[Image Processing Team]

Fully Convolutional Networks for Semantic Segmentation

의 최초논문

발표자: 허다운

팀원: 박수빈, 서지연, 우성한, 장소진, 허다운

2020.01.17.

Contents

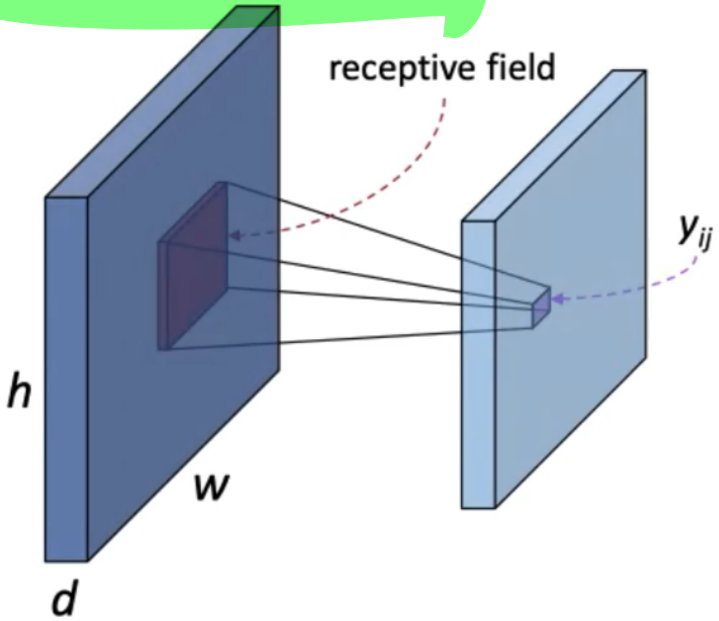
Introduction

- Key insight
  - To build "fully convolutional" networks that take input of arbitrary size and produce correspondingly-sized output with efficient inference and learning

- Characteristic
  - Defined the space of fully convolutional networks
    - Used classification model (supervised pre-trained model; conducted fine-tuning by using trained representation) and applied fully convolutional network to maintain corresponding spatial dimension of input data
  - Defined a skip architecture
    - Combines deep, coarse, semantic information and shallow, fine, appearance information

Fully Convolutional Network

- Convolutional Network



$$\mathbf{y}_{ij} = f_{ks}\left(\{\mathbf{x}_{si+\delta i, sj+\delta j}\}_{0 \le \delta i, \delta j \le k}\right)$$

- x_{ij}: data vector at location (i, j)
- y_{ij}: following layer
- k: kernel size
- s: stride or subsampling factor
- f_{ks}: determines the layer type
  (e.g. multiplication for convolution or average pooling, a spatial max for max pooling, an elementwise nonlinearity for an activation function, etc.)

An FCN naturally operates on an input of any size, and produces an output of corresponding (possibly resampled) spatial dimensions

Fully Convolutional Network

- Adapting classifiers for dense prediction

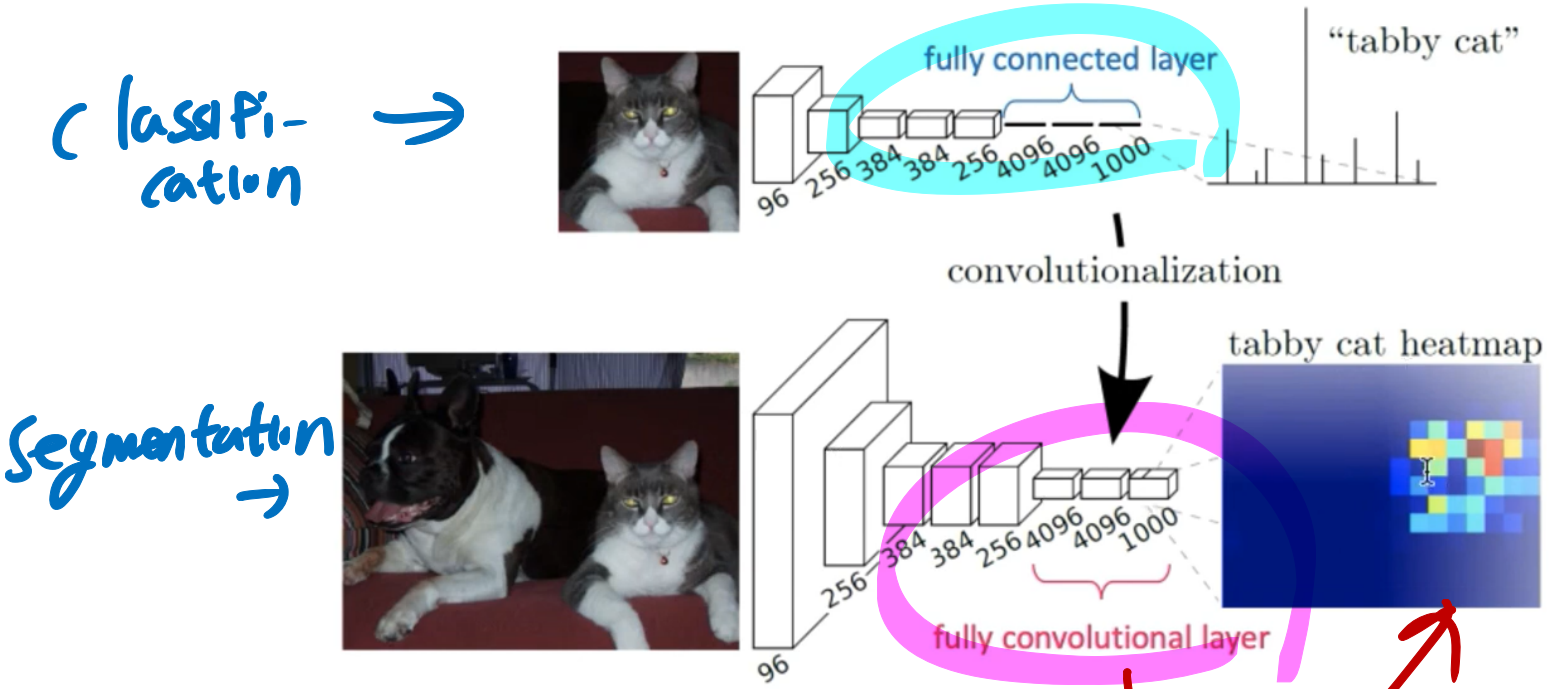


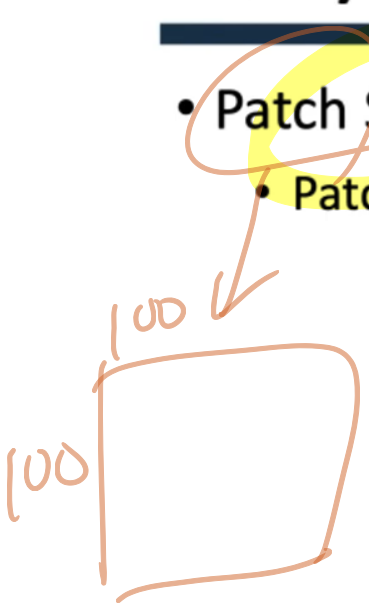
Figure 2. Transforming fully connected layers into convolution layers enables a classification net to output a heatmap. Adding layers and a spatial loss produces an efficient machine for end-to-end dense learning

위치정보유지

https://stackoverflow.com/questions/42636685/patch-wise-training-and-fully-convolutional-training-in-fcn

Fully Convolutional Network

- Patch Sampling
  - Patchwise training is loss sampling



그림의 설명을 때 training

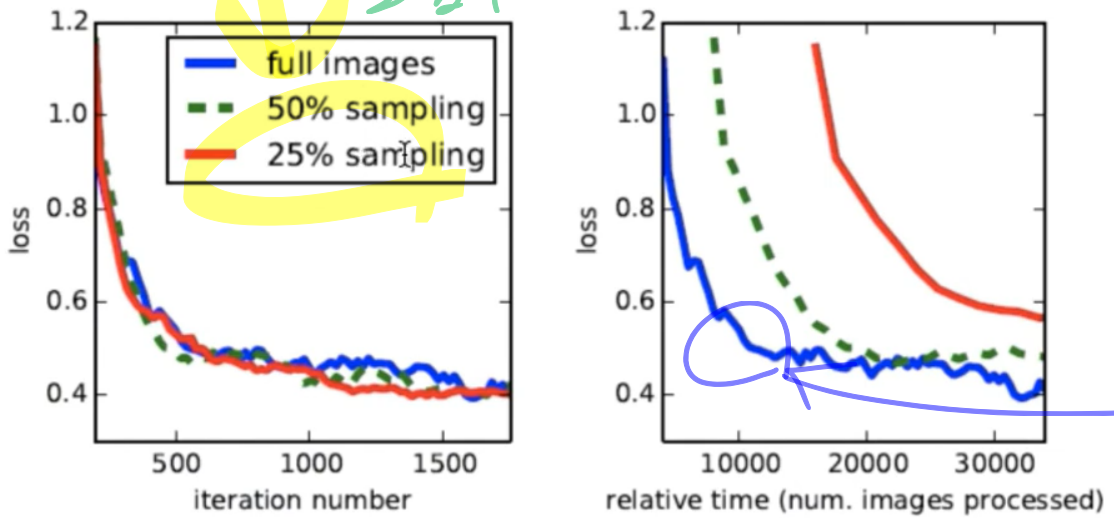
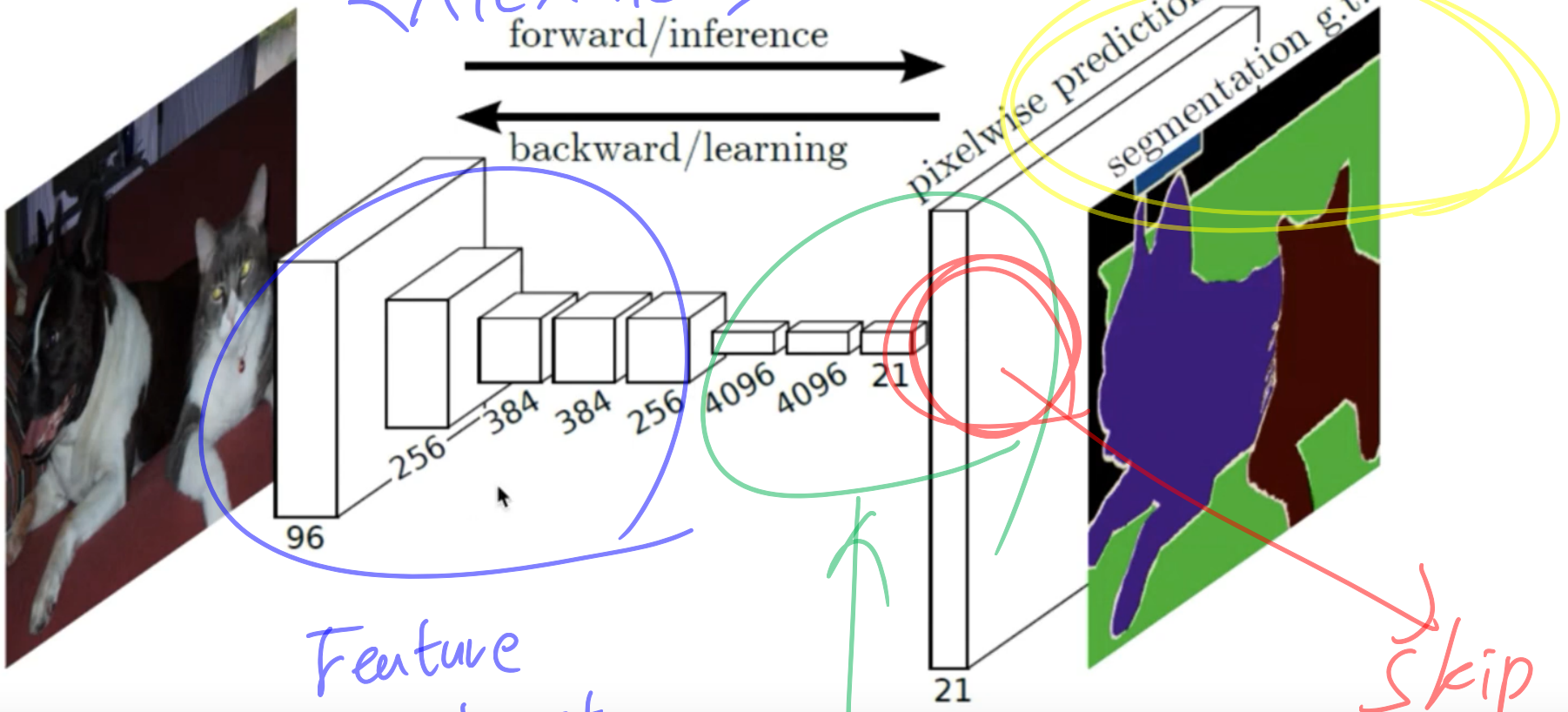


Figure 5. Training on whole images is just as effective as sampling patches, but results in faster (wall time) convergence by making more efficient use of data. Left shows the effect of sampling on convergence rate for a fixed expected batch size, while right plots the same by relative wall time.

비슷함.

patch Image 사용 X

Overall Network Structure of FCN



base code 3개.

Classification Nets for feature extraction

- Compared results of utilized classification networks for feature extraction

	FCN-AlexNet	FCN-VGG16	FCN-GoogLeNet[4]
mean IU	39.8	**56.0**	42.5
forward time	50 ms	210 ms	59 ms
conv. layers	8	16	22
parameters	57M	134M	6M
rf size	355	404	907
max stride	32	32	32

But

Table 1. We adapt and extend three classification convnets. We compare performance by mean intersection over union on the validation set of PASCAL VOC 2011 and by inference time (averaged over 20 trials for a 500 500 input on an NVIDIA Tesla K40c). We detail the architecture of the adapted nets with regard to dense prediction: number of parameter layers, receptive field size of output units, and the coarsest stride within the net. (These numbers give the best performance obtained at a fixed learning rate, not best performance possible.)

Classification Nets for feature extraction

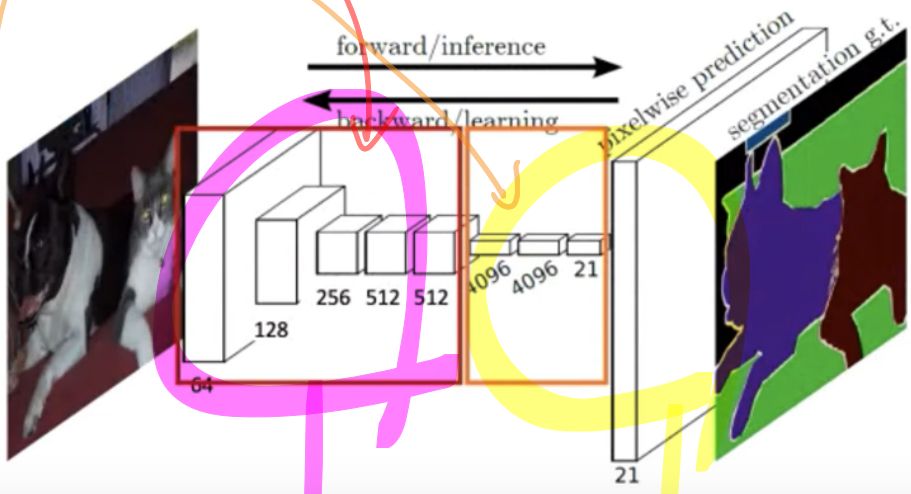
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
					maxpool
					FC-4096
					FC-4096
					FC-1000
					soft-max

- VGG network

: adapted network layers

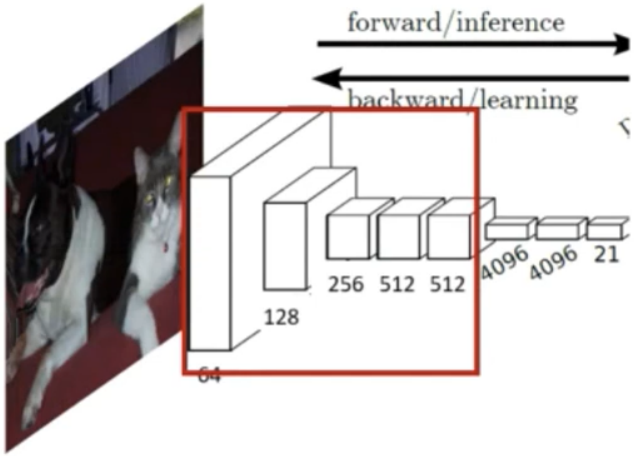
: modified to fully convolutional network

VGG 19 사용



FCN Code (1/5)

- VGG19 network

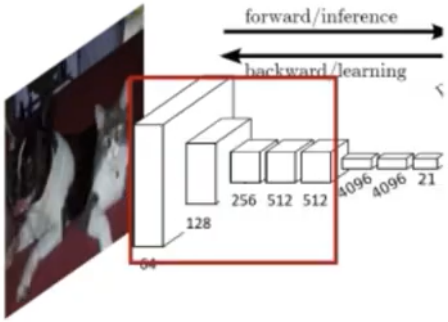



```
def vgg_net(weights, image):
    layers = (
        'conv1_1', 'relu1_1', 'conv1_2', 'relu1_2', 'pool1',

        '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'
    )
```

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

FCN Code (2/5)

- VGG19 network

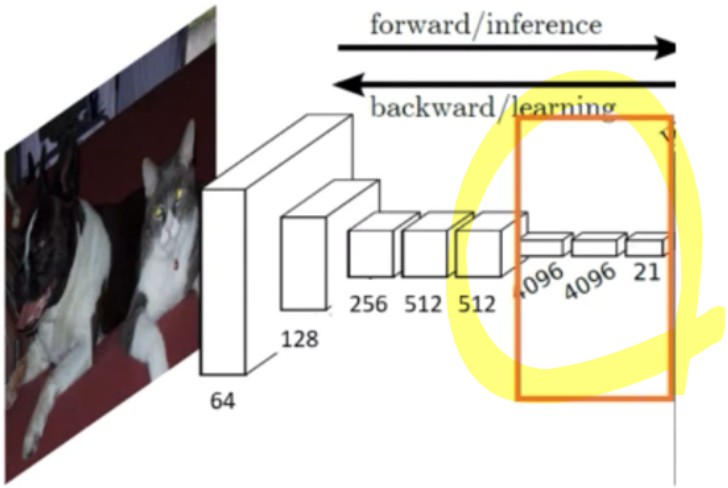



```
net = {}
current = image
for i, name in enumerate(layers):
    kind = name[:4]
    if kind == 'conv':
        kernels, bias = weights[i][0][0][0][0]
        # matconvnet: weights are [width, height, in_channels, out_channels]
        # tensorflow: weights are [height, width, in_channels, out_channels]
        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
```

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

FCN Code (3/5)

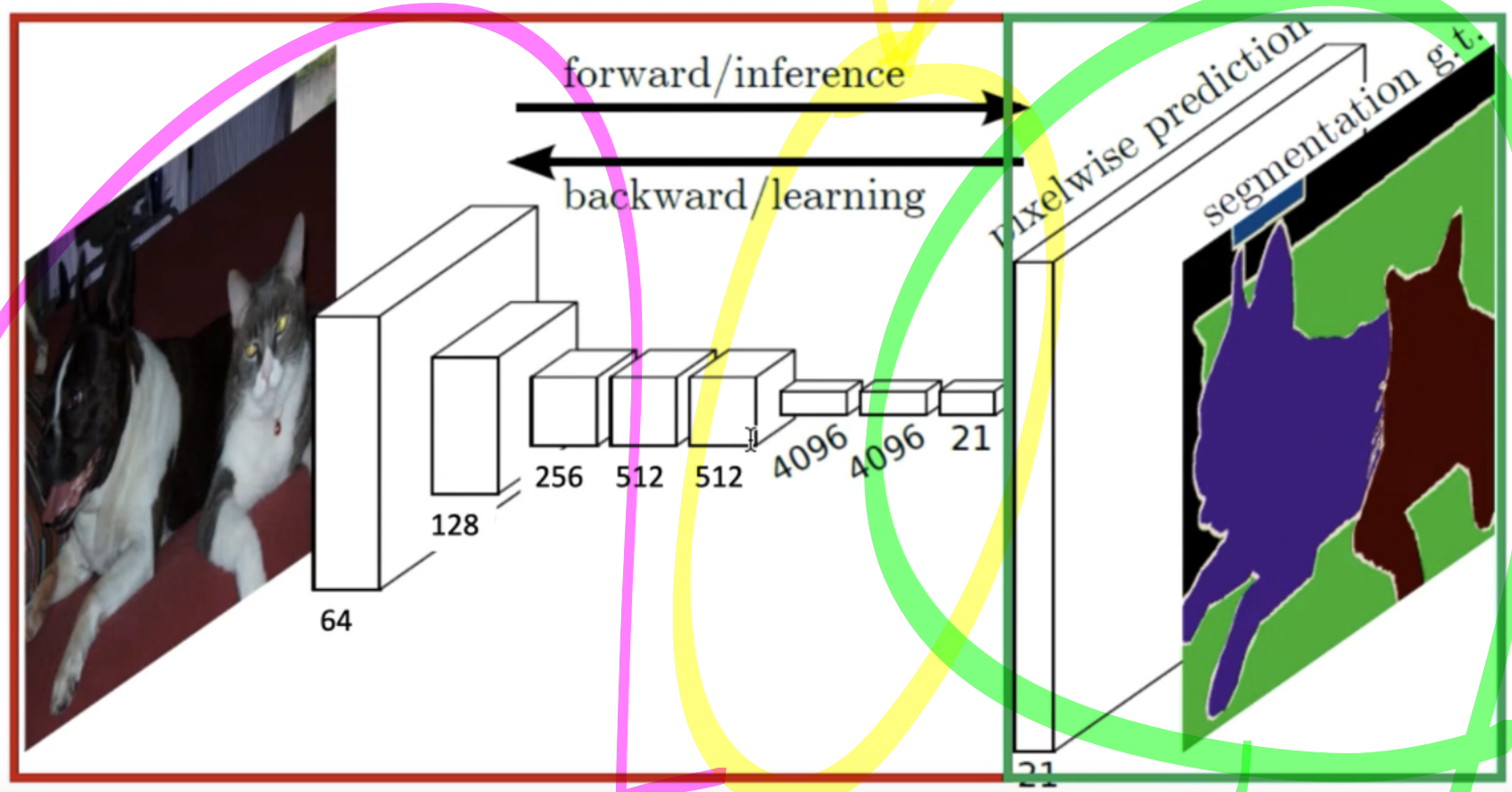
- Fully Convolutional Network




```
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)

    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)
```

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

Overall Network Structure of FCN



Skip Architecture (1/2)

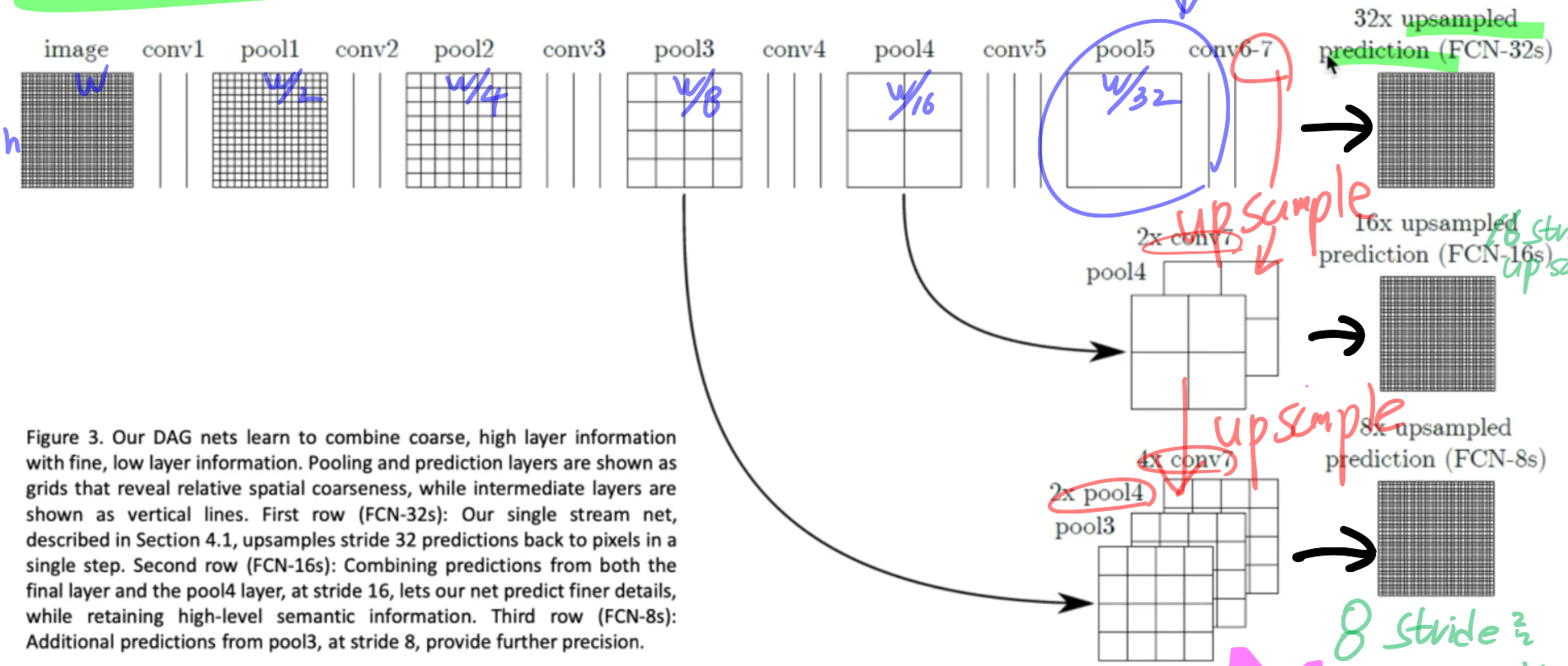
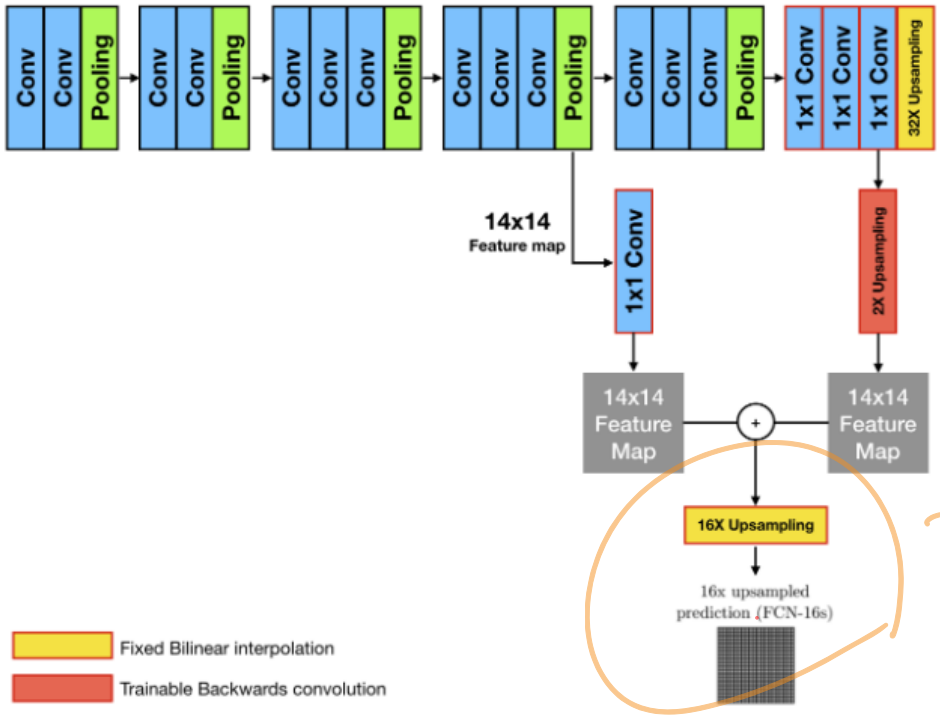
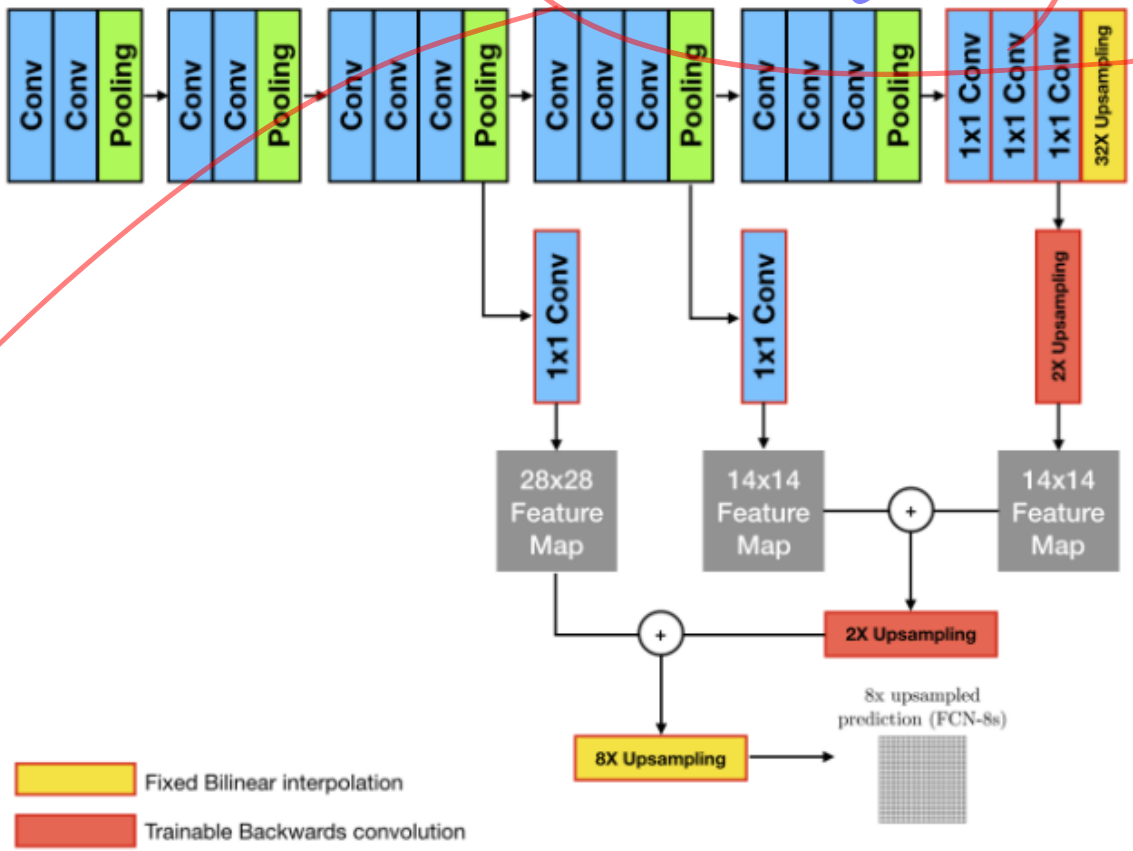


Figure 3. Our DAG nets learn to combine coarse, high layer information with fine, low layer information. Pooling and prediction layers are shown as grids that reveal relative spatial coarseness, while intermediate layers are shown as vertical lines. First row (FCN-32s): Our single stream net, described in Section 4.1, upsamples stride 32 predictions back to pixels in a single step. Second row (FCN-16s): Combining predictions from both the final layer and the pool4 layer, at stride 16, lets our net predict finer details, while retaining high-level semantic information. Third row (FCN-8s): Additional predictions from pool3, at stride 8, provide further precision.

- Initialized the 2 × upsampling to bilinear interpolation


Conv
Conv
Pooling
1x1 Conv
32X Upsampling
14x14
Feature map
2X Upsampling
14x14
Feature
Map
16X Upsampling
16x upsampled
prediction (FCN-16s)
Fixed Bilinear interpolation
Trainable Backwards convolution

FCN-16s


Conv
Conv
Pooling
1x1 Conv
32X Upsampling
28x28
Feature
Map
14x14
Feature
Map
2X Upsampling
8X Upsampling
8x upsampled
prediction (FCN-8s)
Fixed Bilinear interpolation
Trainable Backwards convolution

Skip Architecture (2/2)

- Affect of skip architecture

Figure 4. Refining fully convolutional nets by fusing information from layers with different strides improves segmentation detail. The first three images show the output from our 32, 16, and 8 pixel stride nets (see Figure 3).

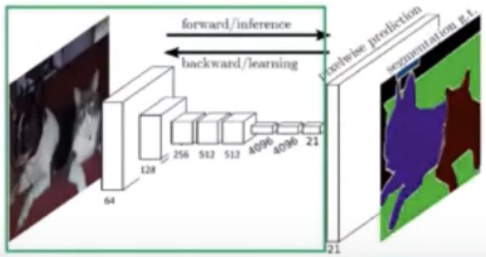
Table 2. Comparison of skip FCNs on a subset[7] of PASCAL VOC 2011 segval. Learning is end-to-end, except for FCN-32s-fixed, where only the last layer is fine-tuned. Note that FCN-32s is FCN-VGG16, renamed to highlight stride.

	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**

FCN Code (4/5)

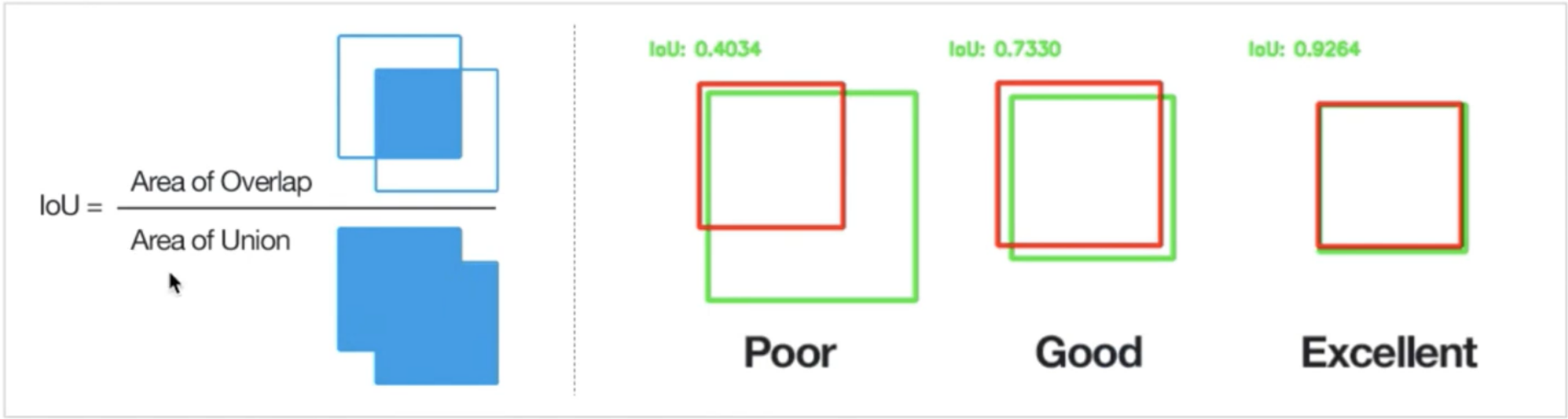
- Upsampling (deconvolution) and skip architecture

```
# now to upscale to actual image size
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")

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")

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")
```



Results 평가방식

- Metrics



- pixel accuracy: $\sum_i n_{ii} / \sum_i t_i$
- mean accuracy : $(1/n_{\text{cl}}) \sum_i n_{ii}/t_i$
- mean IU: $(1/n_{\text{cl}}) \sum_i n_{ii} / \left(t_i + \sum_j n_{ji} - n_{ii}\right)$
- frequency weighted IU:
  $(\sum_k t_k)^{-1} \sum_i t_i n_{ii} / \left(t_i + \sum_j n_{ji} - n_{ii}\right)$

Results

- PASCAL VOC

Table 3. Our fully convolutional net gives a 20% relative improvement over the state-of-the-art on the PASCAL VOC 2011 and 2012 test sets and reduces inference time.

	mean IU VOC2011 test	mean IU VOC2012 test	inference time
R-CNN [10]	47.9	-	-
SDS [15]	52.6	51.6	~ 50 s
FCN-8s	**62.7**	**62.2**	~ **175 ms**

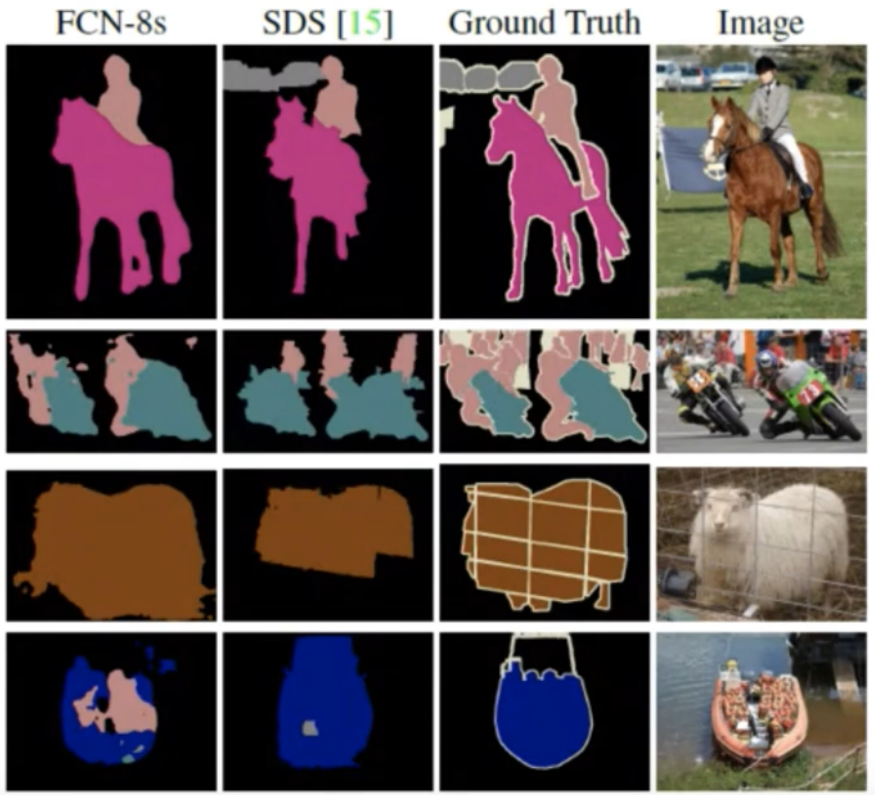


Figure 6. Fully convolutional segmentation nets produce state-of-the-art performance on PASCAL. The left column shows the output of our highest performing net, FCN-8s. The second shows the segmentations produced by the previous state-of-the-art system by Hariharan et al. [15]. Notice the fine structures recovered (first row), ability to separate closely interacting objects (second row), and robustness to occluders (third row). The fourth row shows a failure case: the net sees lifejackets in a boat as people.

Results

- NYUDv2

Table 4. Results on NYUDv2. *RGBD* is early-fusion of the RGB and depth channels at the input. *HHA* is the depth embedding of [13] as horizontal disparity, height above ground, and the angle of the local surface normal with the inferred gravity direction. *RGB-HHA* is the jointly trained late fusion model that sums RGB and HHA predictions.

	pixel acc.	mean acc.	mean IU	f.w. IU
Gupta *et al.* [13]	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**

- SIFT Flow

Table 5. Results on SIFT Flow[9] with class segmentation (center) and geometric segmentation (right). Tighe [33] is a non-parametric transfer method. Tighe 1 is an exemplar SVM while 2 is SVM + MRF. Farabet is a multi-scale convnet trained on class-balanced samples (1) or natural frequency samples (2). Pinheiro is a multi-scale, recurrent convnet, denoted RCNN$_3$ (o^3). The metric for geometry is pixel accuracy.

	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.* [7] 1	72.3	50.8	-	-	-
Farabet *et al.* [7] 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**