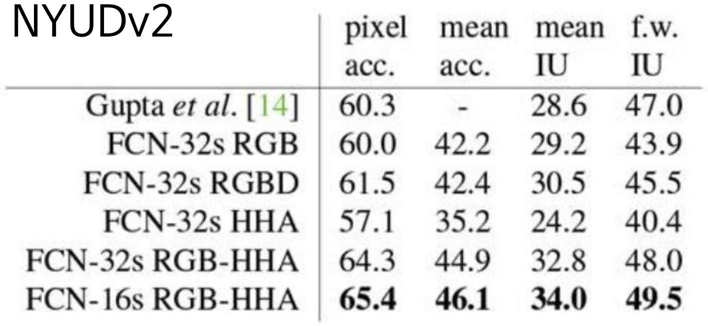
https://github.com/fchollet/keras/issues/5638

03 Result

Result

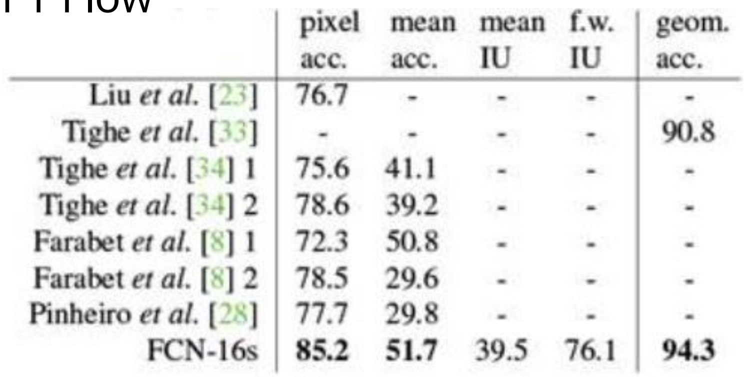
State of the art

(1) NYUDv2



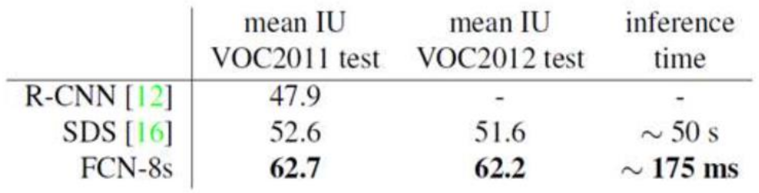
	pixel acc.	mean acc.	mean IU	f.w. IU
Gupta *et al.* [14]	60.3	-	28.6	47.0
FCN-32s RGB	60.0	42.2	29.2	43.9
FCN-32s RGBD	61.5	42.4	30.5	45.5
FCN-32s HHA	57.1	35.2	24.2	40.4
FCN-32s RGB-HHA	64.3	44.9	32.8	48.0
FCN-16s RGB-HHA	**65.4**	**46.1**	**34.0**	**49.5**

(2) SIFT Flow



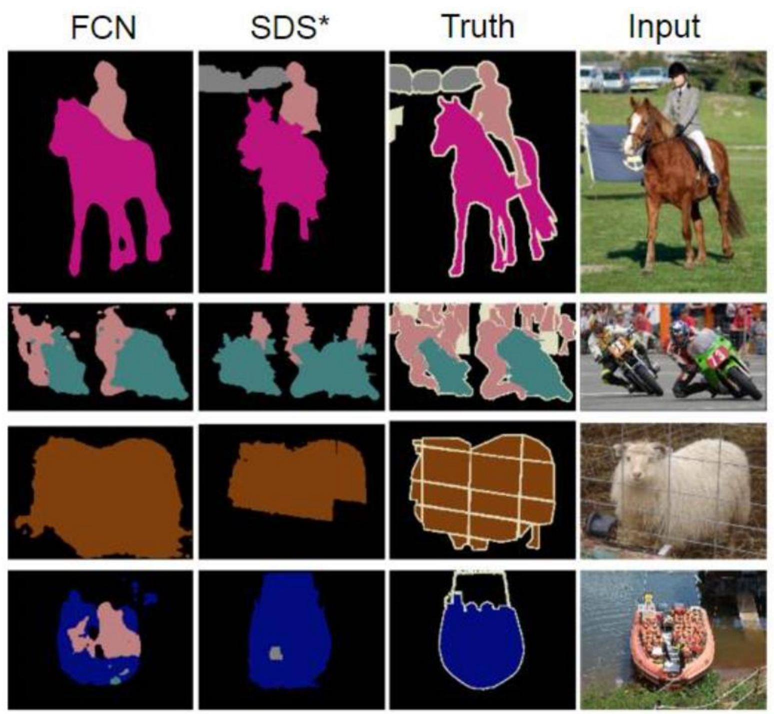
	pixel acc.	mean acc.	mean IU	f.w. IU	geom. acc.
Liu *et al.* [23]	76.7	-	-	-	-
Tighe *et al.* [33]	-	-	-	-	90.8
Tighe *et al.* [34] 1	75.6	41.1	-	-	-
Tighe *et al.* [34] 2	78.6	39.2	-	-	-
Farabet *et al.* [8] 1	72.3	50.8	-	-	-
Farabet *et al.* [8] 2	78.5	29.6	-	-	-
Pinheiro *et al.* [28]	77.7	29.8	-	-	-
FCN-16s	**85.2**	**51.7**	39.5	76.1	**94.3**

(3) PASCAL VOC



	mean IU VOC2011 test	mean IU VOC2012 test	inference time
R-CNN [12]	47.9	-	-
SDS [16]	52.6	51.6	~ 50 s
FCN-8s	**62.7**	**62.2**	**~ 175 ms**

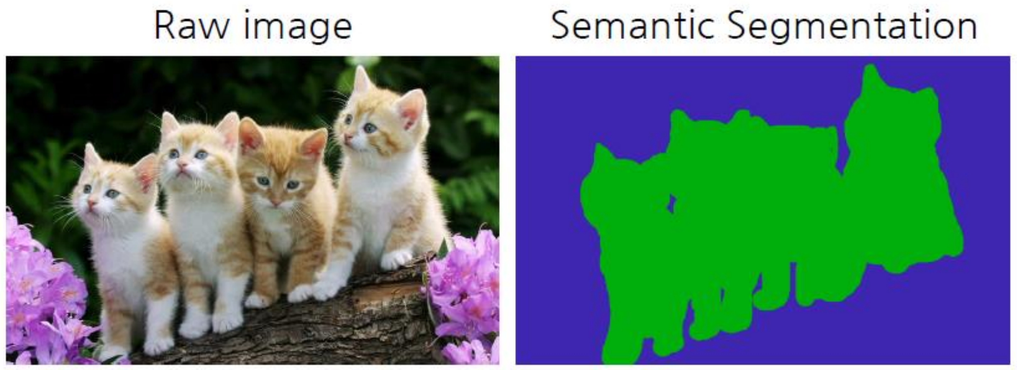
* SDS
: Simultaneous
Detection
and
Segmentation



03 Result

Summary

Semantic Segmentation



: 이미지에서 의미 있는 부분만을 구별하는 것으로,

한 pixel당 하나의 label이 할당되어 있다.

Key Points

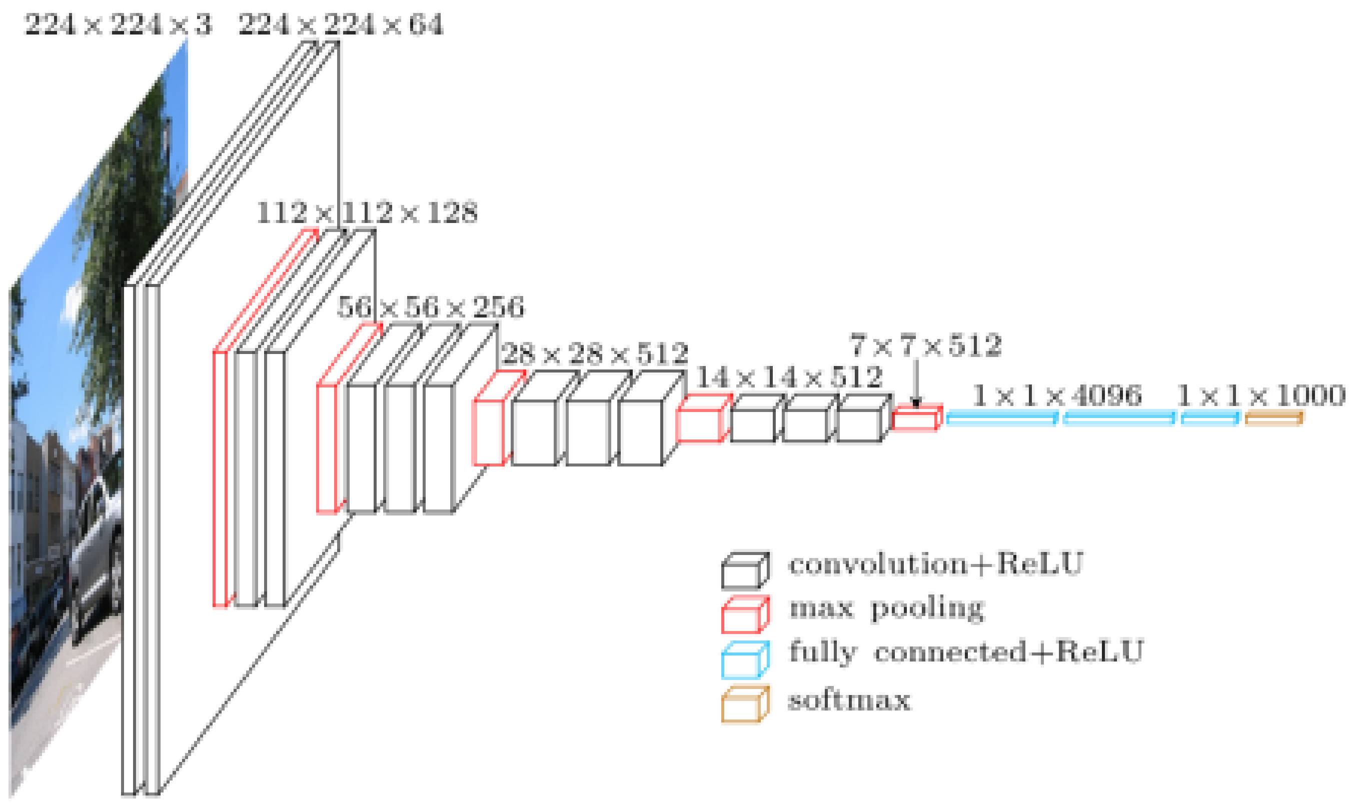
- 일반적인 CNN(Classification)에서 Input(이미지)과 Output(Label)의 형태가 다르다.
  → Fully connected layer 대신, 이미지의 위치적 정보를 유지하도록 1X1 Convolutional Layer를 사용

- Convolution의 과정으로 이미지 크기가 작아져, Upsampling한 결과의 Segmentation의 경계들이 Coarse하다.
  → Skip Layer를 사용하여 경계선을 더욱 세밀하게(fine) 포착

04 Code

FCN VGG16

VGG-Net for classification



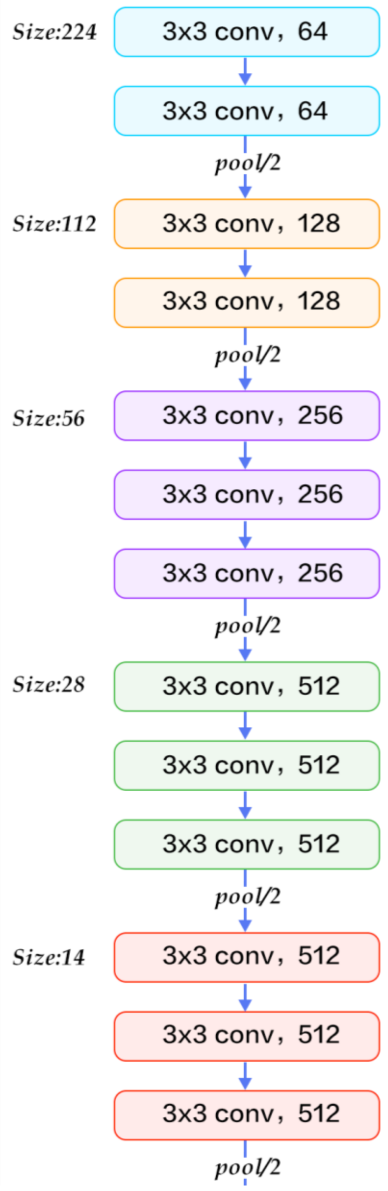
https://github.com/shekkizh/FCN.tensorflow/blob/master/FCN.py

04 Code

FCN VGG16

VGG-Net (Convolution Layer)

```
def vgg_net(weights, image):
  layers = (
    'conv1_1', 'relu1_1', 'conv1_2', 'relu1_2', 'pool1',

    'conv2_1', 'relu2_1', 'conv2_2', 'relu2_2', 'pool2',

    'conv3_1', 'relu3_1', 'conv3_2', 'relu3_2', 'conv3_3', 'relu3_3', 'conv3_4', 'relu3_4', 'pool3',

    'conv4_1', 'relu4_1', 'conv4_2', 'relu4_2', 'conv4_3', 'relu4_3', 'conv4_4', 'relu4_4', 'pool4',

    'conv5_1', 'relu5_1', 'conv5_2', 'relu5_2', 'conv5_3',  'relu5_3', 'conv5_4', 'relu5_4'
  )
```

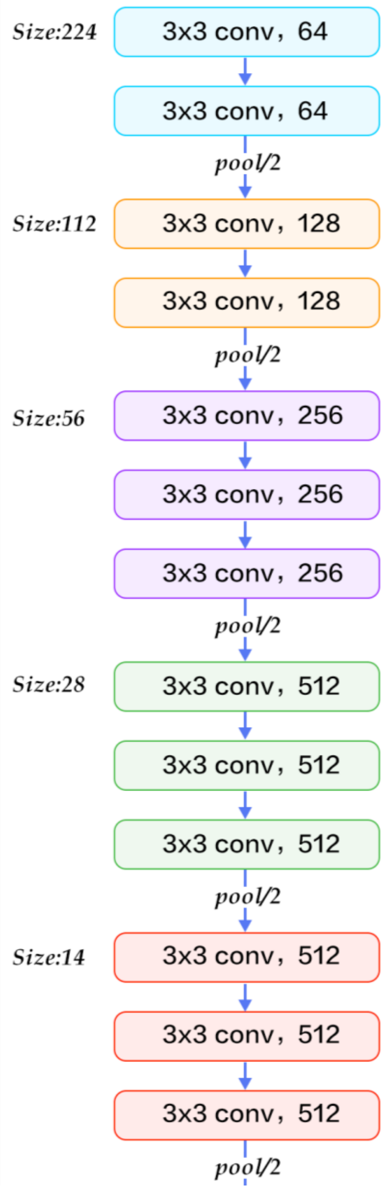


04 Code

FCN VGG16

VGG-Net (Convolution Layer)

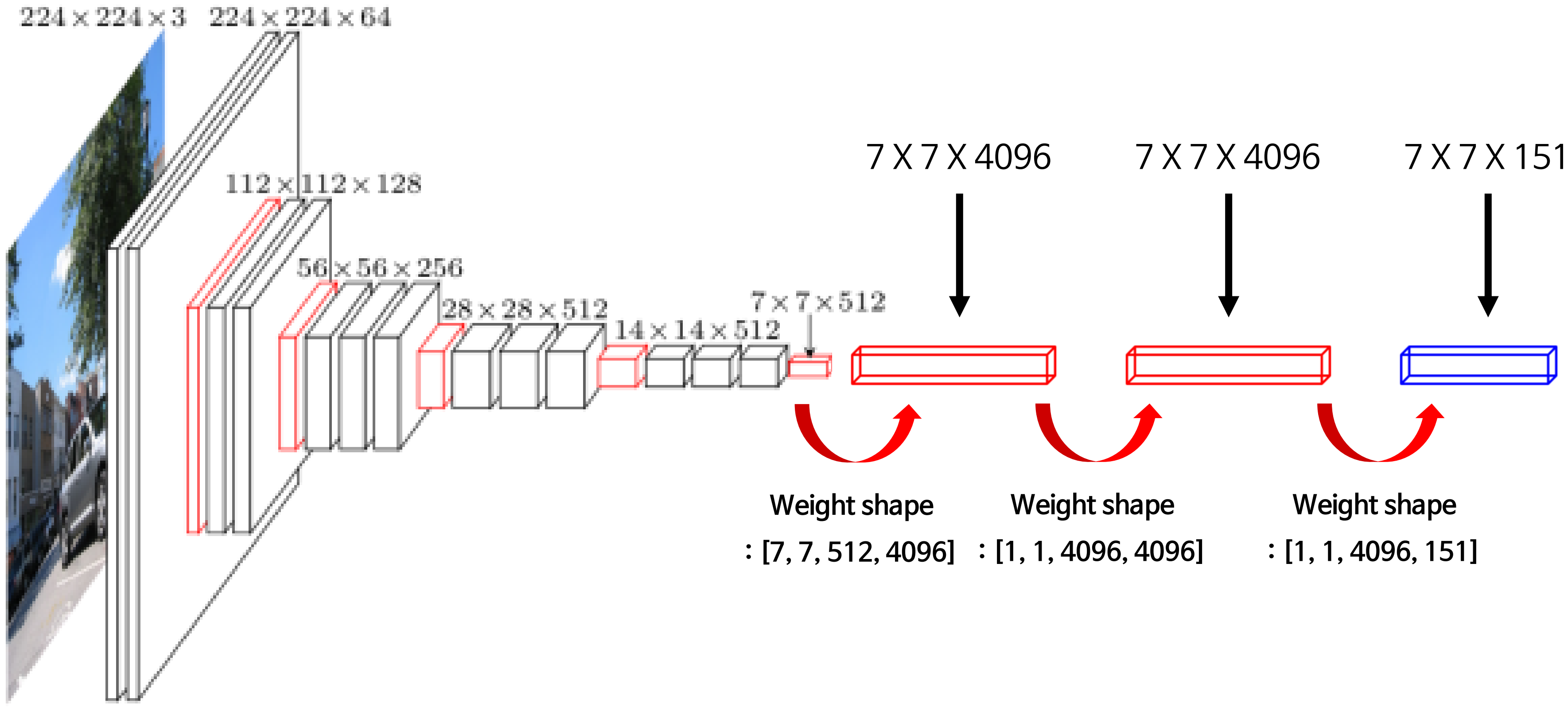
```
net = {}
current = image
for i, name in enumerate(layers):
  kind = name[:4]
  if kind == 'conv':
    kernels, bias = weights[i][0][0][0][0]
    # matconvnet: weights are [width, height, in_channels, out_channels]
    # tensorflow: weights are [height, width, in_channels, out_channels]
    kernels = utils.get_variable(np.transpose(kernels, (1, 0, 2, 3)), name=name + "_w")
    bias = utils.get_variable(bias.reshape(-1), name=name + "_b")
    current = utils.conv2d_basic(current, kernels, bias)
  elif kind == 'relu':
    current = tf.nn.relu(current, name=name)
    if FLAGS.debug:
      utils.add_activation_summary(current)
  elif kind == 'pool':
    current = utils.avg_pool_2x2(current)
  net[name] = current

return net
```



04 Code

FCN VGG16

1 X 1 Convolution Layer



Weight shape
: [7, 7, 512, 4096]

Weight shape
: [1, 1, 4096, 4096]

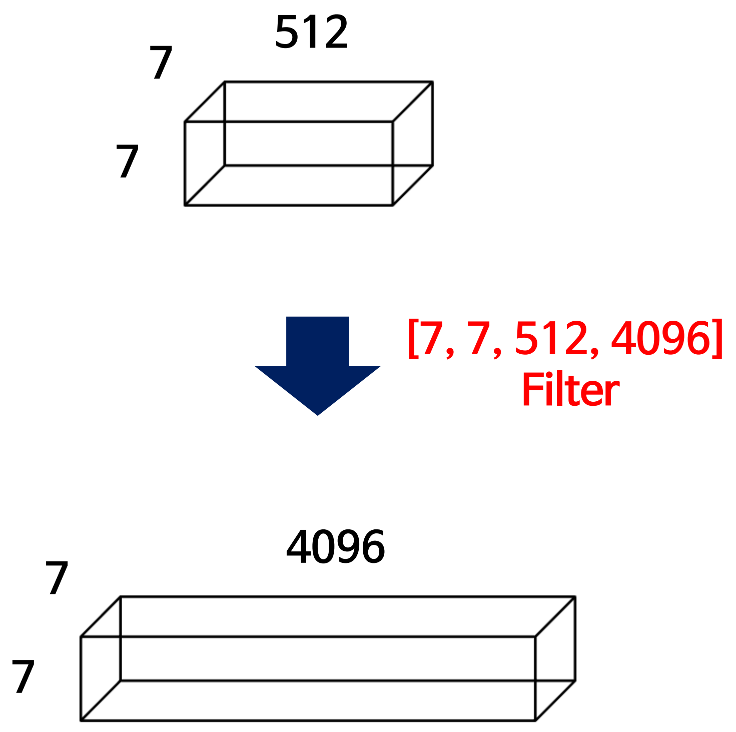
Weight shape
: [1, 1, 4096, 151]

04 Code

FCN VGG16

1 X 1 Convolution Layer

```
with tf.variable_scope("inference"):
    image_net = vgg_net(weights, processed_image)
    conv_final_layer = image_net["conv5_3"]

    pool5 = utils.max_pool_2x2(conv_final_layer)

    W6 = utils.weight_variable([7, 7, 512, 4096], name="W6")
    b6 = utils.bias_variable([4096], name="b6")
    conv6 = utils.conv2d_basic(pool5, W6, b6)
    relu6 = tf.nn.relu(conv6, name="relu6")
    if FLAGS.debug:
        utils.add_activation_summary(relu6)
    relu_dropout6 = tf.nn.dropout(relu6, keep_prob=keep_prob)
```



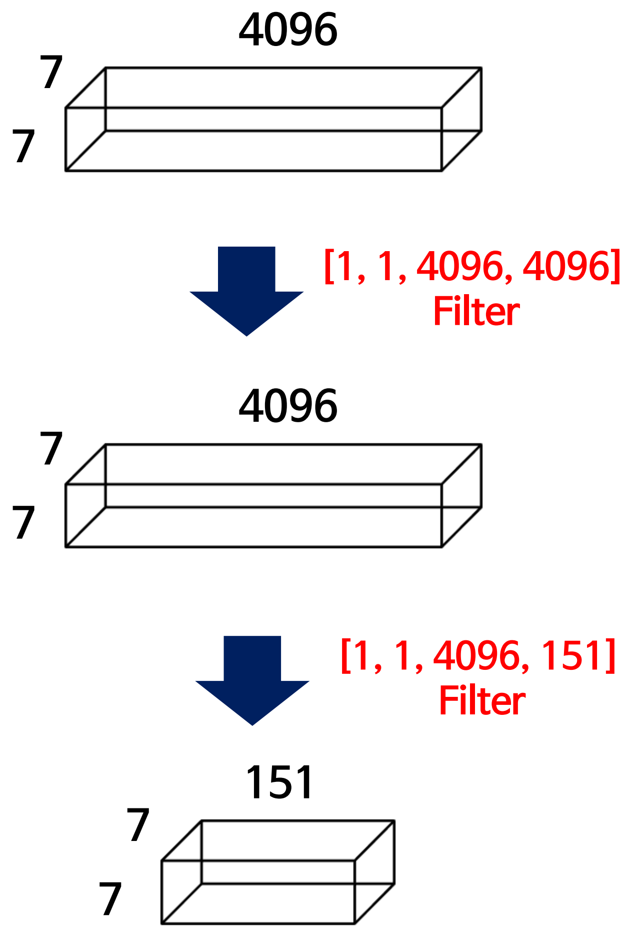
04

Code

FCN VGG16

1 X 1 Convolution Layer

```
W7 = utils.weight_variable([1, 1, 4096, 4096], name="W7")
 b7 = utils.bias_variable([4096], name="b7")
 conv7 = utils.conv2d_basic(relu_dropout6, W7, b7)
 relu7 = tf.nn.relu(conv7, name="relu7")

if FLAGS.debug:
    utils.add_activation_summary(relu7)
 relu_dropout7 = tf.nn.dropout(relu7, keep_prob=keep_prob)

 W8 = utils.weight_variable([1, 1, 4096, NUM_OF_CLASSESS], name="W8")
 b8 = utils.bias_variable([NUM_OF_CLASSESS], name="b8")
 conv8 = utils.conv2d_basic(relu_dropout7, W8, b8)
```

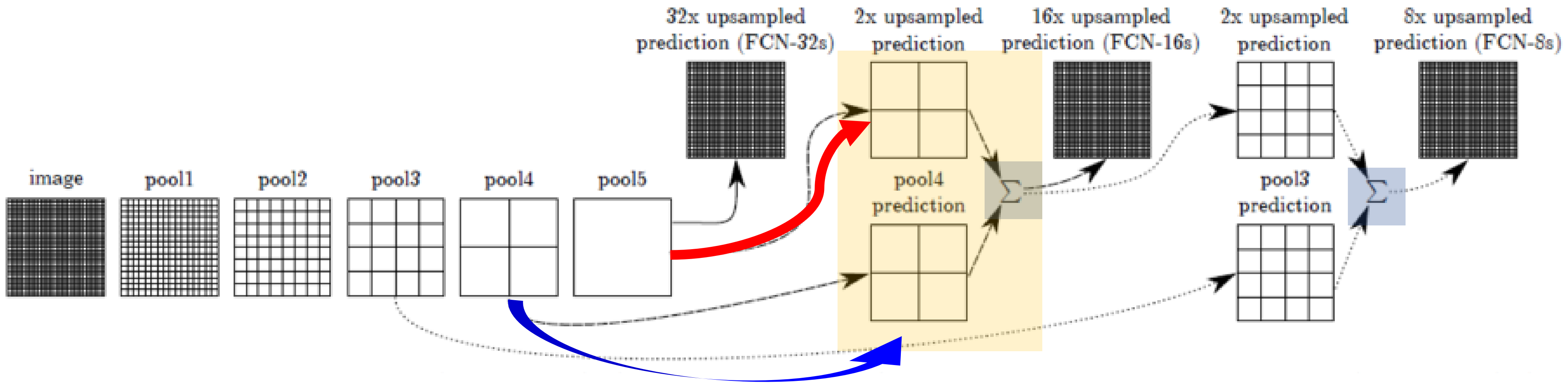


04 Code

FCN VGG16

Upsampling

```
deconv_shape1 = image_net["pool4"].get_shape()
W_t1 = utils.weight_variable([4, 4, deconv_shape1[3].value, NUM_OF_CLASSESS], name="W_t1")
b_t1 = utils.bias_variable([deconv_shape1[3].value], name="b_t1")
conv_t1 = utils.conv2d_transpose_strided(conv8, W_t1, b_t1, output_shape=tf.shape(image_net["pool4"]))
fuse_1 = tf.add(conv_t1, image_net["pool4"], name="fuse_1")
```

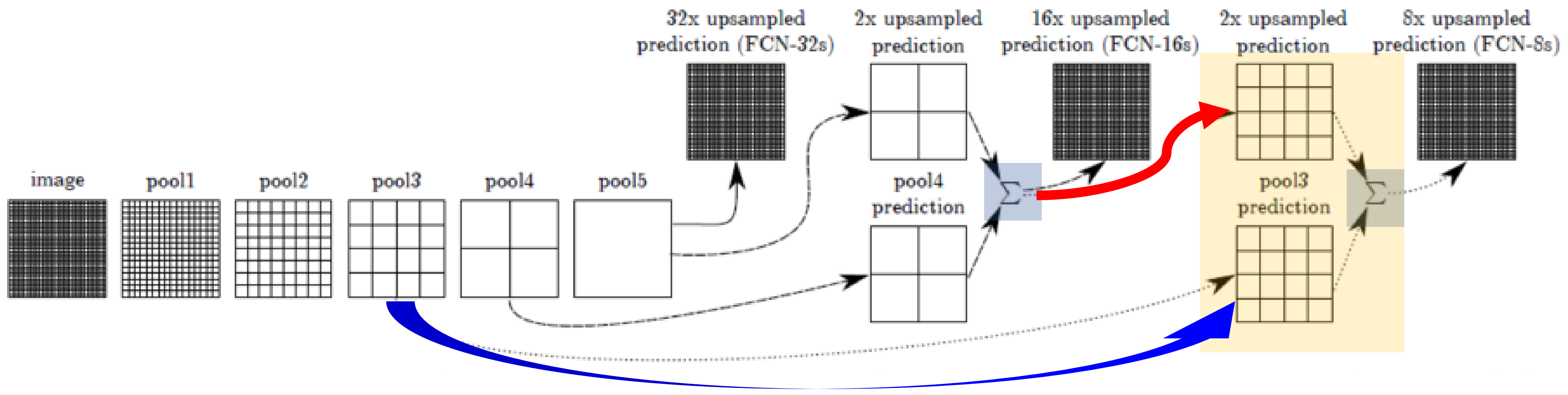


04 Code

FCN VGG16

Upsampling

```
deconv_shape2 = image_net["pool3"].get_shape()
W_t2 = utils.weight_variable([4, 4, deconv_shape2[3].value, deconv_shape1[3].value], name="W_t2")
b_t2 = utils.bias_variable([deconv_shape2[3].value], name="b_t2")
conv_t2 = utils.conv2d_transpose_strided(fuse_1, W_t2, b_t2, output_shape=tf.shape(image_net["pool3"]))
fuse_2 = tf.add(conv_t2, image_net["pool3"], name="fuse_2")
```

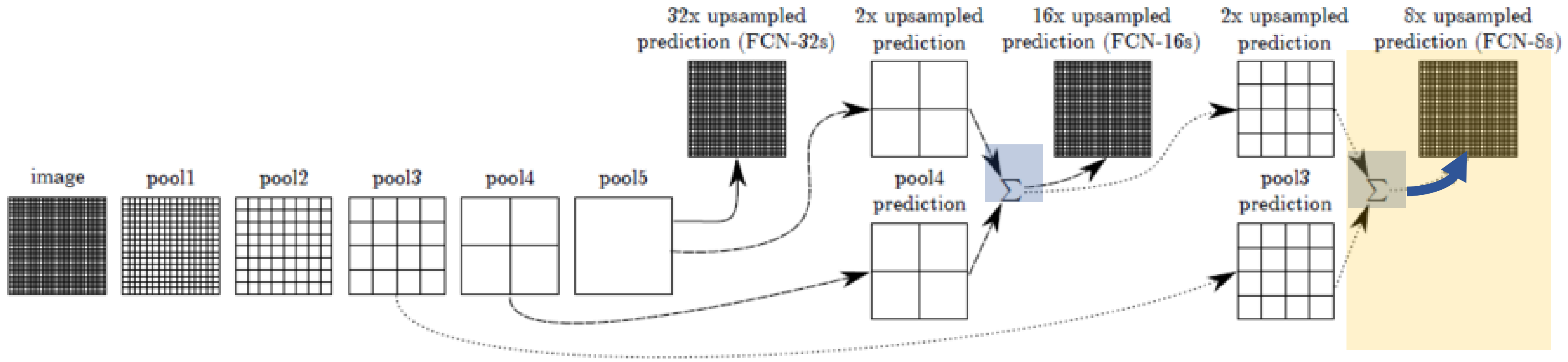


04 Code

FCN VGG16

Upsampling

```
shape = tf.shape(image)
deconv_shape3 = tf.stack([shape[0], shape[1], shape[2], NUM_OF_CLASSESS])
W_t3 = utils.weight_variable([16, 16, NUM_OF_CLASSESS, deconv_shape2[3].value], name="W_t3")
b_t3 = utils.bias_variable([NUM_OF_CLASSESS], name="b_t3")
conv_t3 = utils.conv2d_transpose_strided(fuse_2, W_t3, b_t3, output_shape=deconv_shape3, stride=8)

annotation_pred = tf.argmax(conv_t3, dimension=3, name="prediction")
```


32x upsampled prediction (FCN-32s)
2x upsampled prediction
16x upsampled prediction (FCN-16s)
2x upsampled prediction
8x upsampled prediction (FCN-8s)
image
pool1
pool2
pool3
pool4
pool5
pool4 prediction
pool3 prediction
∑
∑