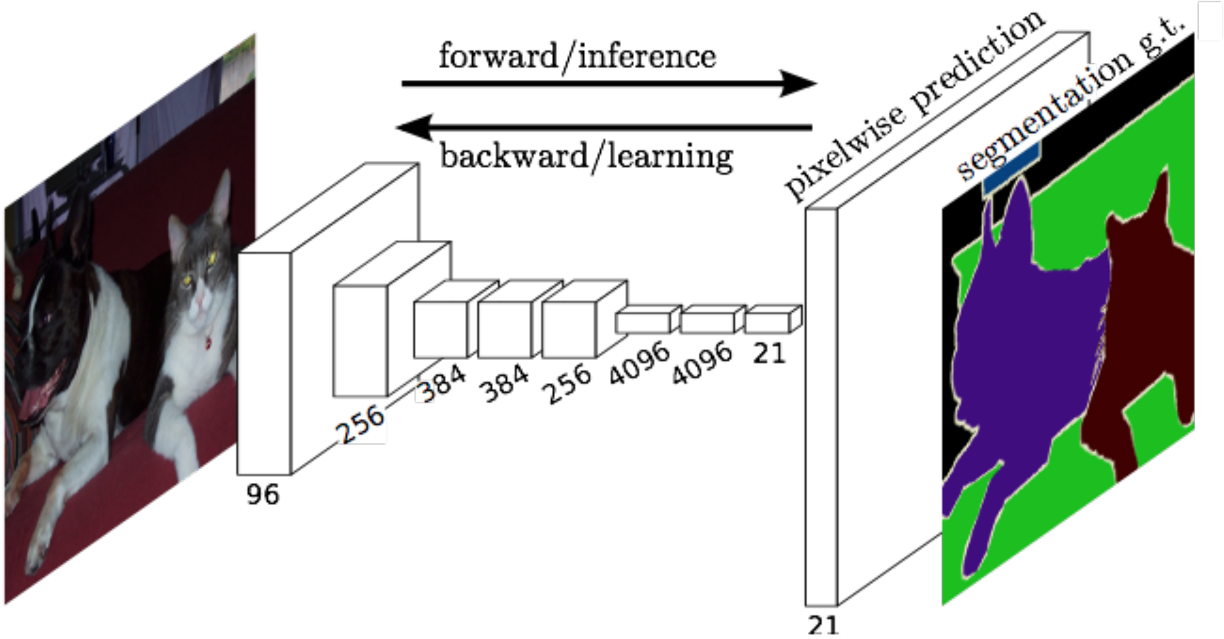
Fully Convolutional Networks

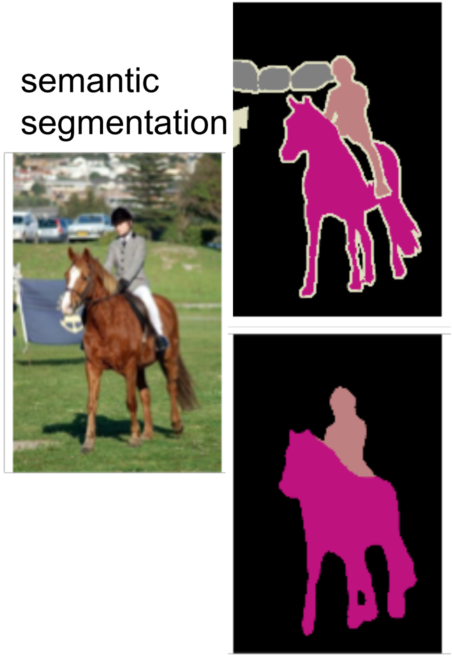


Jon Long and Evan Shelhamer

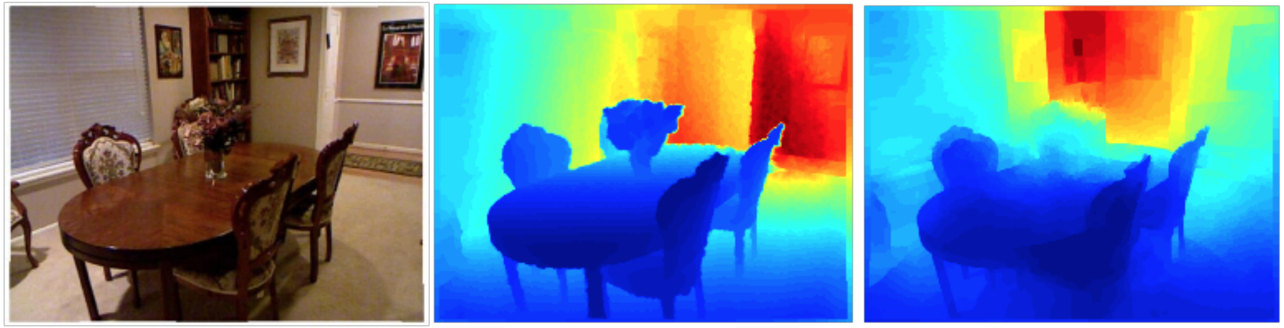
CVPR15 Caffe Tutorial

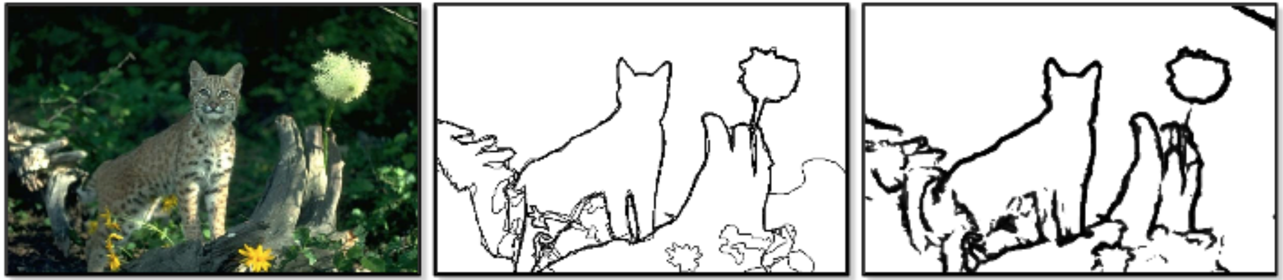
pixels in, pixels out

semantic
segmentation

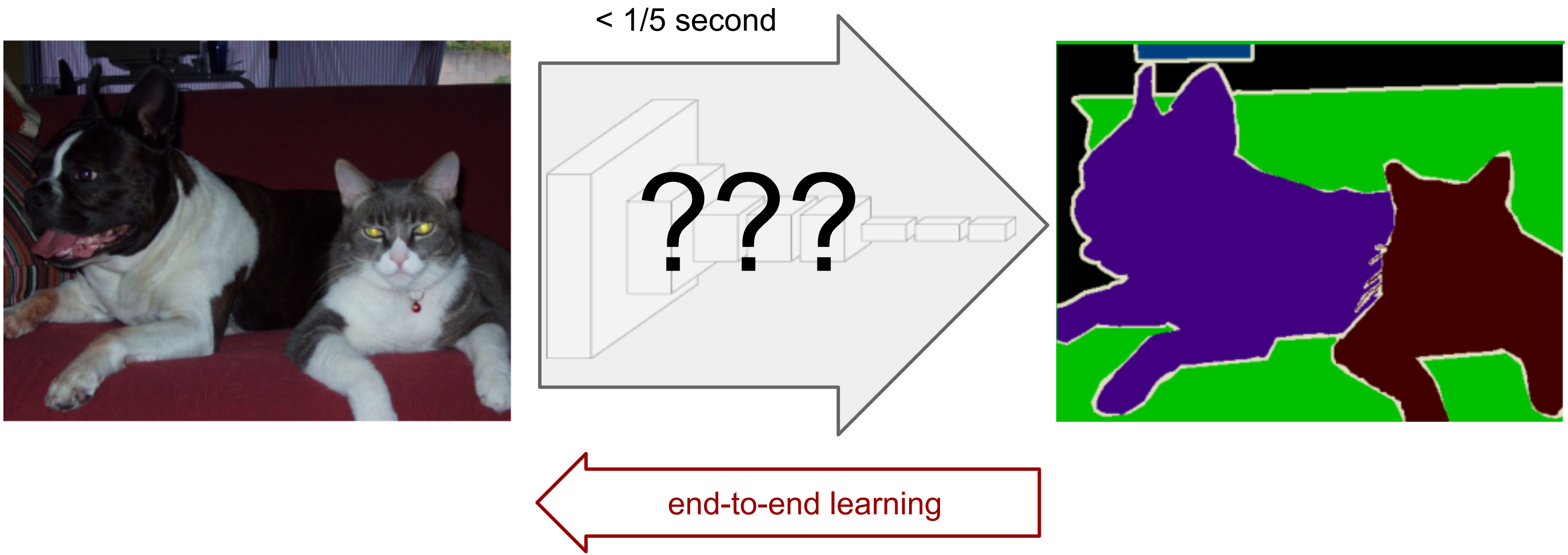


monocular depth estimation (Liu et al. 2015)

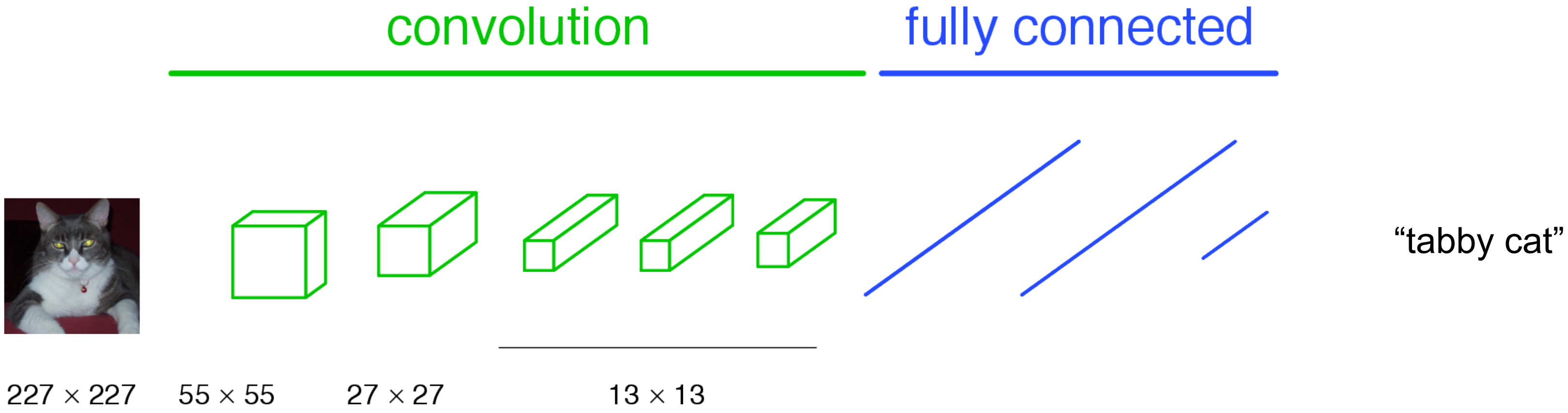




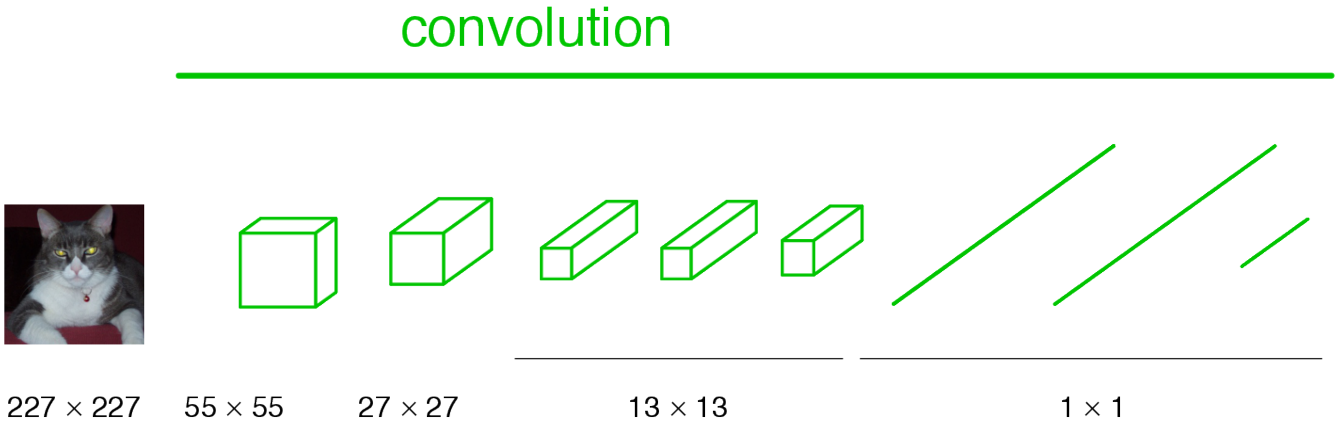
boundary prediction (Xie & Tu 2015)


< 1/5 second
???
end-to-end learning

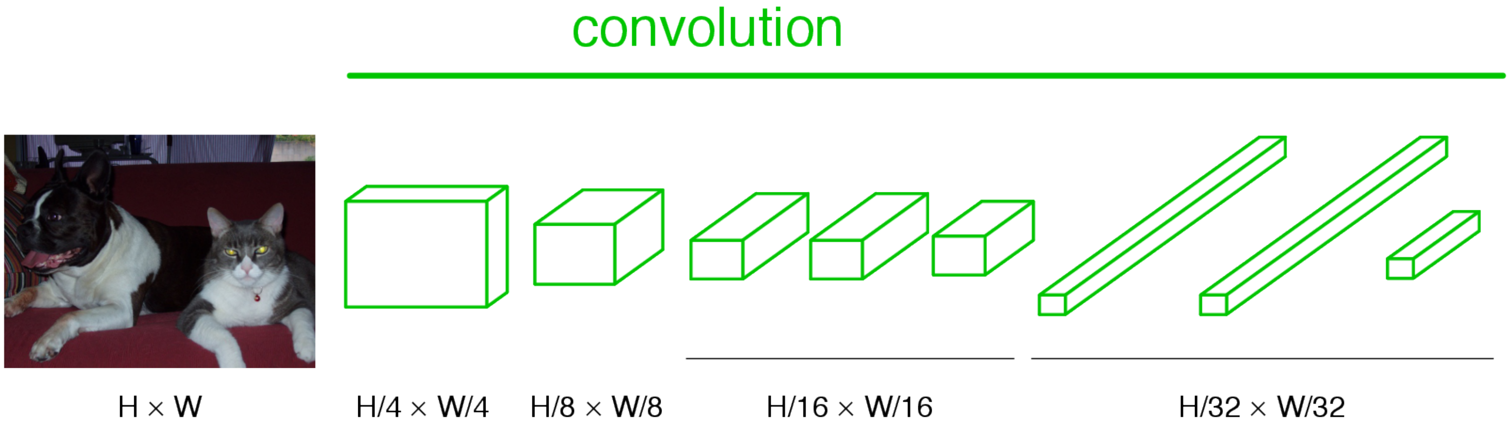
a classification network



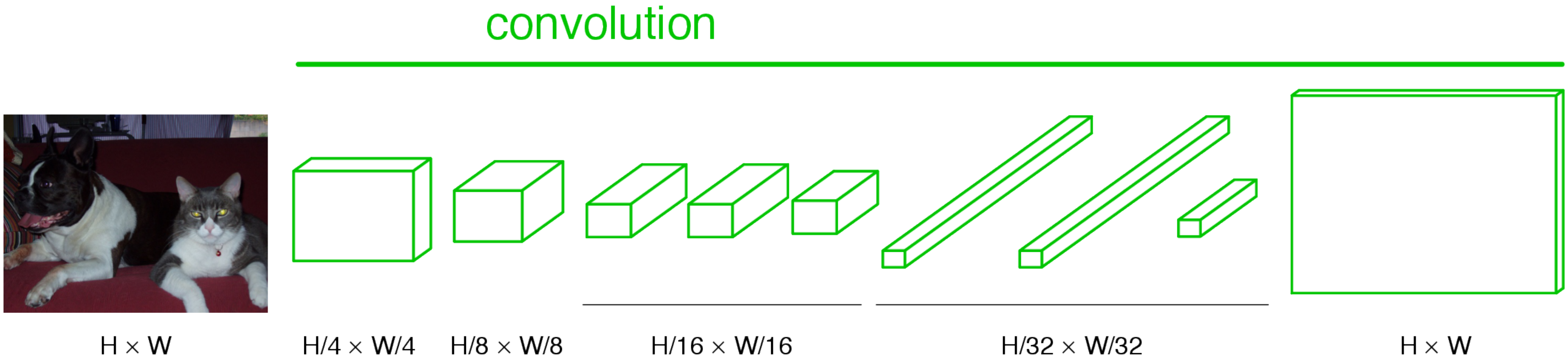
becoming fully convolutional



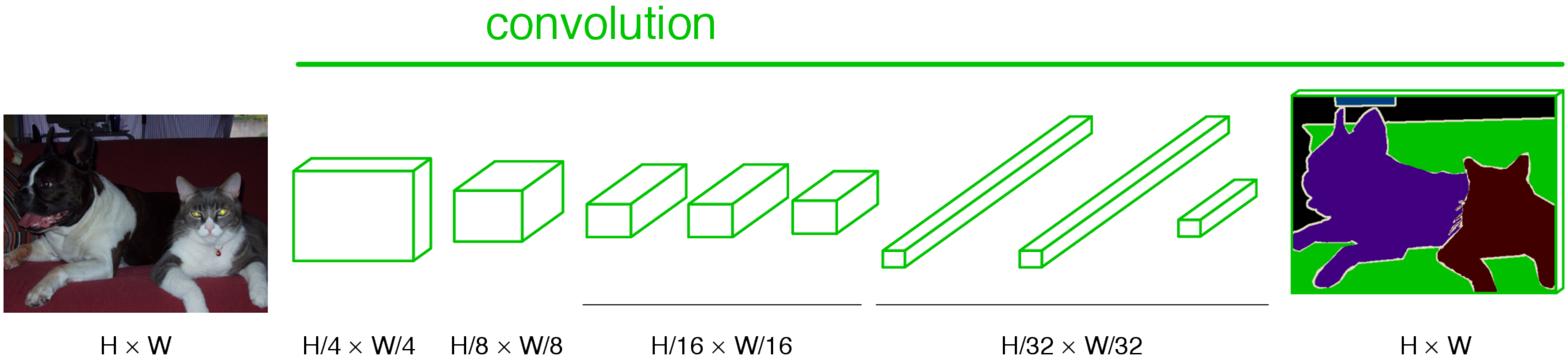
becoming fully convolutional



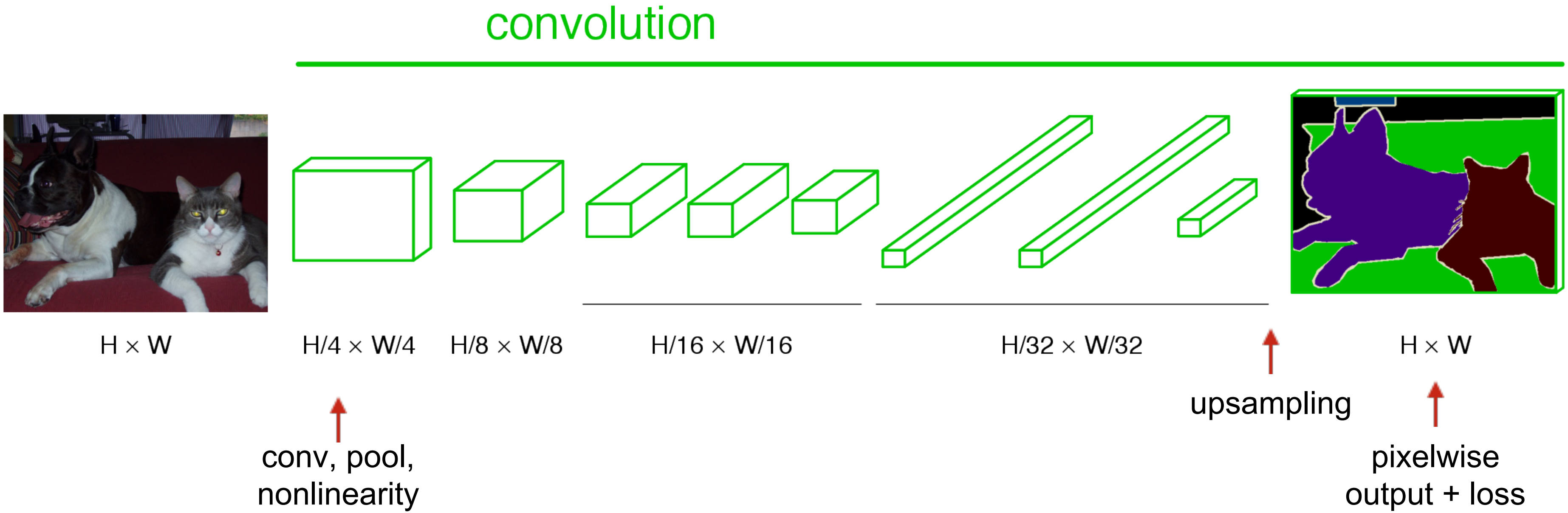
upsampling output



end-to-end, pixels-to-pixels network

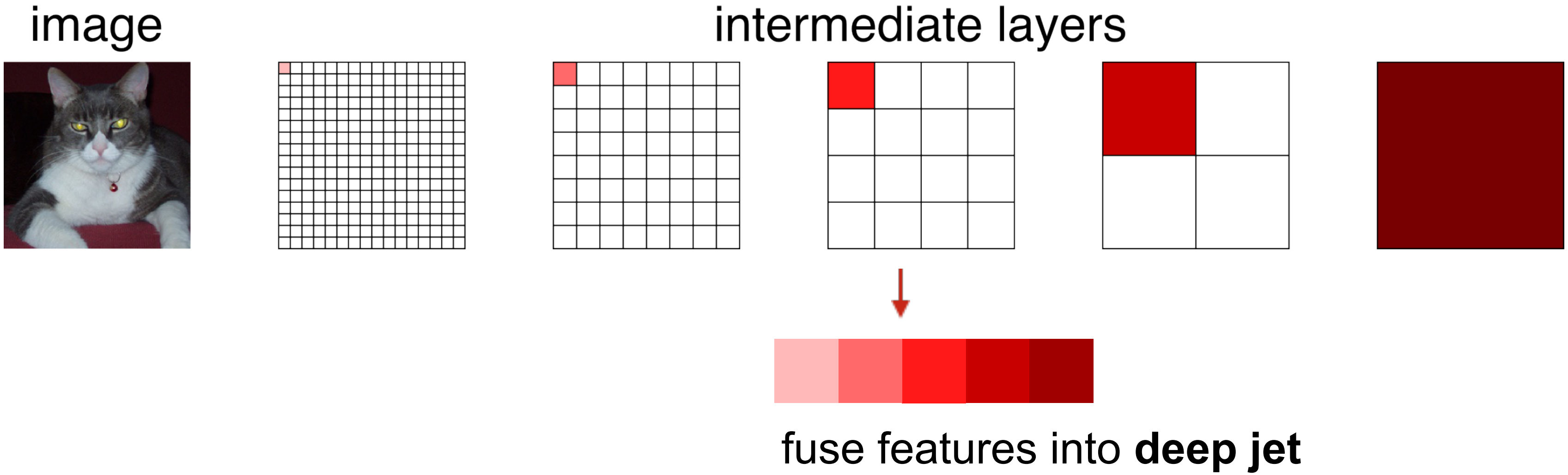


end-to-end, pixels-to-pixels network



spectrum of deep features

combine *where* (local, shallow) with *what* (global, deep)



(cf. Hariharan et al. CVPR15 “hypercolumn”)

skip layers

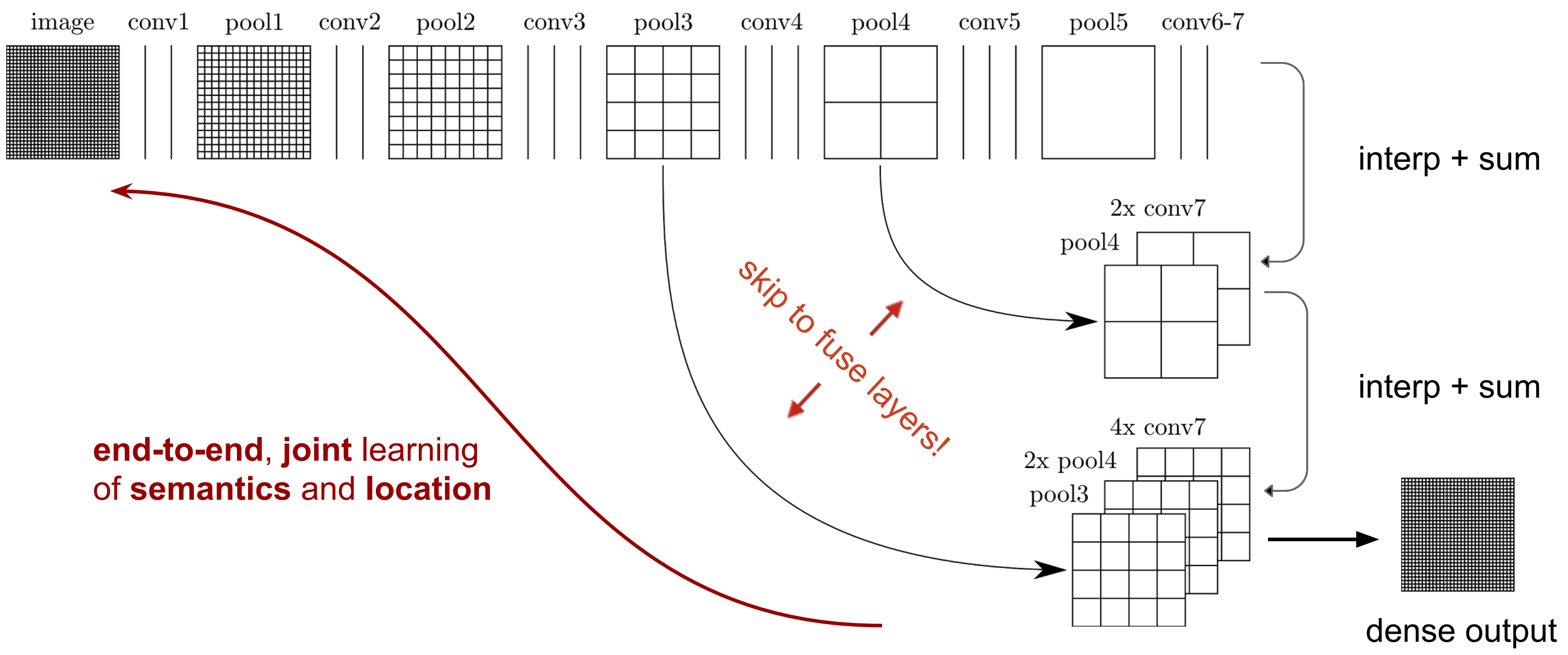
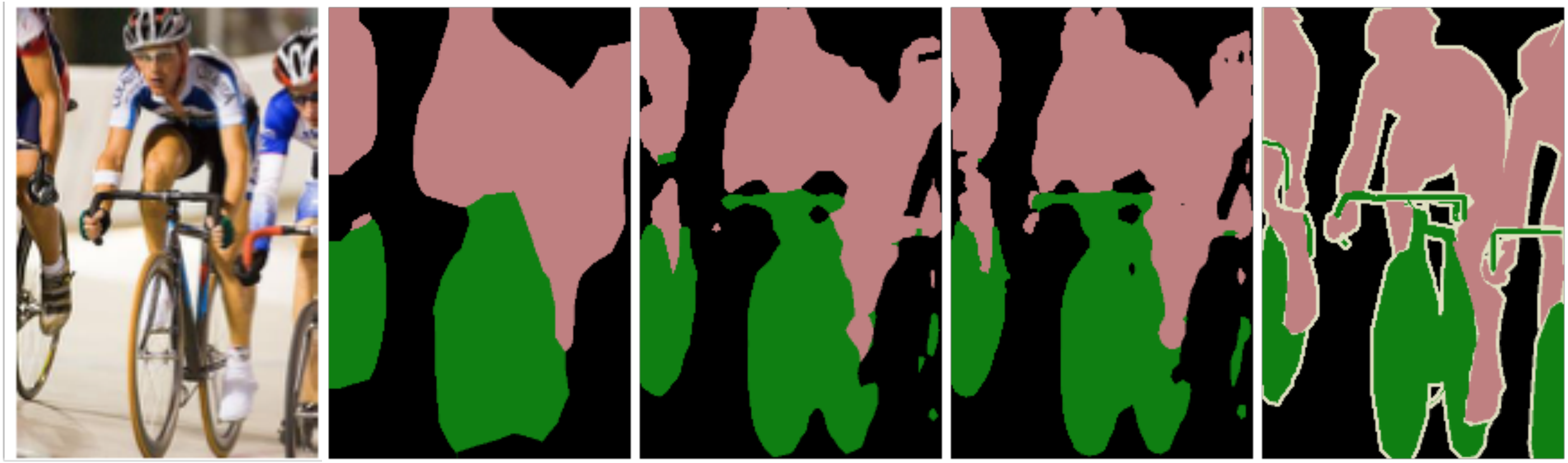


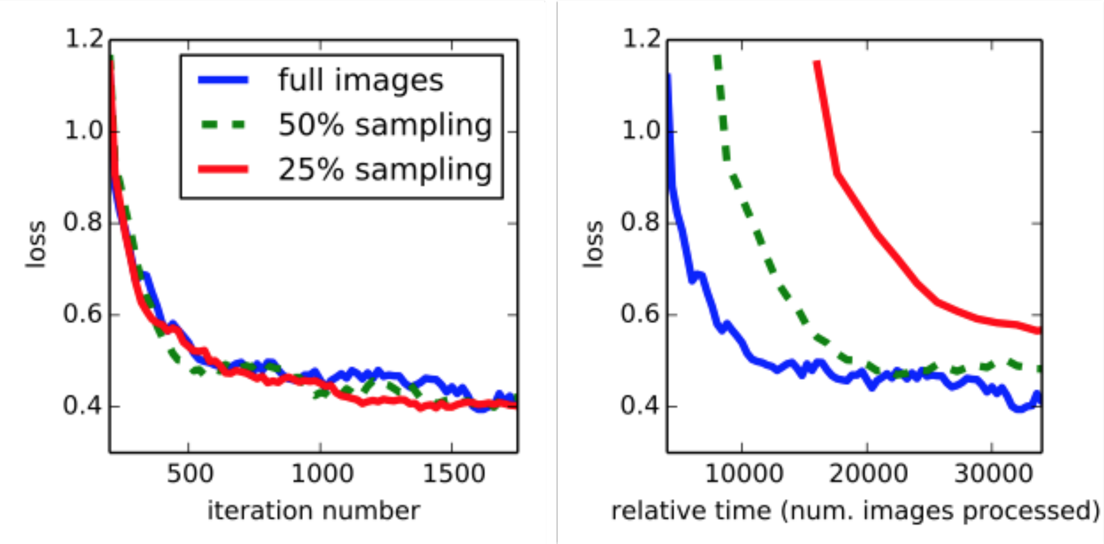
image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6-7
interp + sum
2x conv7
pool4
skip to fuse layers!
interp + sum
4x conv7
2x pool4
pool3
end-to-end, joint learning
of semantics and location
dense output

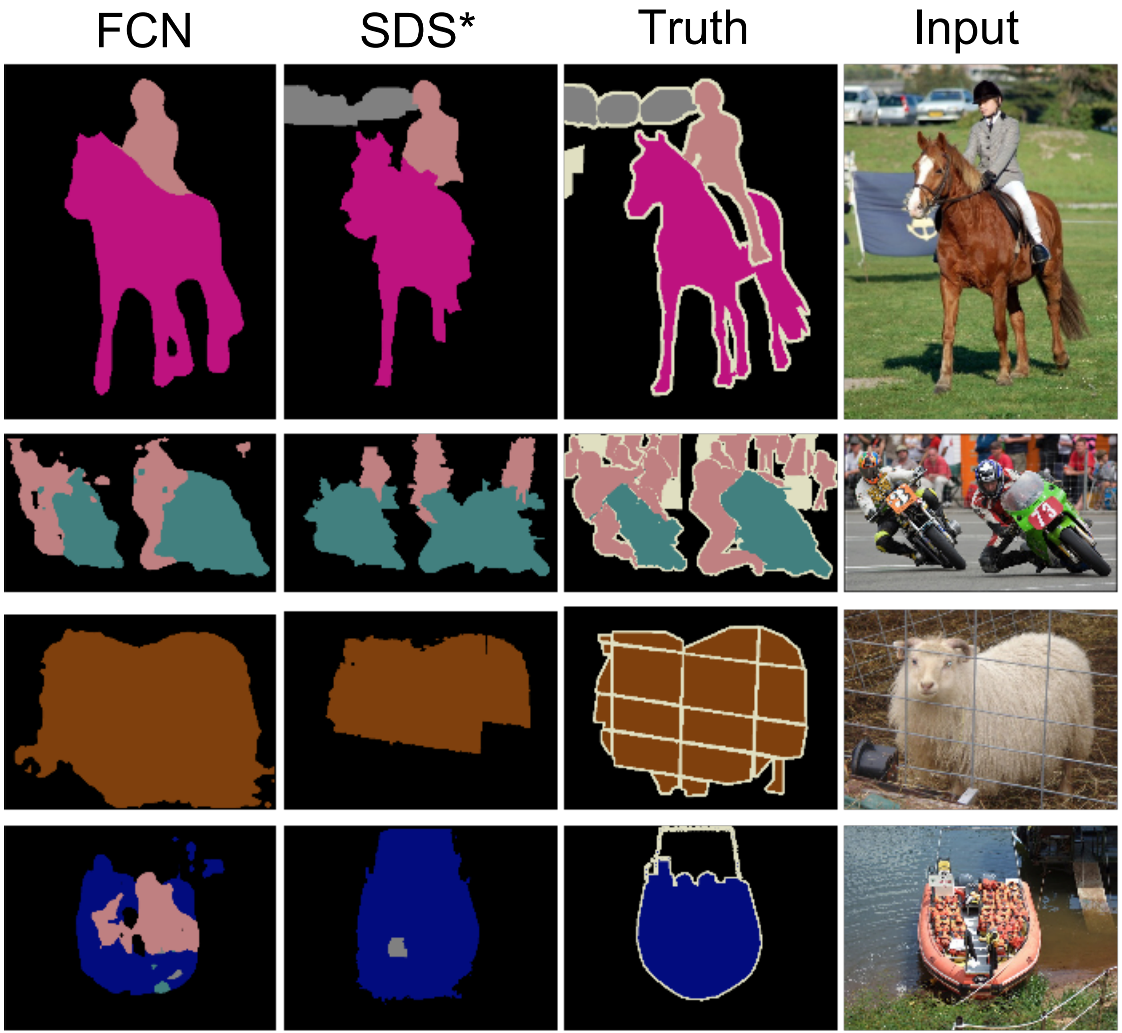
skip layer refinement



training + testing

- train full image at a time *without patch sampling*
- reshape network to take input of any size
- forward time is ~150ms for 500 x 500 x 21 output





Relative to prior state-of-the-art SDS:

- 20% improvement for mean IoU
- 286× faster

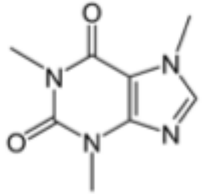
*Simultaneous Detection and Segmentation
Hariharan et al. ECCV14

models + code

fully convolutional networks are fast, end-to-end models for pixelwise problems

- **code** in Caffe branch (merged soon)
- **models** for PASCAL VOC, NYUDv2, SIFT Flow, PASCAL-Context in Model Zoo

fcn.berkeleyvision.org



caffe.berkeleyvision.org

github.com/BVLC/caffe

models

- **PASCAL VOC** standard for object segmentation
- **NYUDv2** multi-modal rgb + depth scene segmentation
- **SIFT Flow** multi-task for semantic + geometric segmentation
- **PASCAL-Context** object + scene segmentation

inference

eval.py

Raw

```
import numpy as np
from PIL import Image

import caffe

# load image, switch to BGR, subtract mean, and make dims C x H x W for Caffe
im = Image.open('pascal/VOC2010/JPEGImages/2007_000129.jpg')
in_ = np.array(im, dtype=np.float32)
in_ = in_[:,:,::-1]
in_ -= np.array((104.00698793,116.66876762,122.67891434))
in_ = in_.transpose((2,0,1))

# load net
net = caffe.Net('deploy.prototxt', 'fcn-32s-pascalcontext.caffemodel', caffe.TEST)
# shape for input (data blob is N x C x H x W), set data
net.blobs['data'].reshape(1, *in_.shape)
net.blobs['data'].data[...] = in_
# run net and take argmax for prediction
net.forward()
out = net.blobs['score'].data[0].argmax(axis=0)
```

[inference script (gist)]

solving

```
# base net -- follow the editing model parameters example to make
# a fully convolutional VGG16 net.
# http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/net_surgery.ipynb
base_weights = 'vgg16fc.caffemodel'

# init
caffe.set_mode_gpu()
caffe.set_device(0)

solver = caffe.SGDSolver('solver.prototxt')

# do net surgery to set the deconvolution weights for bilinear interpolation
interp_layers = [k for k in solver.net.params.keys() if 'up' in k]
interp_surgery(solver.net, interp_layers)

# copy base weights for fine-tuning
solver.net.copy_from(base_weights)

# solve straight through -- a better approach is to define a solving loop to
# 1. take SGD steps
# 2. score the model by the test net `solver.test_nets[0]`
# 3. repeat until satisfied
solver.step(80000)
```

[solving script (gist)]()

Reshape

- Decide shape on-the-fly in C++ / Python / MATLAB
- DataLayer automatically reshapes
  for batch size == 1
- Essentially free
  (only reallocates when necessary)

Helpful Layers

- Losses can take spatial predictions + truths
- Deconvolution / “backward convolution” can compute interpolation
- Crop: maps coordinates between layers

FCN for Pose Estimation

Georgia Gkioxari
UC Berkeley

FCN for Pose Estimation

Input data:

Image

FCN for Pose Estimation

Input data:

Image

Keypoints

FCN for Pose Estimation

Input data:

Image

Keypoints

Define an area around the keypoint as its positive neighborhood with radius r.

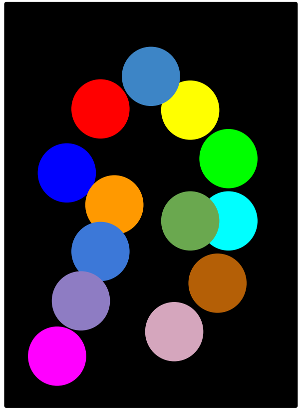
FCN for Pose Estimation

Input data:

Image

Keypoints

Labels

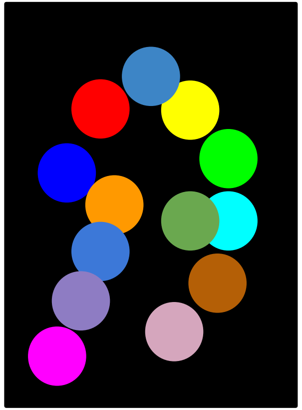


FCN for Pose Estimation

Input data:

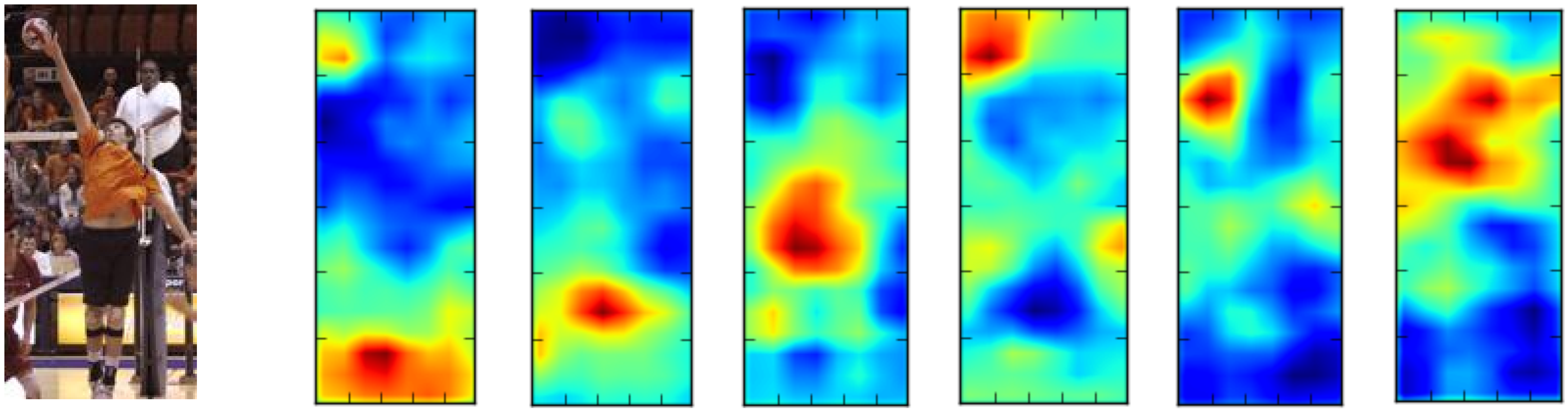
Image

Labels

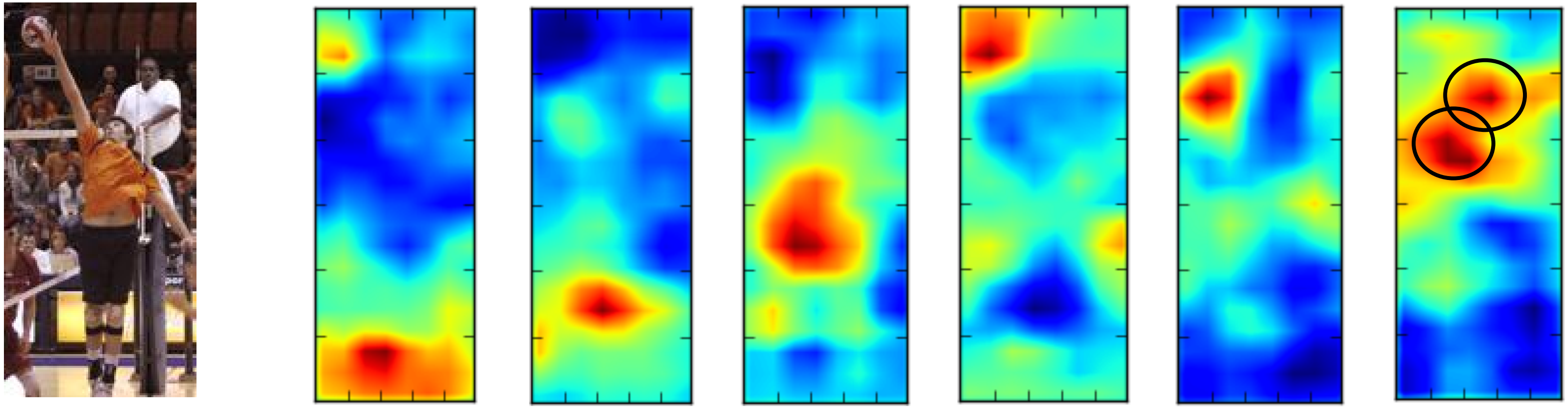


Heat Map Predictions from FCN

Test Image | Right Ankle | Right Knee | Right Hip | Right Wrist | Right Elbow | Right Shoulder



Heat Map Predictions from FCN



Two modes because there are two Right Shoulders in the image!

Heat Maps to Keypoints

PCK @ 0.2	LSP test set
Ankle	56.5
Knee	60.0
Hip	56.6
Wrist	62.9
Elbow	71.8
Shoulder	78.8
Head	93.6

FCN baseline PCK == ~69%

State-of-the-art == ~72%

Details

Architecture:

- FCN - 32 stride. **No** data augmentation.
- radius = 0.1*im.shape[0] (**no** cross validation)

Runtime on a K40:

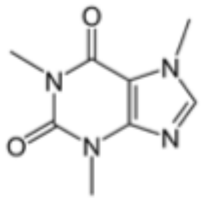
- 0.7 sec/iteration for training (15hrs for 80K iterations)
- 0.25 sec/image for inference for all keypoints

conclusion

fully convolutional networks are fast, end-to-end models for pixelwise problems

- **code** in Caffe branch (merged soon)
- **models** for PASCAL VOC, NYUDv2, SIFT Flow, PASCAL-Context in Model Zoo

fcn.berkeleyvision.org



caffe.berkeleyvision.org

github.com/BVLC/caffe