Lecture 13:

Segmentation and Attention

Administrative

- Assignment 3 due tonight!
- We are reading your milestones

Last time: Software Packages

Caffe

No need to write code!

1. Convert data (run a script)
2. Define net (edit prototxt)
3. Define solver (edit prototxt)
4. Train (with pretrained weights)

Torch

```lua
require 'torch'
require 'nn'
require 'optim'

-- Batch size, input dim, hidden dim, num classes
local N, D, H, C = 100, 1000, 100, 10

-- Build a one-layer ReLU network
local net = nn.Sequential()
net:add(nn.Linear(D, H))
net:add(nn.ReLU())
net:add(nn.Linear(H, C))

-- Collect all weights and gradients in a single Tensor
local weights, grad_weights = net:getParameters()

-- Loss functions are called "criterions"
local crit = nn.CrossEntropyCriterion()  -- Softmax loss

-- Calback to interface with optim methods
local function f(w)
  assert(w == weights)

  -- Generate some random input data
  local x = torch.randn(N, D)
  local y = torch.Tensor(N):random(C)

  -- Forward pass: Compute scores and loss
  local scores = net:forward(x)
  local loss = crit:forward(scores, y)

  -- Backward pass: compute gradients
  grad_weights:zero()
  local dscores = crit:backward(scores, y)
  local dx = net:backward(x, dscores)

  return loss, grad_weights
end

-- Make a step using Adam
local state = {learningRate=1e-3}
optim.adam(f, weights, state)
```

Theano

```python
import theano
import theano.tensor as T

# Batch size, input dim, hidden dim, num classes
N, D, H, C = 64, 1000, 100, 10

x = T.matrix('x')
y = T.vector('y', dtype='int64')
w1 = T.matrix('w1')
w2 = T.matrix('w2')

# Forward pass: Compute scores
a = x.dot(w1)
a_relu = T.nnet.relu(a)
scores = a_relu.dot(w2)

# Forward pass: compute softmax loss
probs = T.nnet.softmax(scores)
loss = T.nnet.categorical_crossentropy(probs, y).mean()

# Backward pass: compute gradients
dw1, dw2 = T.grad(loss, [w1, w2])

f = theano.function(
      inputs=[x, y, w1, w2],
      outputs=[loss, scores, dw1, dw2],
    )
```

Lasagne

```python
import numpy as np
import theano
import theano.tensor as T
import lasagne

N, D, H, C = 64, 1000, 100, 10

x = T.matrix('x')
y = T.vector('y', dtype='int64')

relu = lasagne.nonlinearities.rectify
softmax = lasagne.nonlinearities.softmax
net = lasagne.layers.InputLayer(shape=(None, D), input_var=x)
net = lasagne.layers.DenseLayer(net, H, nonlinearity=relu)
net = lasagne.layers.DenseLayer(net, C, nonlinearity=softmax)

probs = lasagne.layers.get_output(net)
loss = lasagne.objectives.categorical_crossentropy(probs, y).mean()

params = lasagne.layers.get_all_params(net, trainable=True)
updates = lasagne.updates.nesterov_momentum(loss, params,
                    learning_rate=1e-2, momentum=0.0)

train_fn = theano.function([x, y], loss, updates=updates)

xx = np.random.randn(N, D)
yy = np.random.randint(C, size=N).astype(np.int64)

for t in xrange(100):
  loss_val = train_fn(xx, yy)
  print loss_val
```

TensorFlow

```python
import tensorflow as tf
import numpy as np

N, D, H, C = 64, 1000, 100, 10

x = tf.placeholder(tf.float32, shape=[None, D])
y = tf.placeholder(tf.float32, shape=[None, C])

w1 = tf.Variable(1e-3 * np.random.randn(D, H).astype(np.float32))
w2 = tf.Variable(1e-3 * np.random.randn(H, C).astype(np.float32))

a = tf.matmul(x, w1)
a_relu = tf.nn.relu(a)
scores = tf.matmul(a_relu, w2)
probs = tf.nn.softmax(scores)
loss = -tf.reduce_sum(y * tf.log(probs))

learning_rate = 1e-2
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

xx = np.random.randn(N, D).astype(np.float32)
yy = np.zeros((N, C)).astype(np.float32)
yy[np.arange(N), np.random.randint(C, size=N)] = 1

with tf.Session() as sess:
  sess.run(tf.initialize_all_variables())

  for t in xrange(100):
    _, loss_value = sess.run([train_step, loss],
                             feed_dict={x: xx, y: yy})
    print loss_value
```

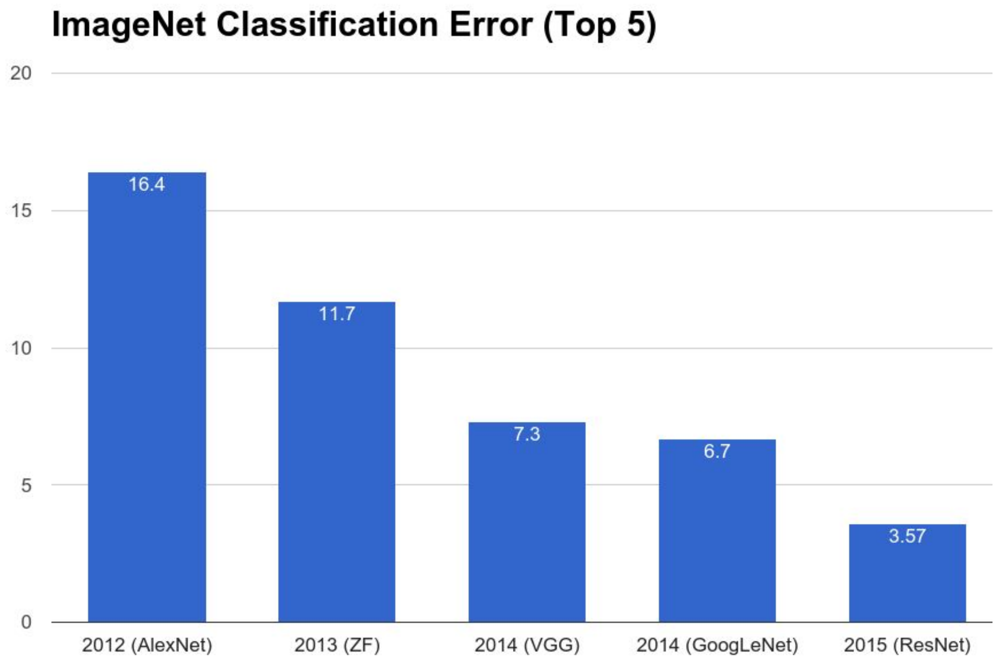
Keras

```python
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD
from keras.utils import np_utils

D, H, C = 1000, 100, 10

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=C))
model.add(Activation('softmax'))

sgd = SGD(lr=1e-3, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd)

N, N_batch = 1000, 32
X = np.random.randn(N, D)
y = np.random.randint(C, size=N)
y = np_utils.to_categorical(y)

model.fit(X, y, nb_epoch=5, batch_size=N_batch, verbose=2)
```

Today

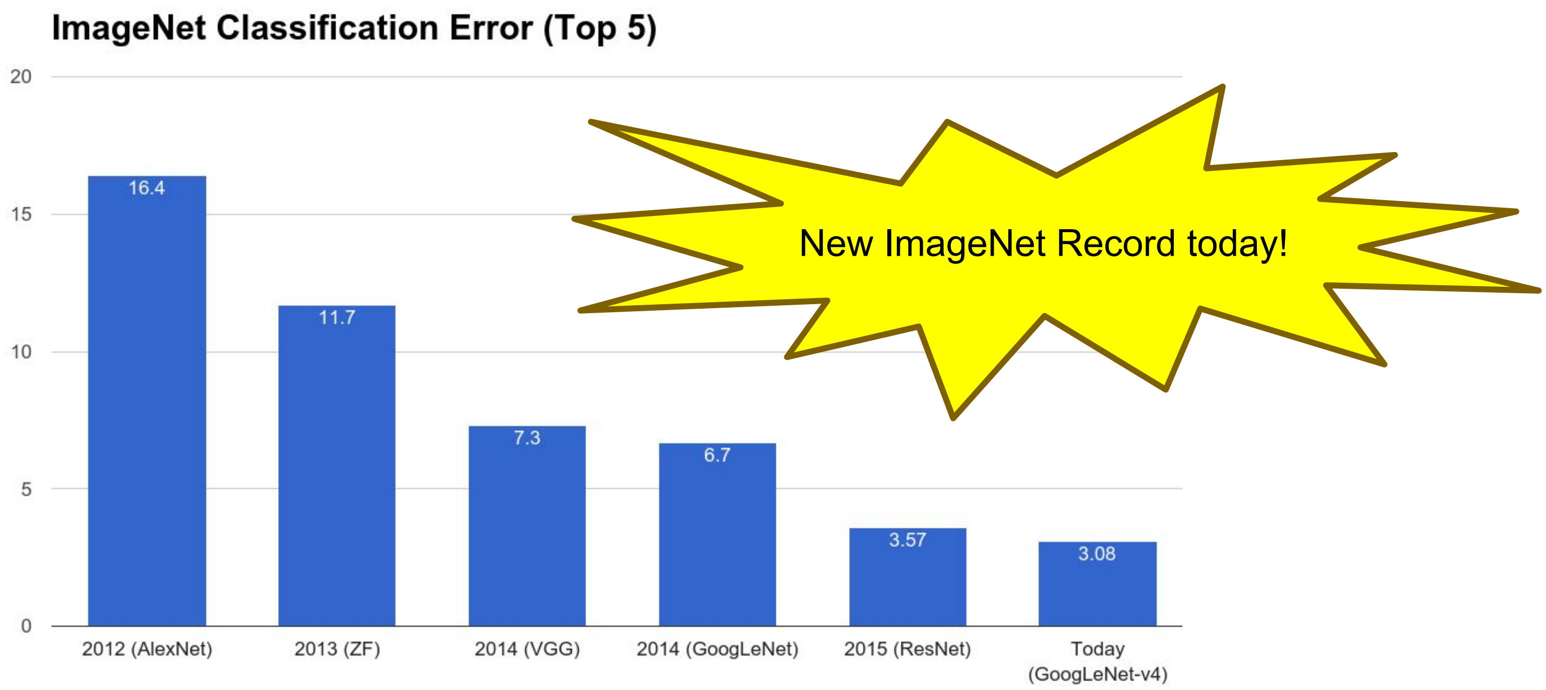
- Segmentation
    - Semantic Segmentation
    - Instance Segmentation
- (Soft) Attention
    - Discrete locations
    - Continuous locations (Spatial Transformers)

But first….



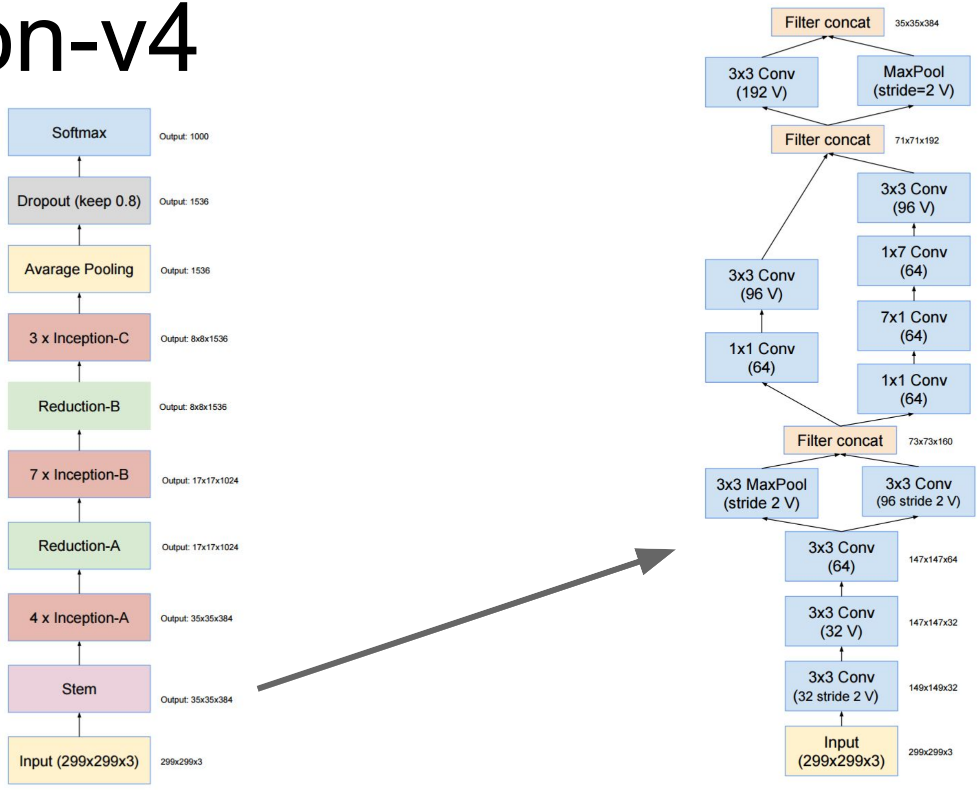
ImageNet Classification Error (Top 5)
20
15
10
5
0
16.4
11.7
7.3
6.7
3.57
2012 (AlexNet)
2013 (ZF)
2014 (VGG)
2014 (GoogLeNet)
2015 (ResNet)

But first….



Szegedy et al, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, arXiv 2016

Inception-v4



9 layers

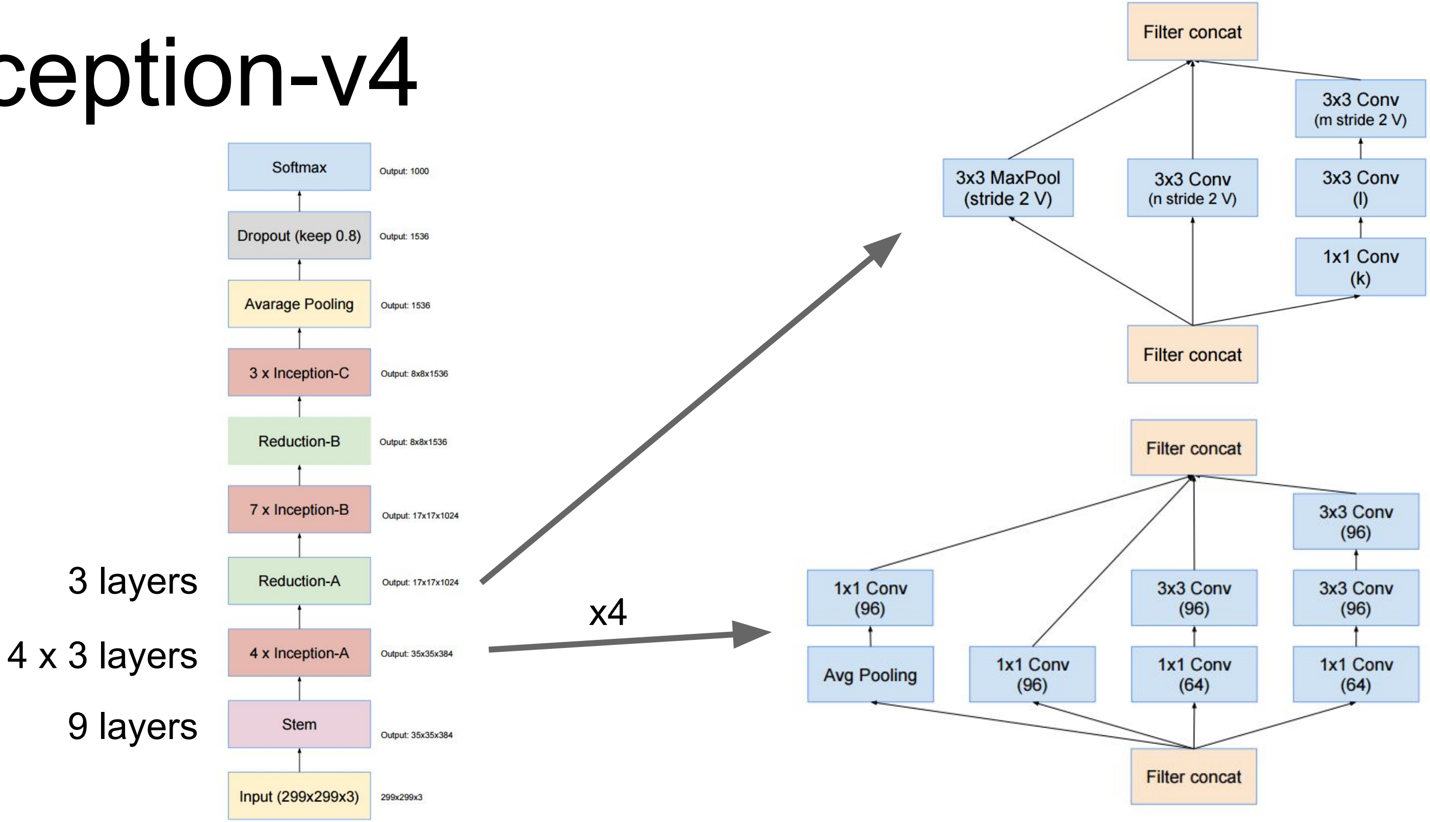
1x7, 7x1 filters

Strided convolution AND max pooling

V = Valid convolutions (no padding)

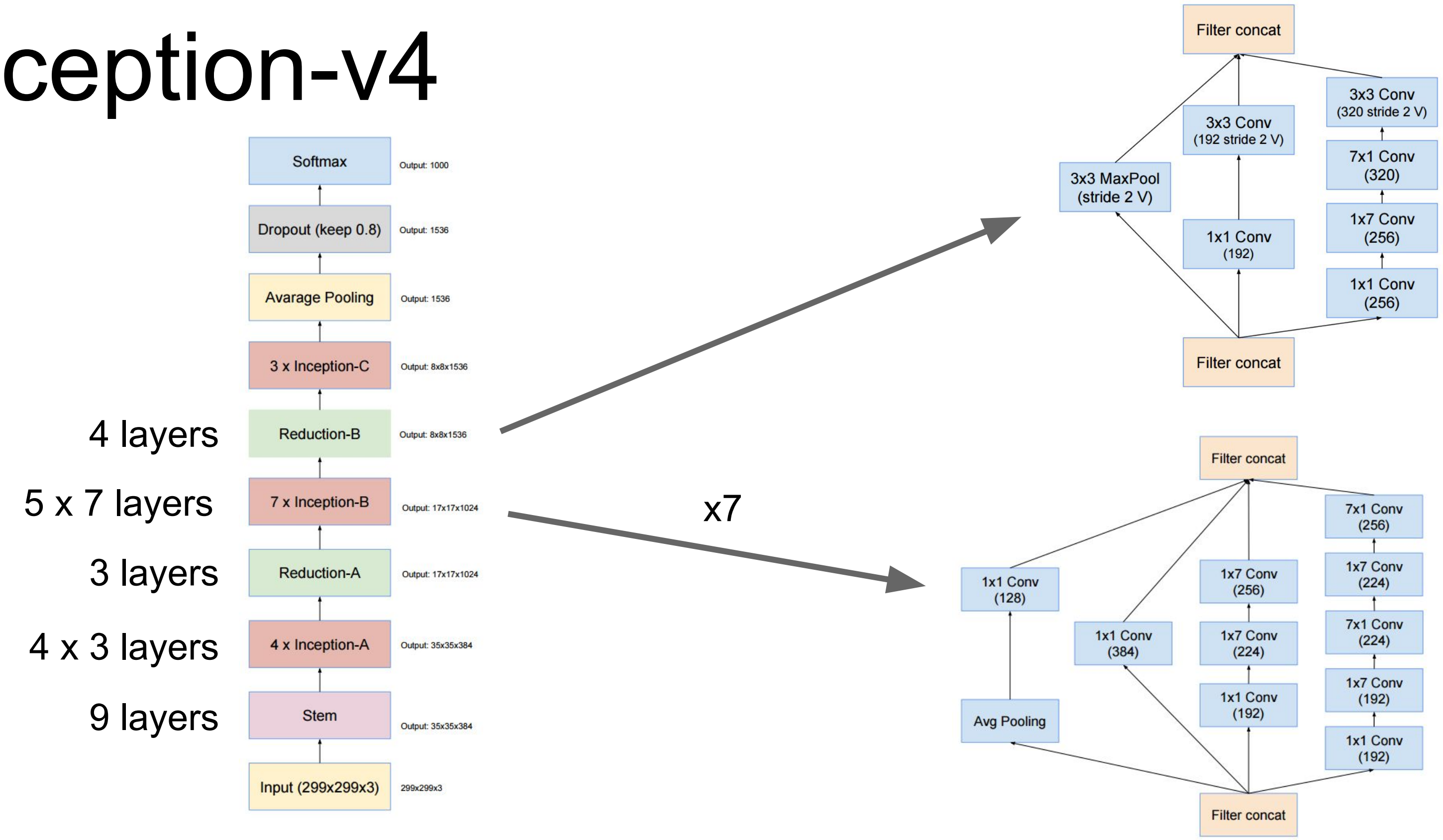
Szegedy et al, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, arXiv 2016

Inception-v4



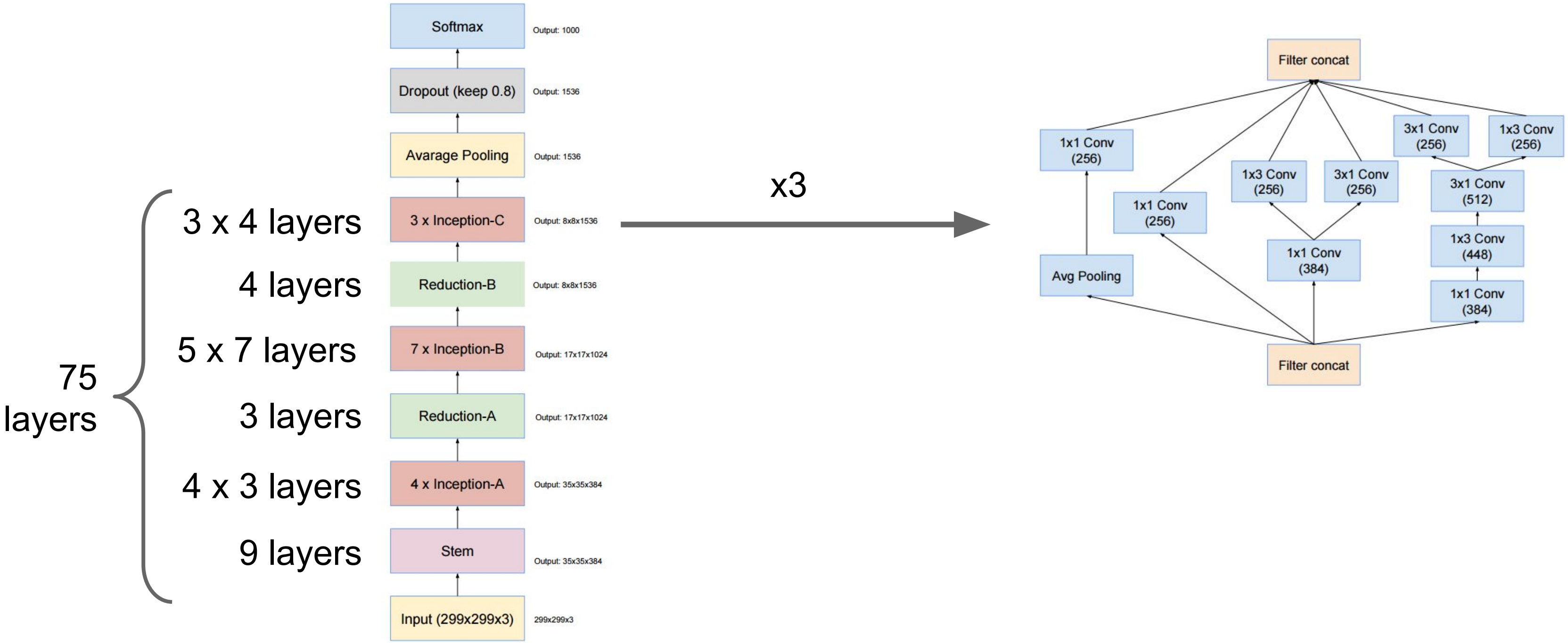
Szegedy et al, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, arXiv 2016

Inception-v4



Szegedy et al, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, arXiv 2016

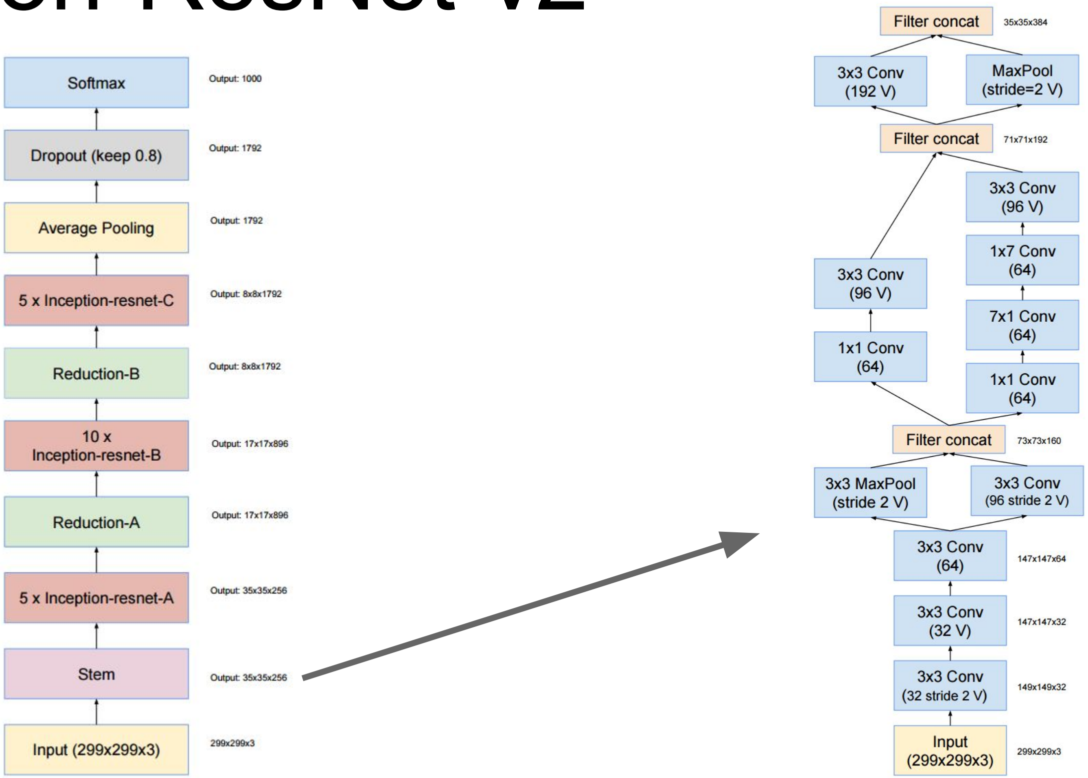
Inception-v4



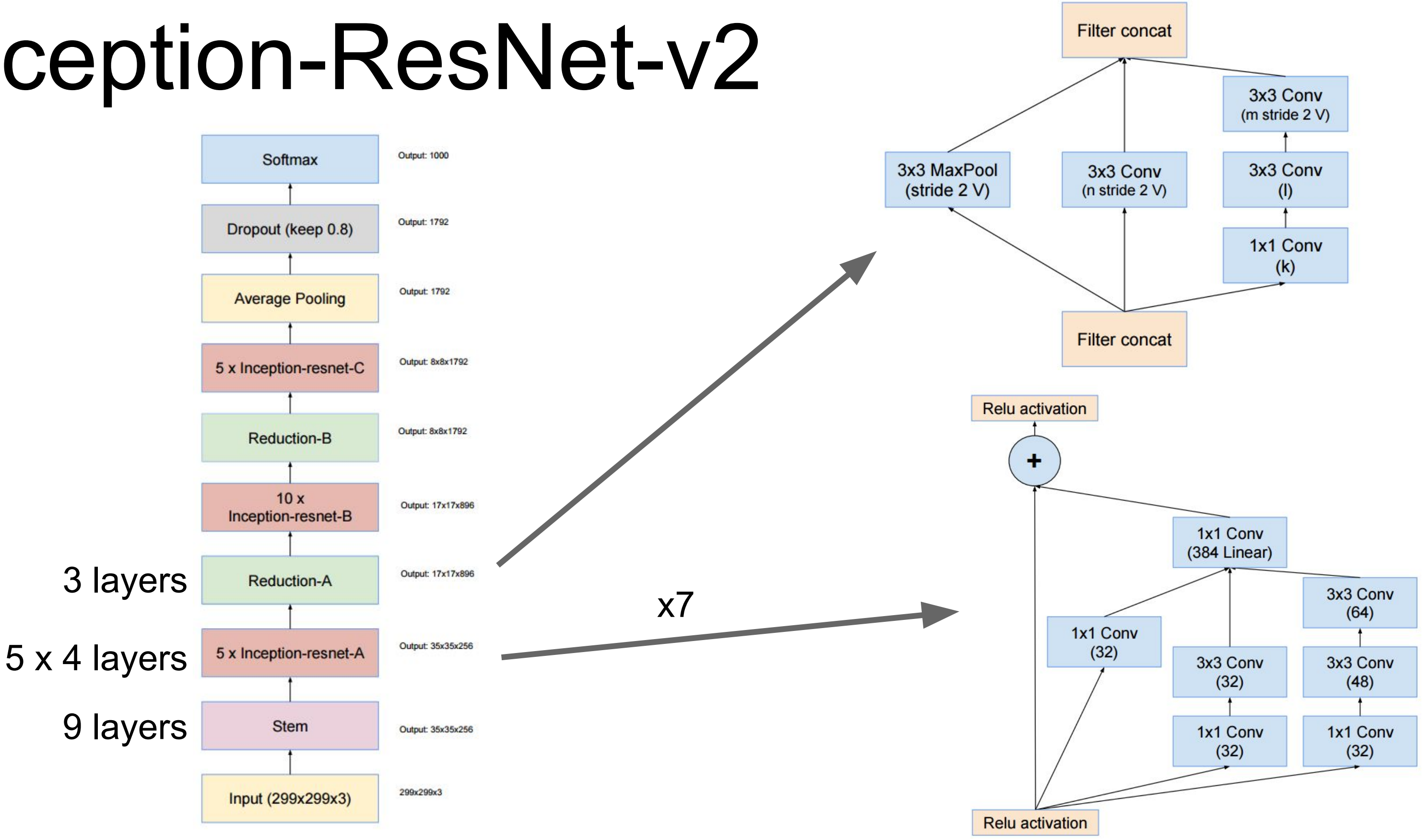
Szegedy et al, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, arXiv 2016

Inception-ResNet-v2

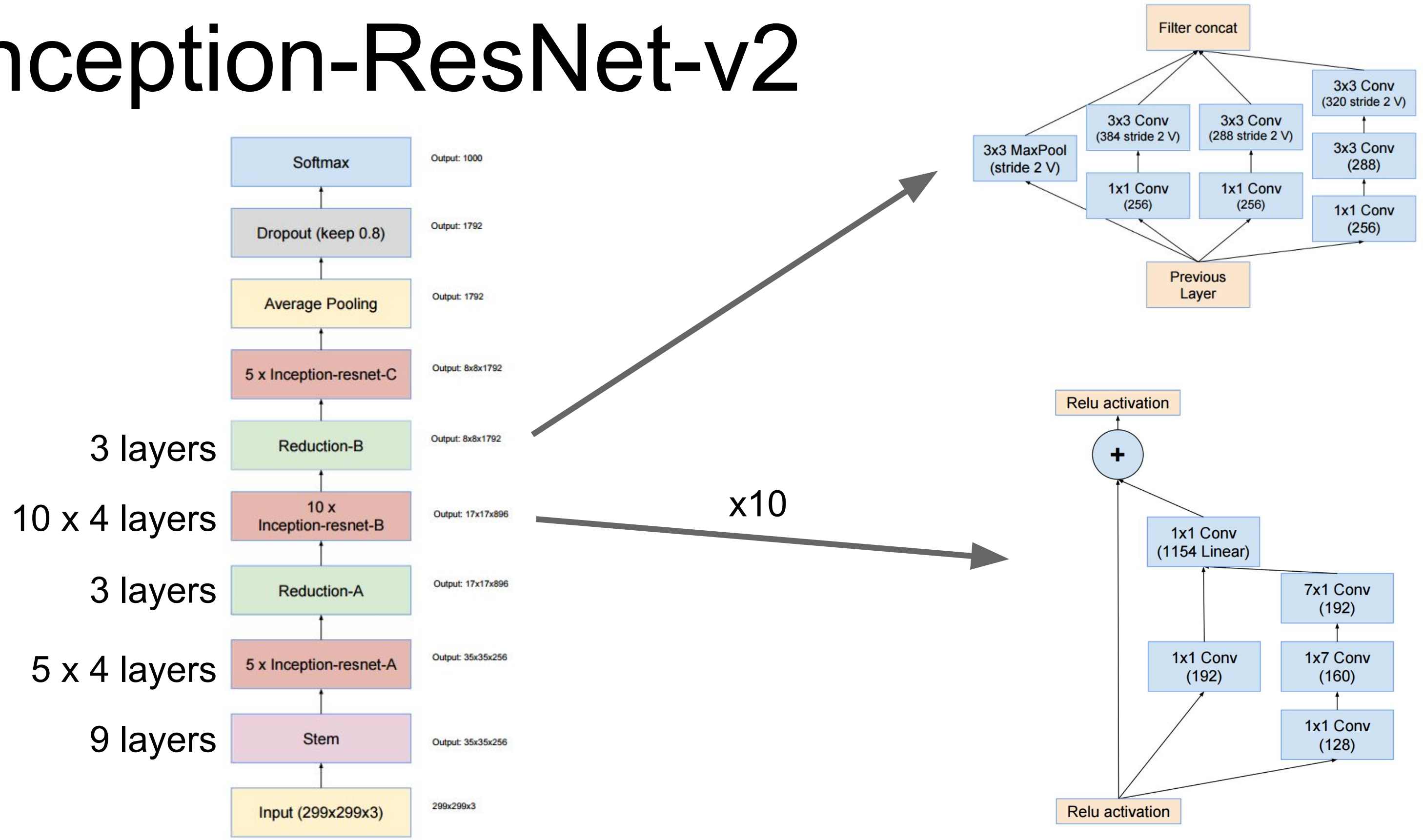
9 layers



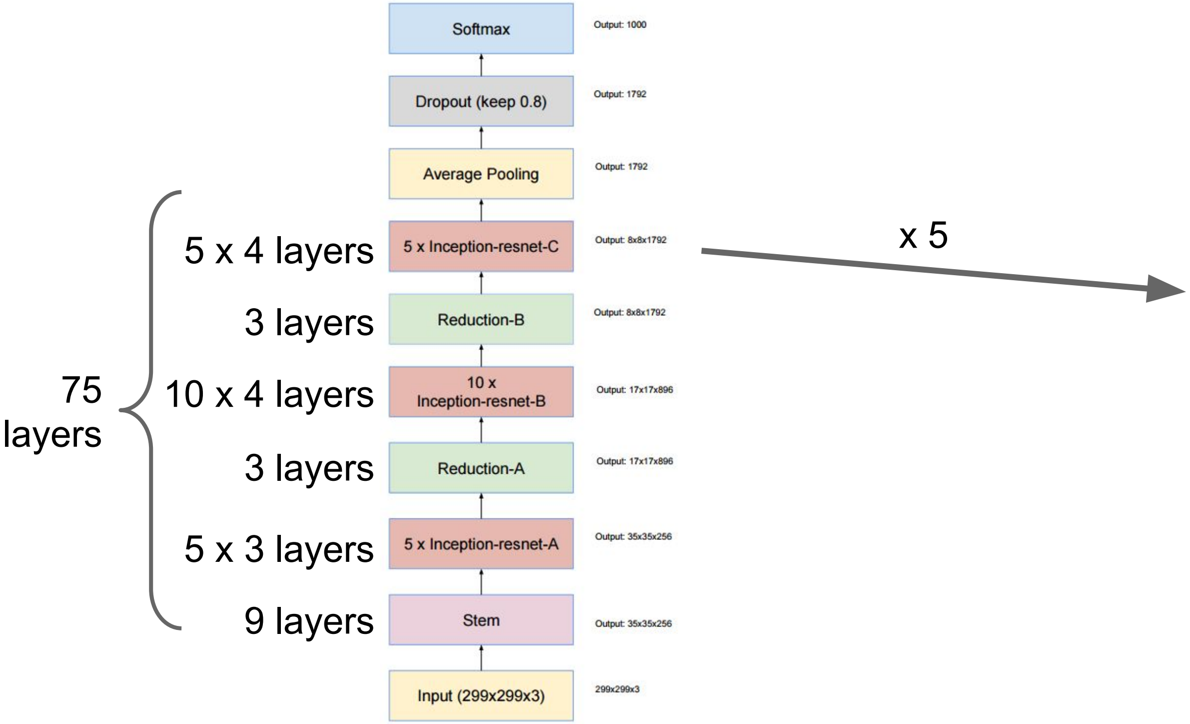
Inception-ResNet-v2

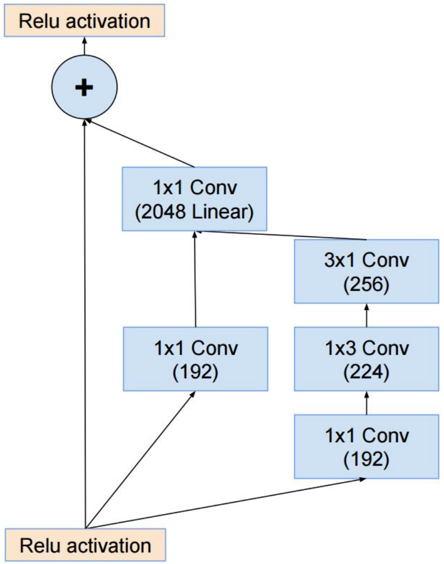


Inception-ResNet-v2

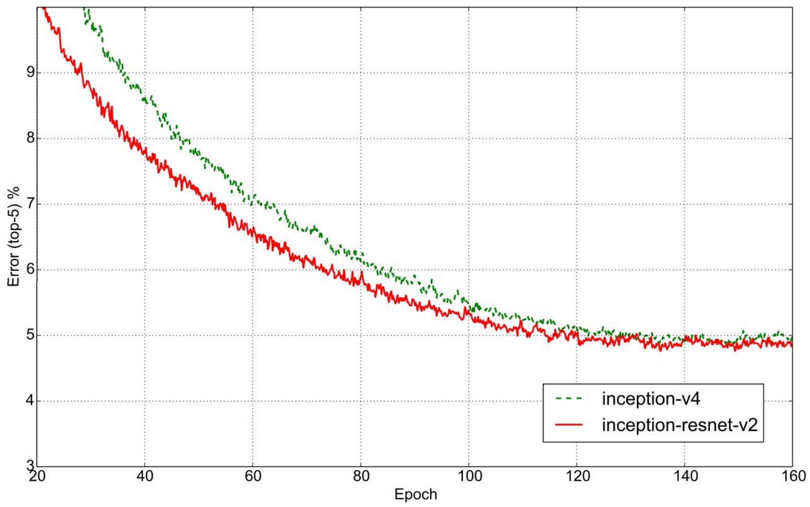


Inception-ResNet-v2





Inception-ResNet-v2



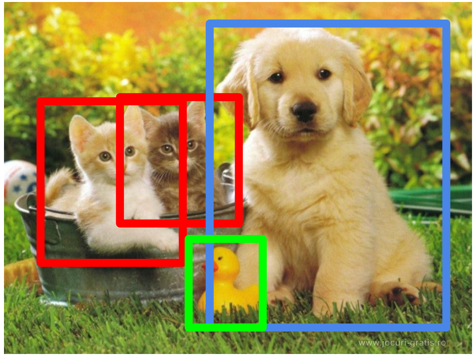
Residual and non-residual converge to similar value, but residual learns faster

Today

- Segmentation
  - Semantic Segmentation
  - Instance Segmentation
- (Soft) Attention
  - Discrete locations
  - Continuous locations (Spatial Transformers)

Segmentation

Computer Vision Tasks

Classification	Classification + Localization	Object Detection	Segmentation
			
CAT	CAT	CAT, DOG, DUCK	CAT, DOG, DUCK

Single object

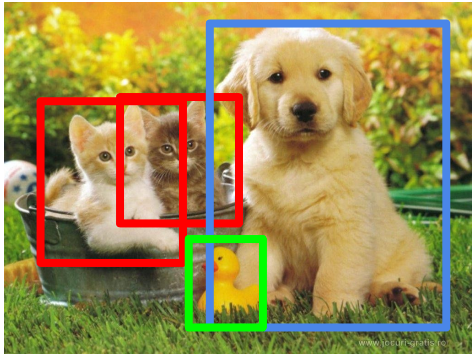
Multiple objects

Computer Vision Tasks

Classification

Classification + Localization

Object Detection



Segmentation

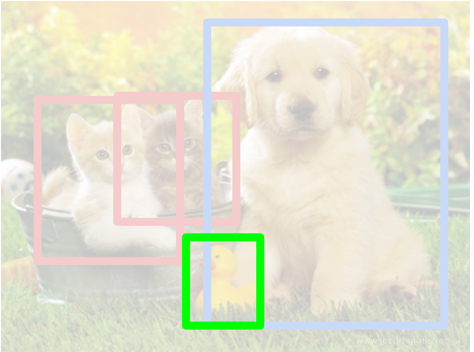
Lecture 8

Computer Vision Tasks

Classification

Classification + Localization

Object Detection



Segmentation

Today

Semantic Segmentation

Label every pixel!

Don't differentiate instances (cows)

Classic computer vision problem

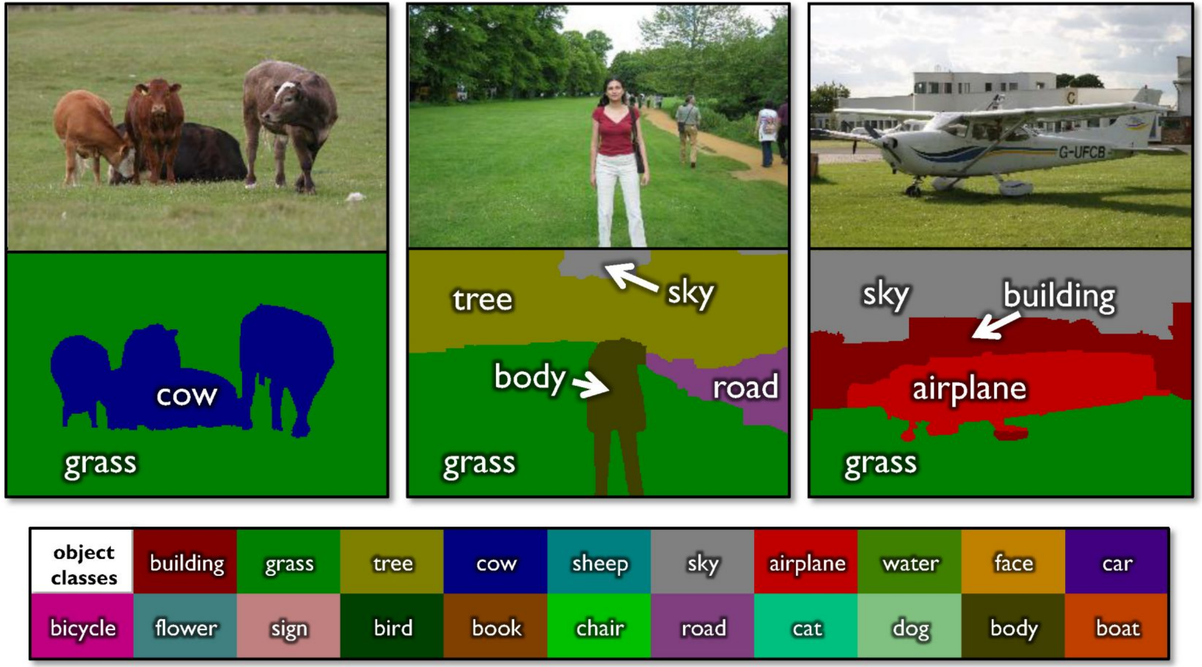


Figure credit: Shotton et al, "TextonBoost for Image Understanding: Multi-Class Object Recognition and Segmentation by Jointly Modeling Texture, Layout, and Context", IJCV 2007

Instance Segmentation

Detect instances, give category, label pixels

“simultaneous detection and segmentation” (SDS)

Lots of recent work (MS-COCO)

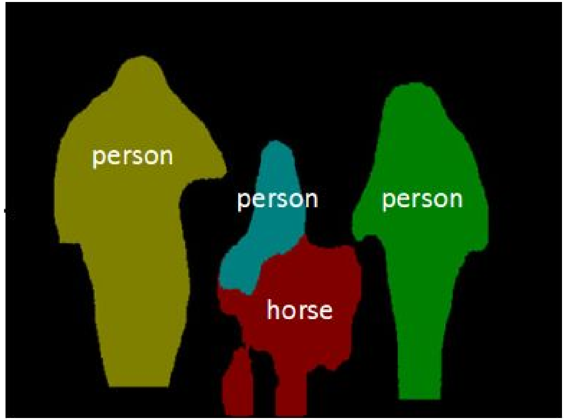
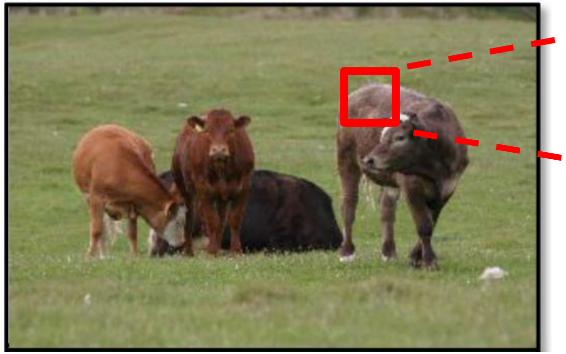


Figure credit: Dai et al, “Instance-aware Semantic Segmentation via Multi-task Network Cascades”, arXiv 2015

Semantic Segmentation

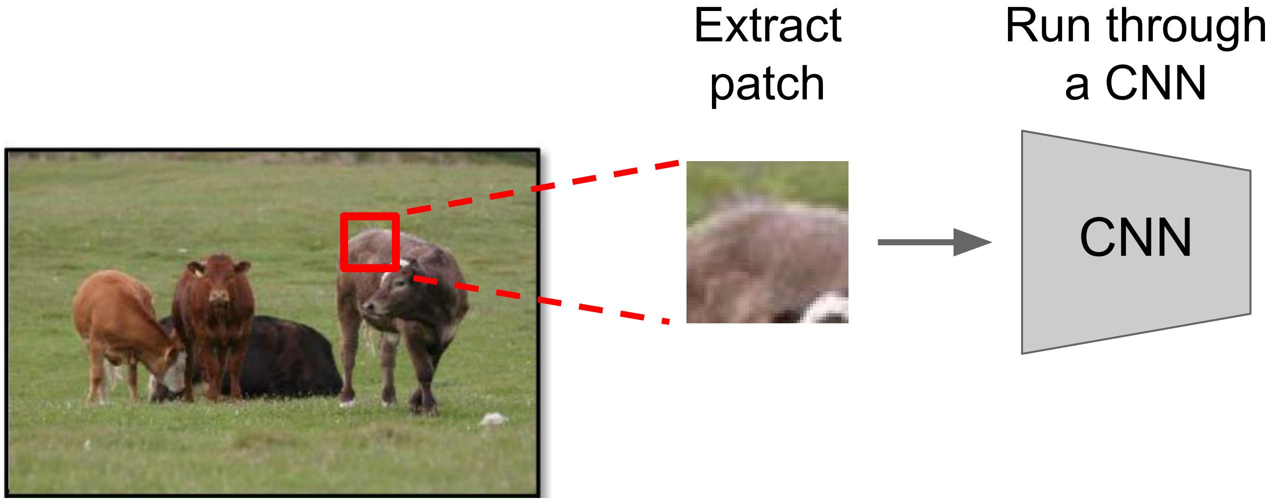
Semantic Segmentation

Semantic Segmentation

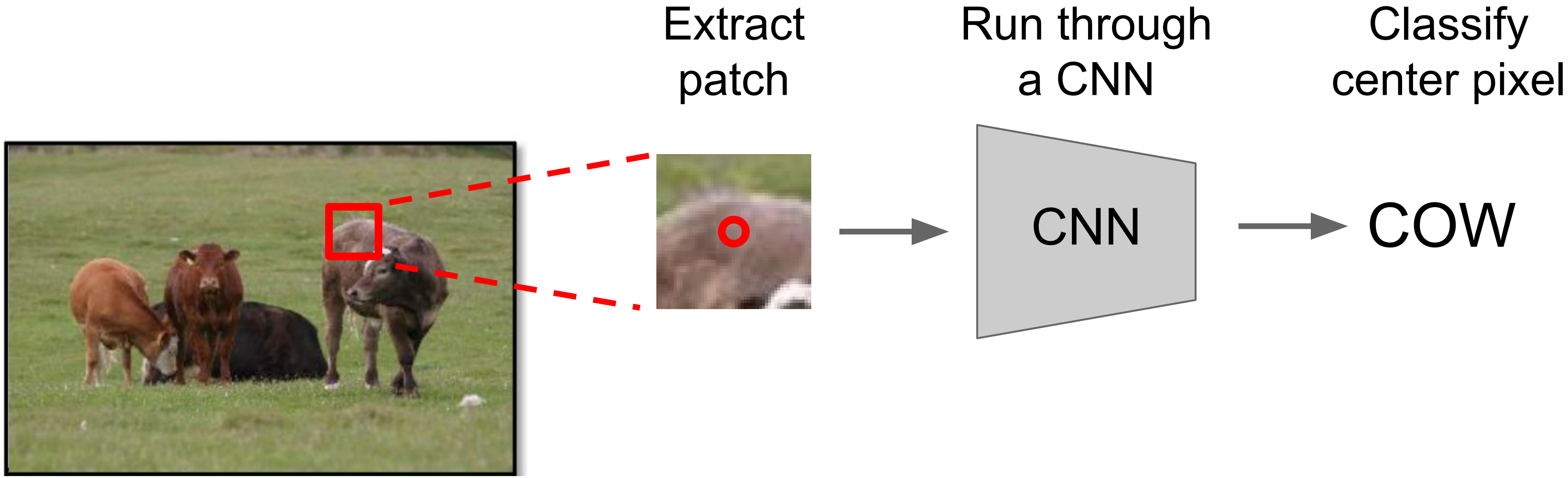


Extract patch

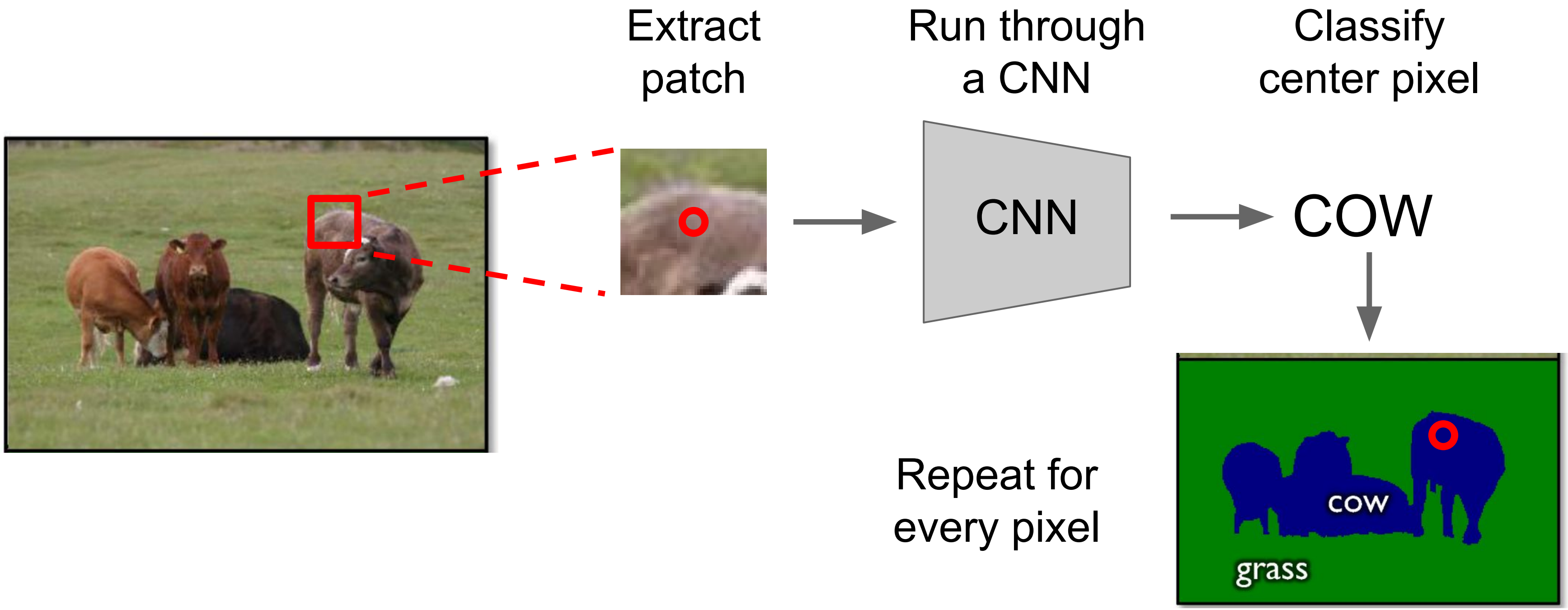
Semantic Segmentation



Semantic Segmentation

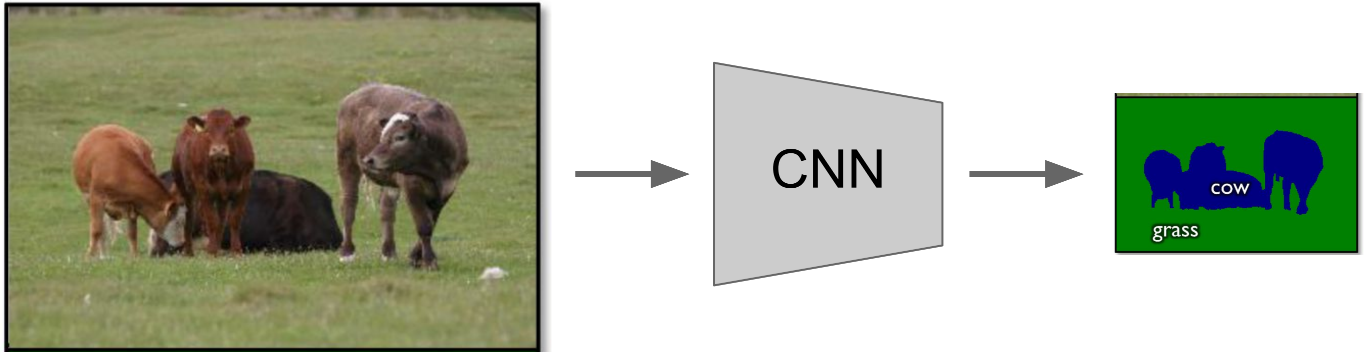


Semantic Segmentation



Semantic Segmentation

Run “fully convolutional” network
to get all pixels at once



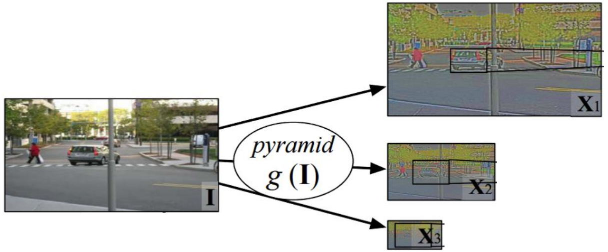
Smaller output
due to pooling

Semantic Segmentation: Multi-Scale

Farabet et al, “Learning Hierarchical Features for Scene Labeling,” TPAMI 2013

Semantic Segmentation: Multi-Scale

Resize image to
multiple scales

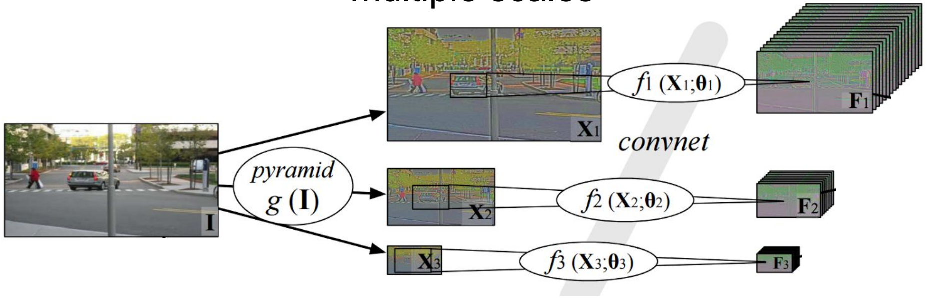


Farabet et al, “Learning Hierarchical Features for Scene Labeling,” TPAMI 2013

Semantic Segmentation: Multi-Scale

Resize image to multiple scales

Run one CNN per scale



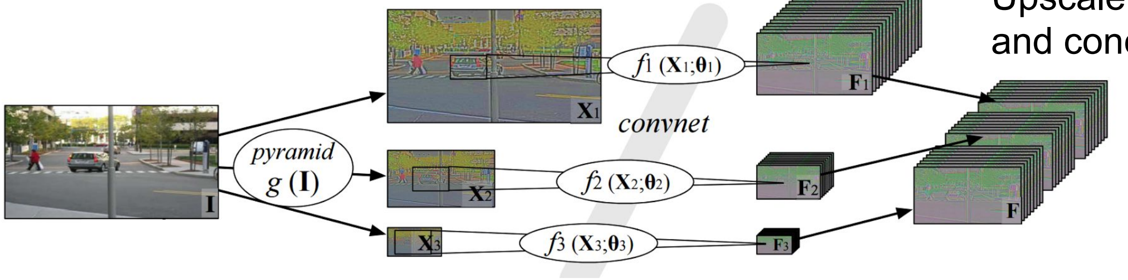
Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013

Semantic Segmentation: Multi-Scale

Resize image to multiple scales

Run one CNN per scale

Upscale outputs and concatenate



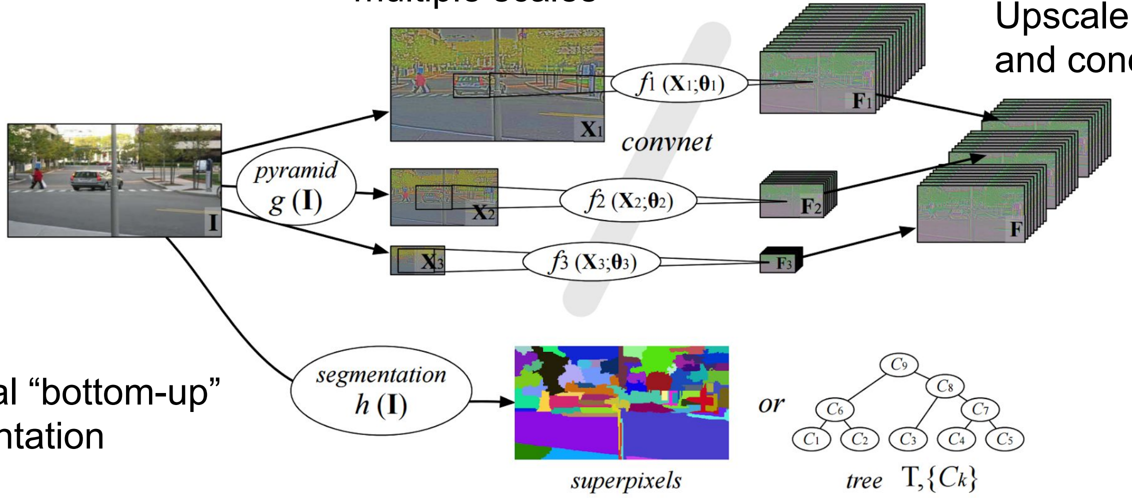
Farabet et al, “Learning Hierarchical Features for Scene Labeling,” TPAMI 2013

Semantic Segmentation: Multi-Scale

Resize image to multiple scales

Run one CNN per scale

Upscale outputs and concatenate



External “bottom-up” segmentation

Farabet et al, “Learning Hierarchical Features for Scene Labeling,” TPAMI 2013

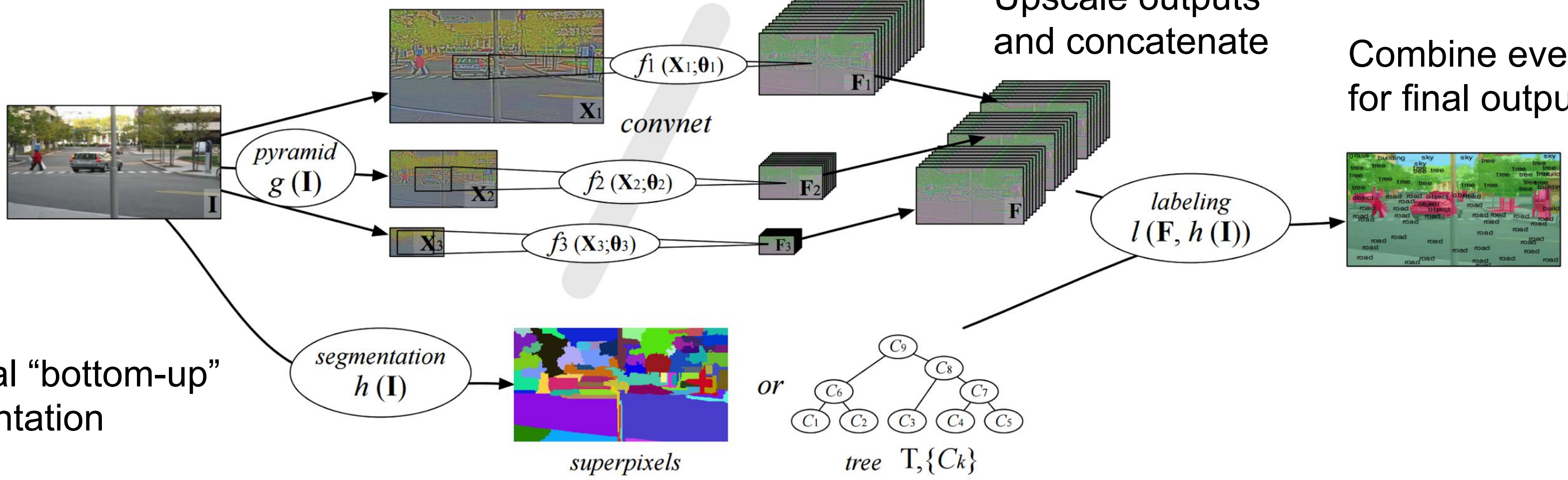
Semantic Segmentation: Multi-Scale

Resize image to multiple scales

Run one CNN per scale

Upscale outputs and concatenate

Combine everything for final outputs

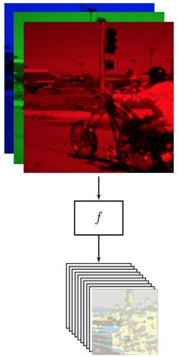


External "bottom-up" segmentation

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013

Semantic Segmentation: Refinement

Apply CNN once
to get labels

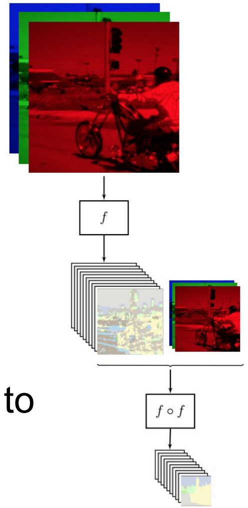


Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Semantic Segmentation: Refinement

Apply CNN once to get labels

Apply AGAIN to refine labels



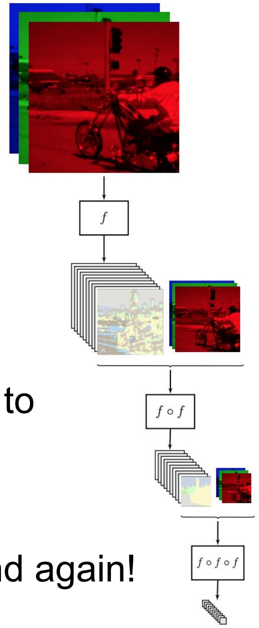
Pinheiro and Collobert, “Recurrent Convolutional Neural Networks for Scene Labeling”, ICML 2014

Semantic Segmentation: Refinement

Apply CNN once to get labels

Apply AGAIN to refine labels

And again!



Pinheiro and Collobert, “Recurrent Convolutional Neural Networks for Scene Labeling”, ICML 2014

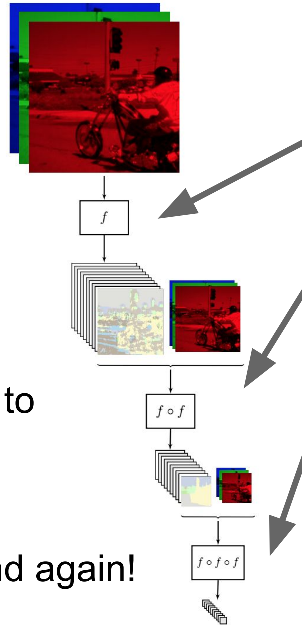
Semantic Segmentation: Refinement

Same CNN weights:
recurrent convolutional network

Apply CNN once
to get labels

Apply AGAIN to
refine labels

And again!



Pinheiro and Collobert, “Recurrent Convolutional Neural Networks for Scene Labeling”, ICML 2014

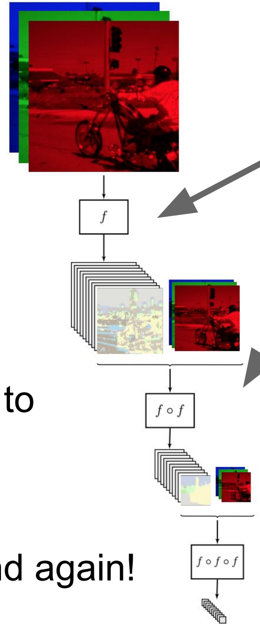
Semantic Segmentation: Refinement

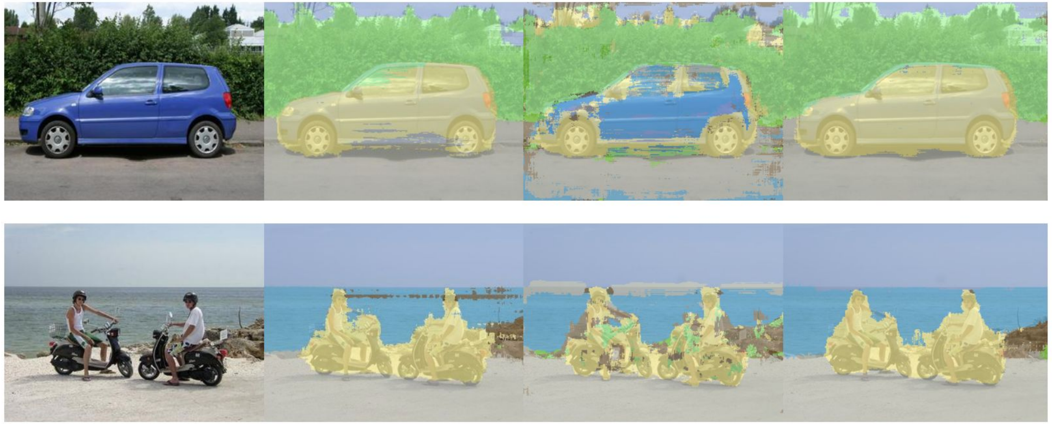
Same CNN weights:
recurrent convolutional network

Apply CNN once to get labels

Apply AGAIN to refine labels

And again!

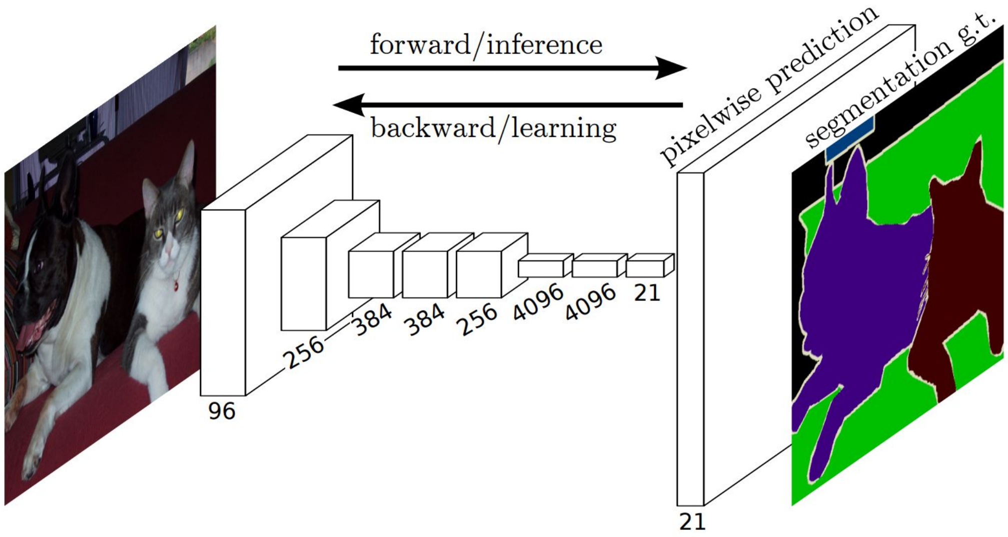




More iterations improve results

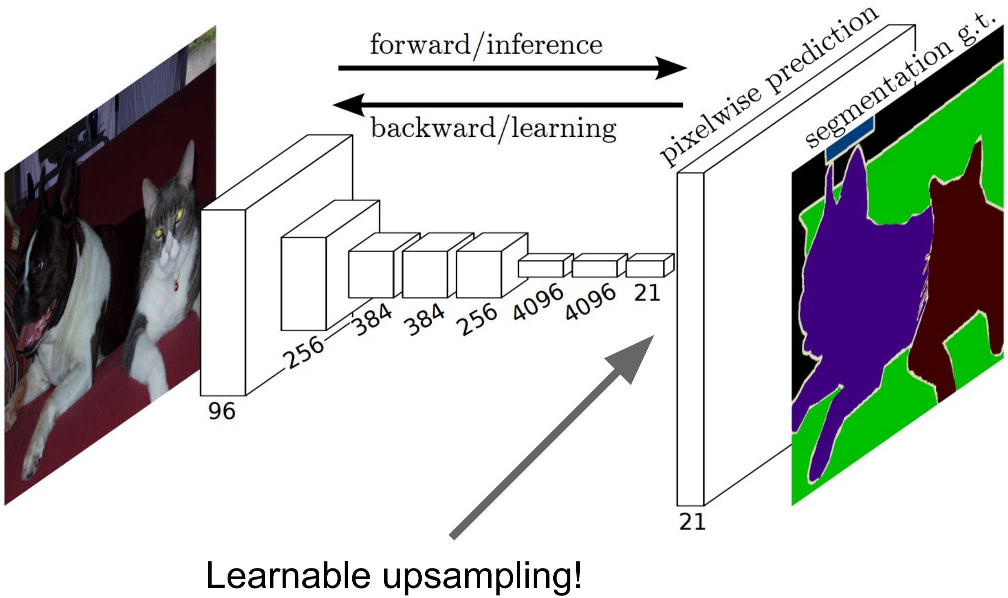
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Semantic Segmentation: Upsampling



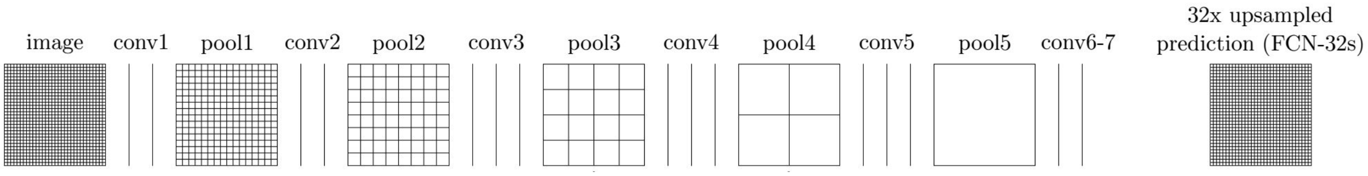
Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015

Semantic Segmentation: Upsampling



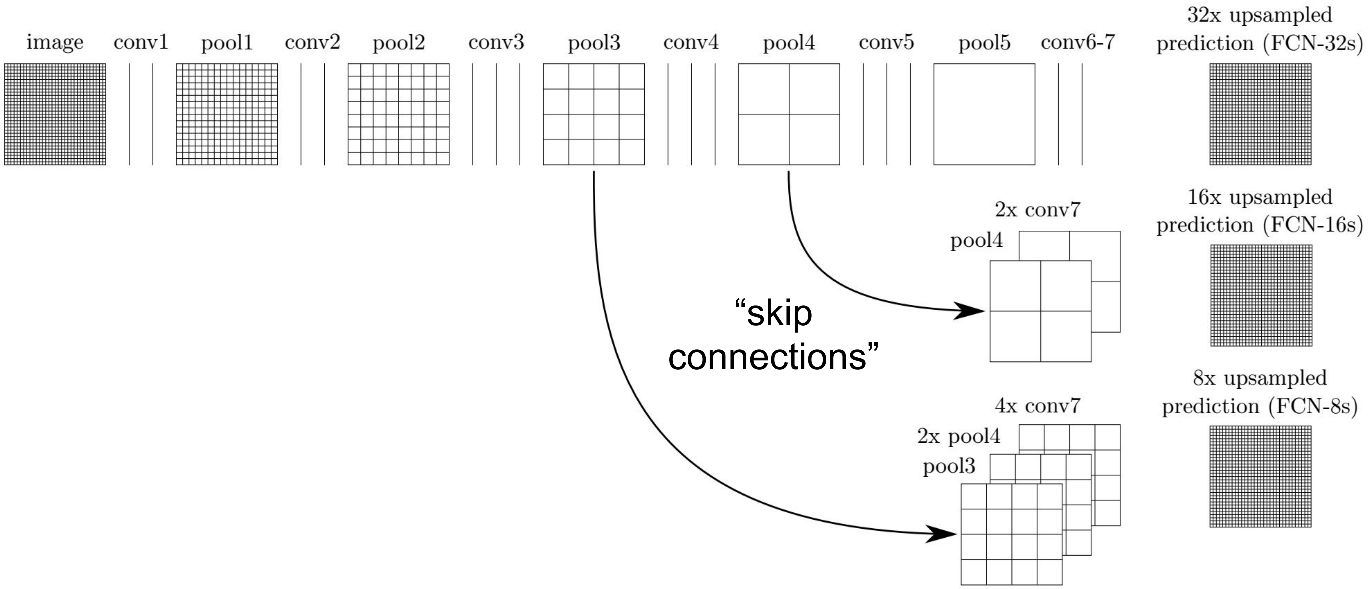
Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015

Semantic Segmentation: Upsampling



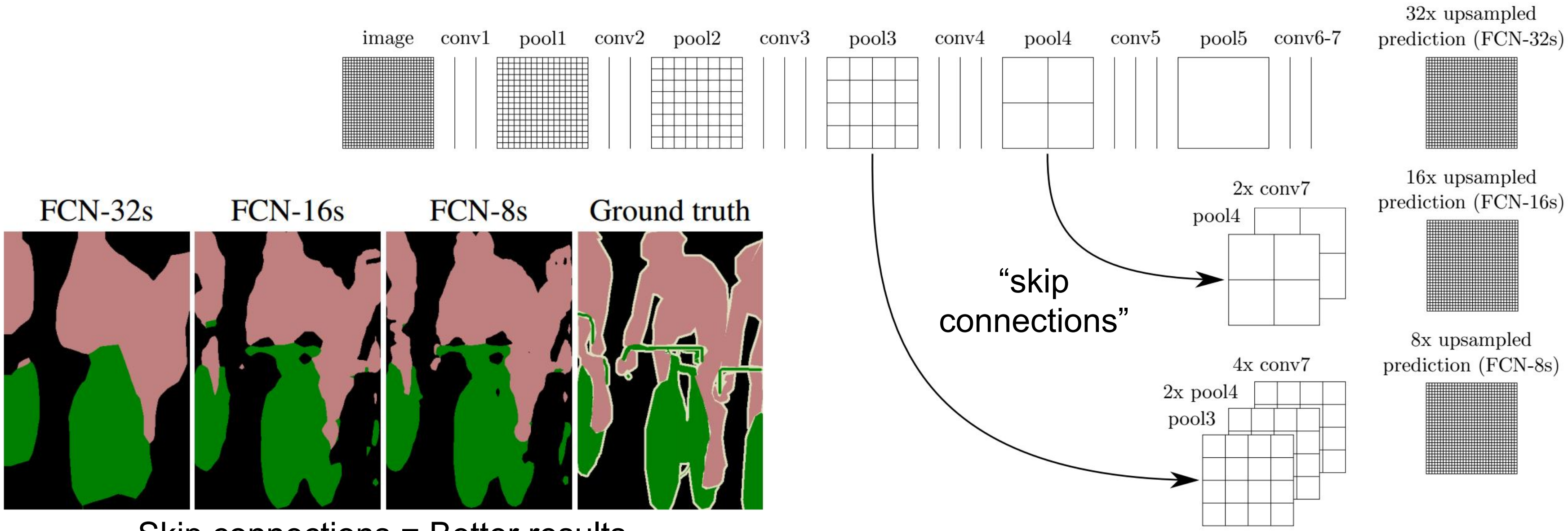
Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015

Semantic Segmentation: Upsampling



Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015

Semantic Segmentation: Upsampling

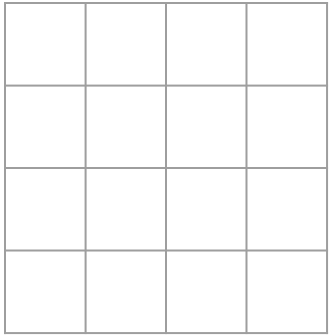


Skip connections = Better results

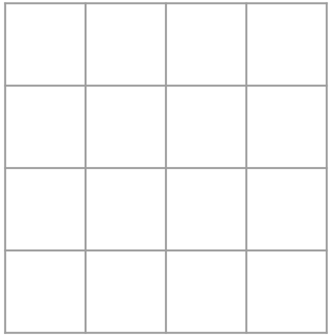
Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015

Learnable Upsampling: “Deconvolution”

Typical 3 x 3 convolution, stride 1 pad 1



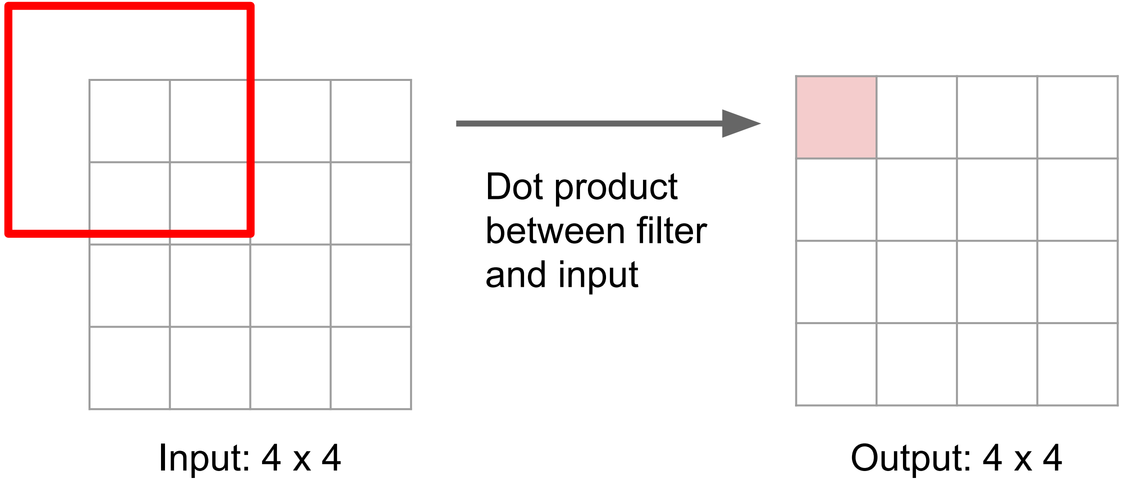
Input: 4 x 4



Output: 4 x 4

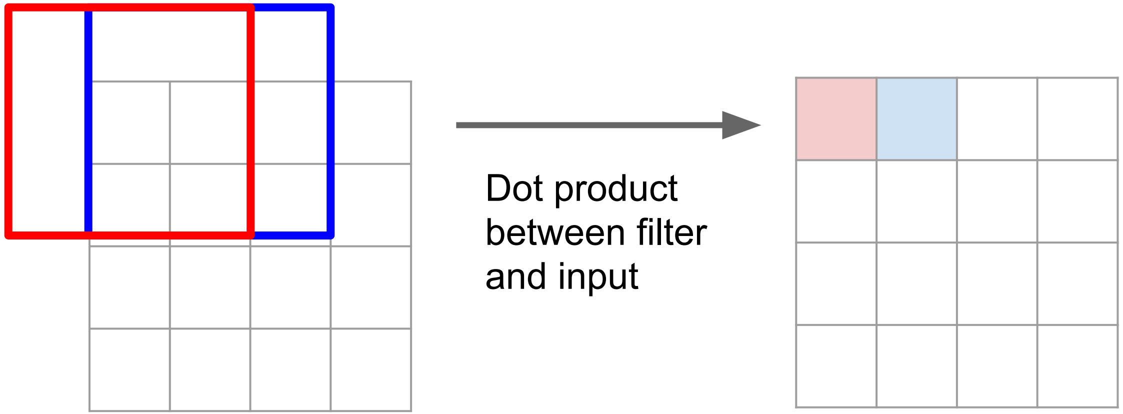
Learnable Upsampling: "Deconvolution"

Typical 3 x 3 convolution, stride 1 pad 1



Learnable Upsampling: “Deconvolution”

Typical 3 x 3 convolution, stride 1 pad 1

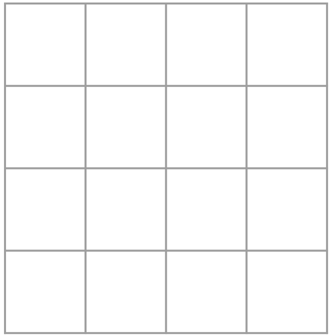


Input: 4 x 4

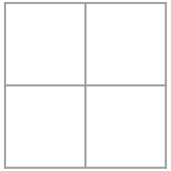
Output: 4 x 4

Learnable Upsampling: "Deconvolution"

Typical 3 x 3 convolution, **stride 2** pad 1



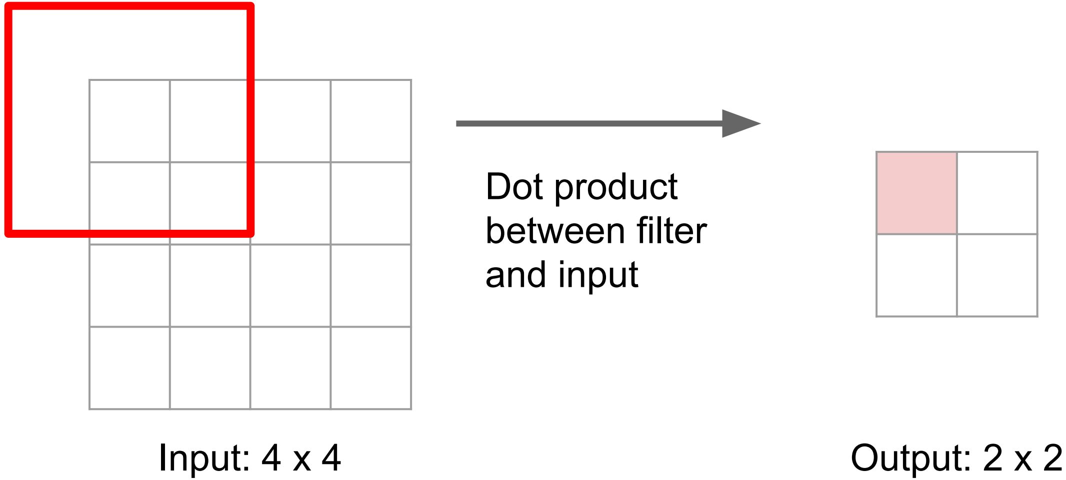
Input: 4 x 4



Output: 2 x 2

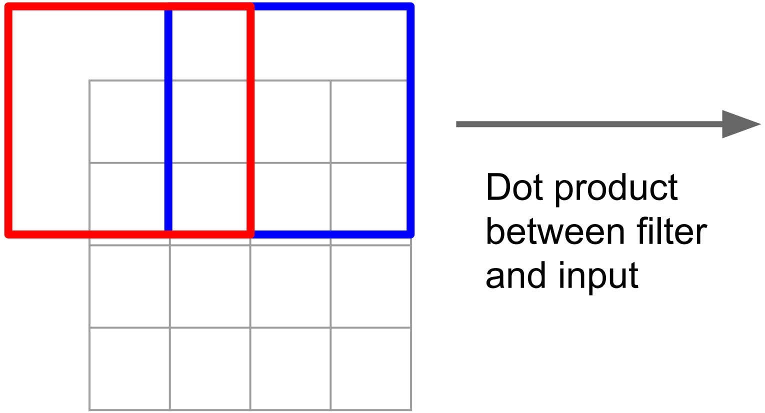
Learnable Upsampling: “Deconvolution”

Typical 3 x 3 convolution, stride 2 pad 1



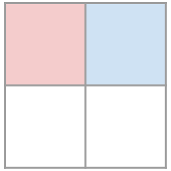
Learnable Upsampling: “Deconvolution”

Typical 3 x 3 convolution, stride 2 pad 1



Input: 4 x 4

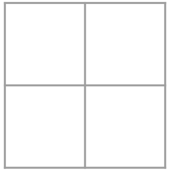
Dot product between filter and input



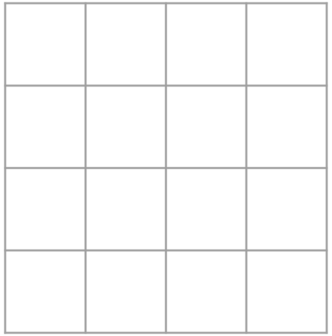
Output: 2 x 2

Learnable Upsampling: “Deconvolution”

3 x 3 “deconvolution”, stride 2 pad 1



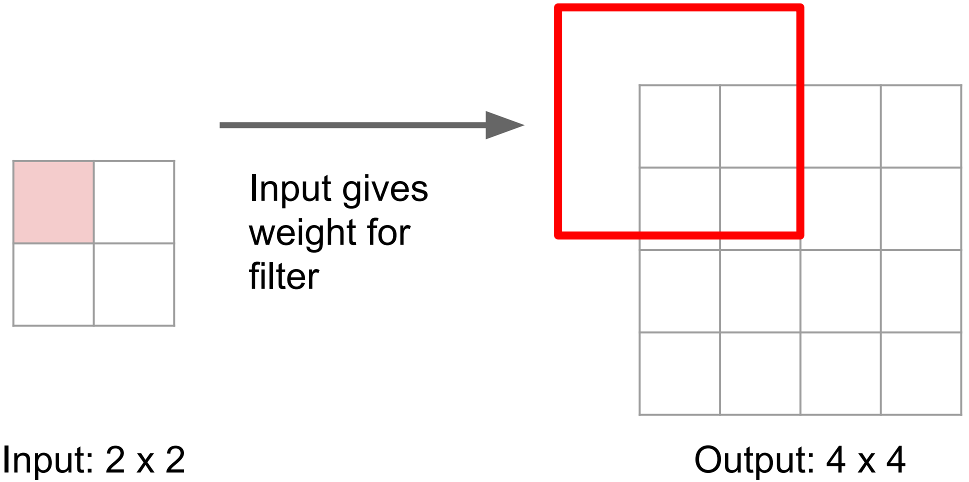
Input: 2 x 2



Output: 4 x 4

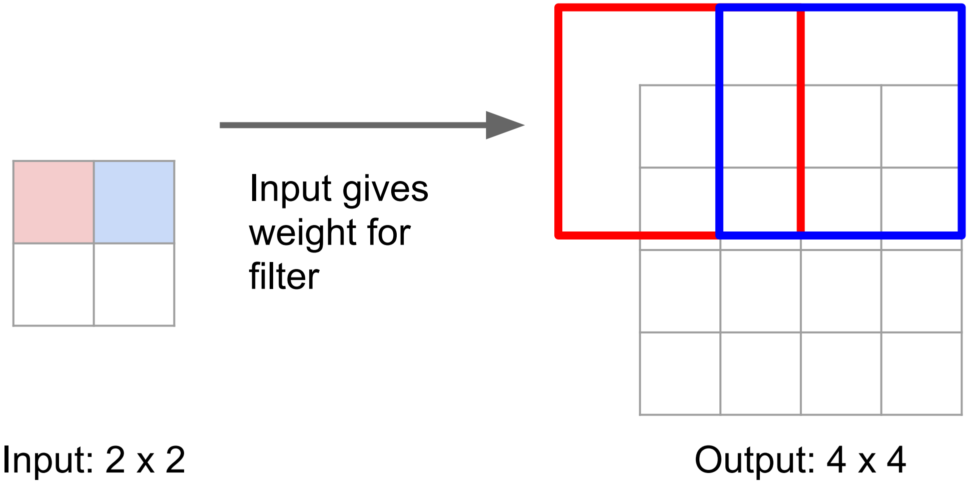
Learnable Upsampling: “Deconvolution”

3 x 3 “deconvolution”, stride 2 pad 1



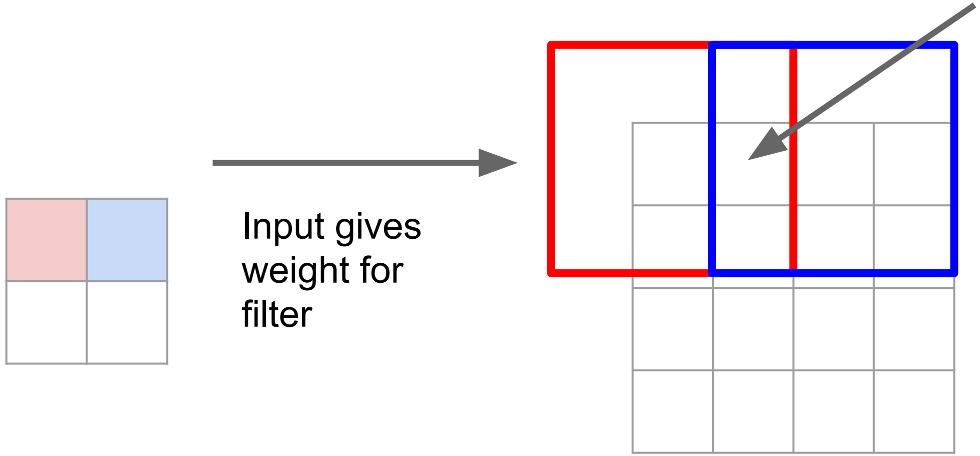
Learnable Upsampling: "Deconvolution"

3 x 3 "deconvolution", stride 2 pad 1

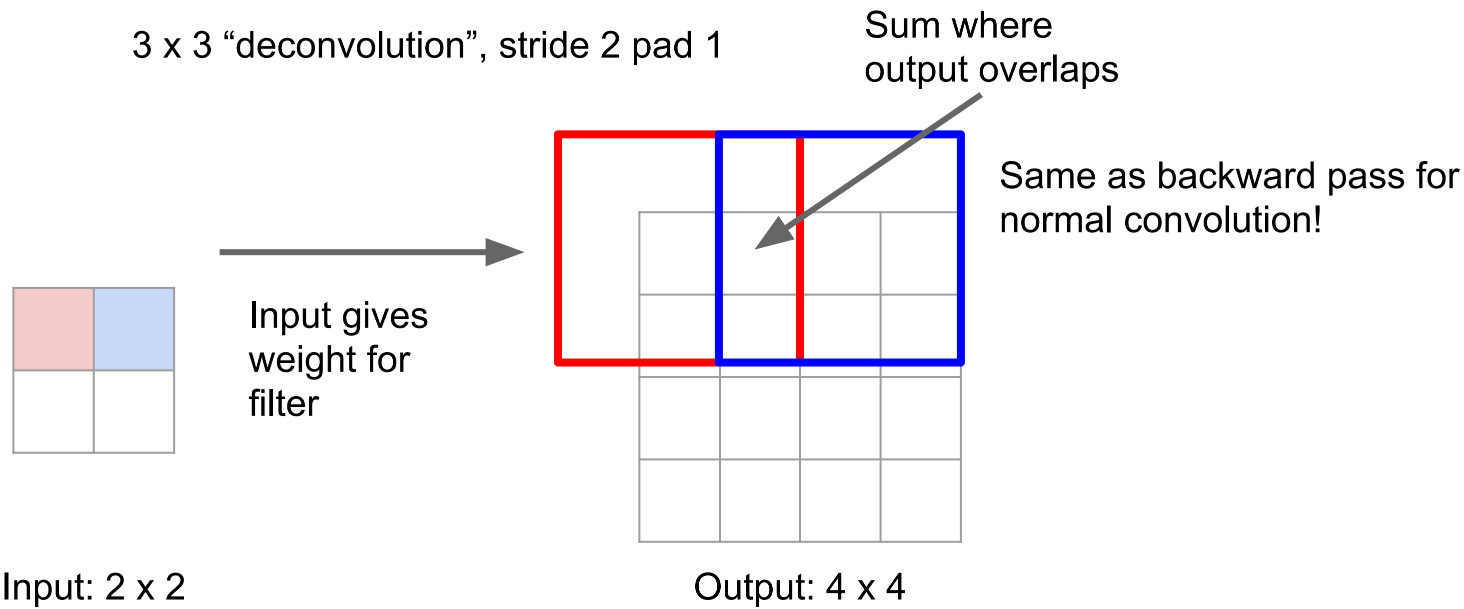


Learnable Upsampling: “Deconvolution”

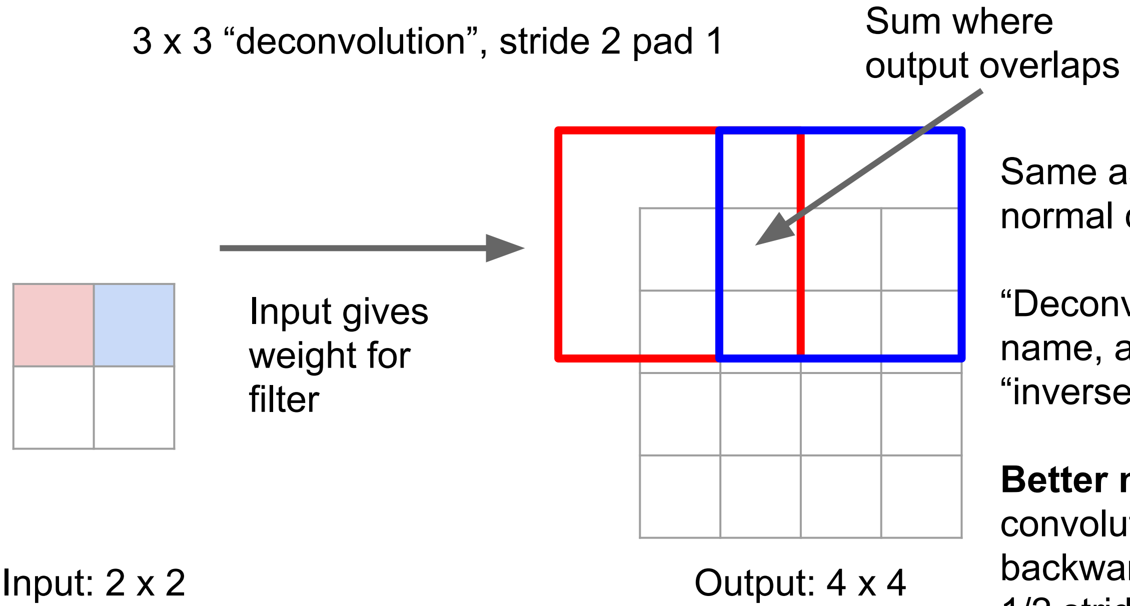
3 x 3 “deconvolution”, stride 2 pad 1



Learnable Upsampling: “Deconvolution”



Learnable Upsampling: “Deconvolution”



Same as backward pass for normal convolution!

“Deconvolution” is a bad name, already defined as “inverse of convolution”

Better names:
convolution transpose,
backward strided convolution,
1/2 strided convolution,
upconvolution

Learnable Upsampling: "Deconvolution"

[1]It is more proper to say "convolutional transpose operation" rather than "deconvolutional" operation. Hence, we will be using the term "convolutional transpose" from now.

Im et al, "Generating images with recurrent adversarial networks", arXiv 2016

A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions)

Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

"Deconvolution" is a bad name, already defined as "inverse of convolution"

Better names:
convolution transpose,
backward strided convolution,
1/2 strided convolution,
upconvolution

Learnable Upsampling: "Deconvolution"

Great explanation in appendix

[1]It is more proper to say "convolutional transpose operation" rather than "deconvolutional" operation. Hence, we will be using the term "convolutional transpose" from now.

Im et al, "Generating images with recurrent adversarial networks", arXiv 2016

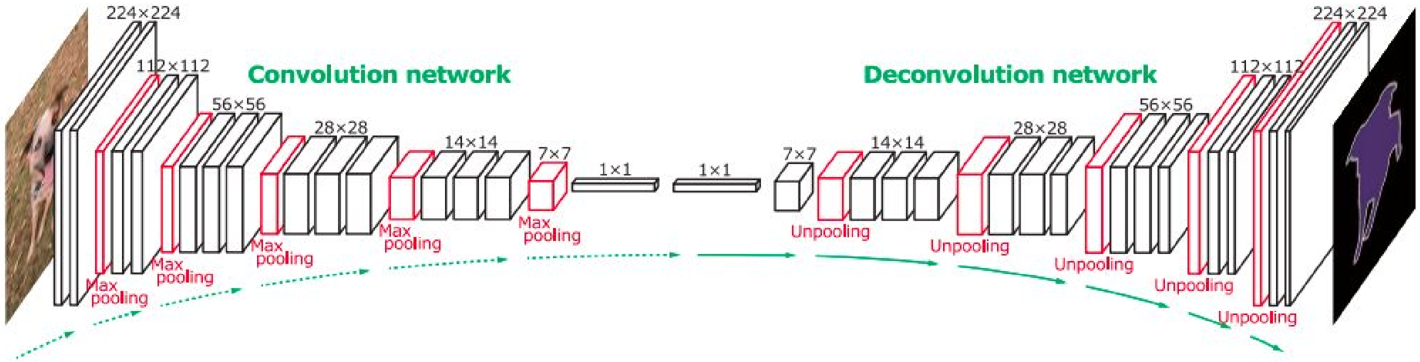
A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions)

Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

"Deconvolution" is a bad name, already defined as "inverse of convolution"

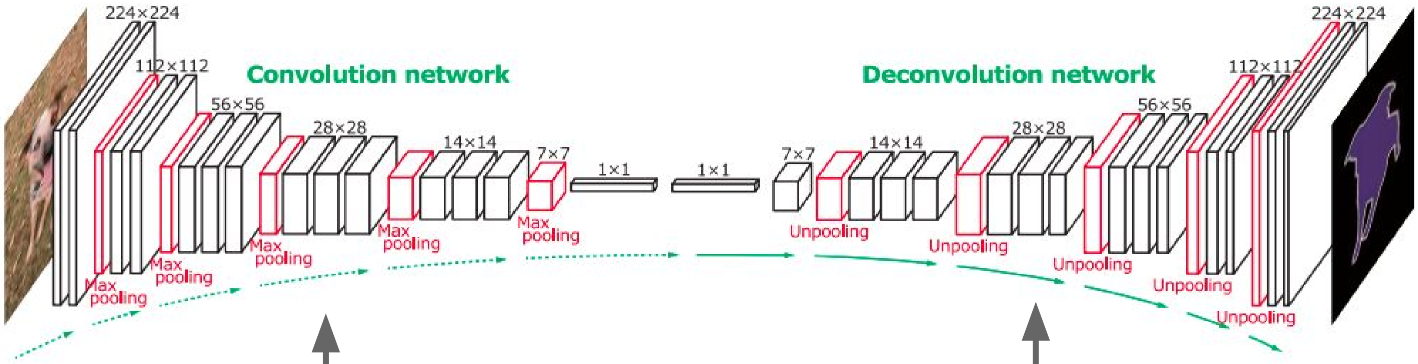
Better names:
convolution transpose,
backward strided convolution,
1/2 strided convolution,
upconvolution

Semantic Segmentation: Upsampling



Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

Semantic Segmentation: Upsampling



Normal VGG

"Upside down" VGG

Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

6 days of training on Titan X…

Instance Segmentation

Instance Segmentation

Detect instances, give category, label pixels

"simultaneous detection and segmentation" (SDS)

Lots of recent work (MS-COCO)

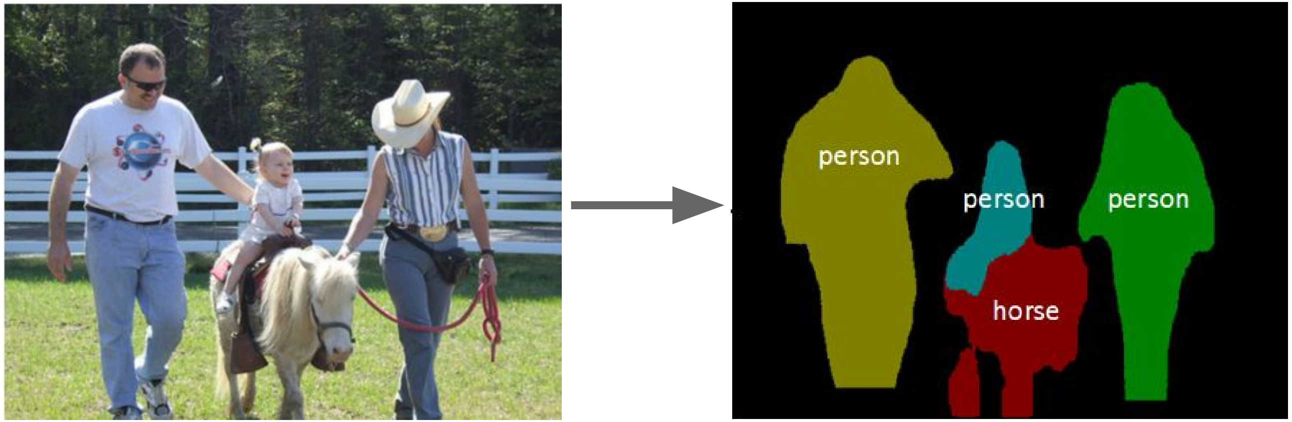


Figure credit: Dai et al, "Instance-aware Semantic Segmentation via Multi-task Network Cascades", arXiv 2015

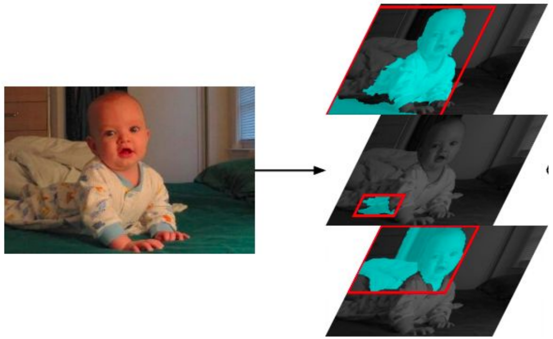
Instance Segmentation

Similar to R-CNN, but with segments

Hariharan et al, “Simultaneous Detection and Segmentation”, ECCV 2014

Instance Segmentation

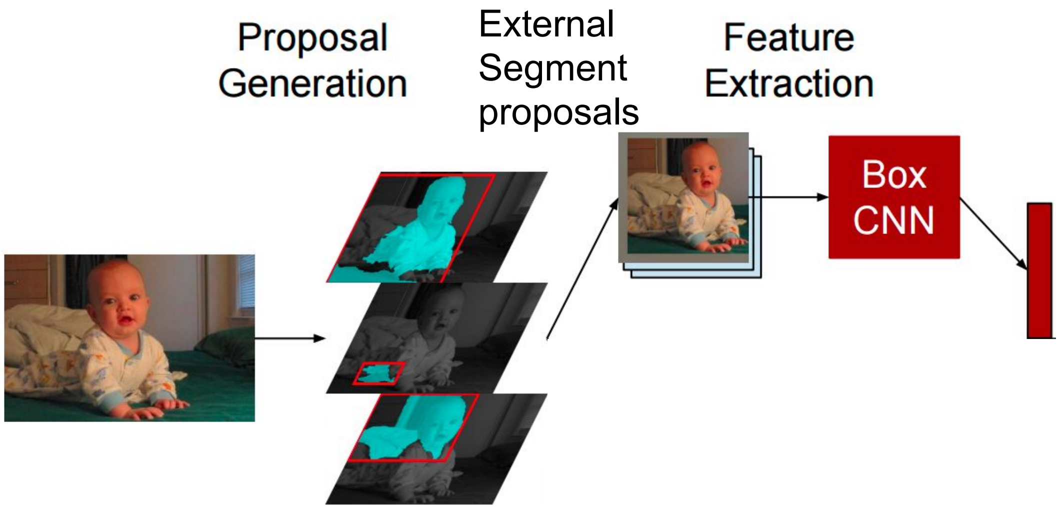
Similar to R-CNN, but with segments



Hariharan et al, "Simultaneous Detection and Segmentation", ECCV 2014

Instance Segmentation

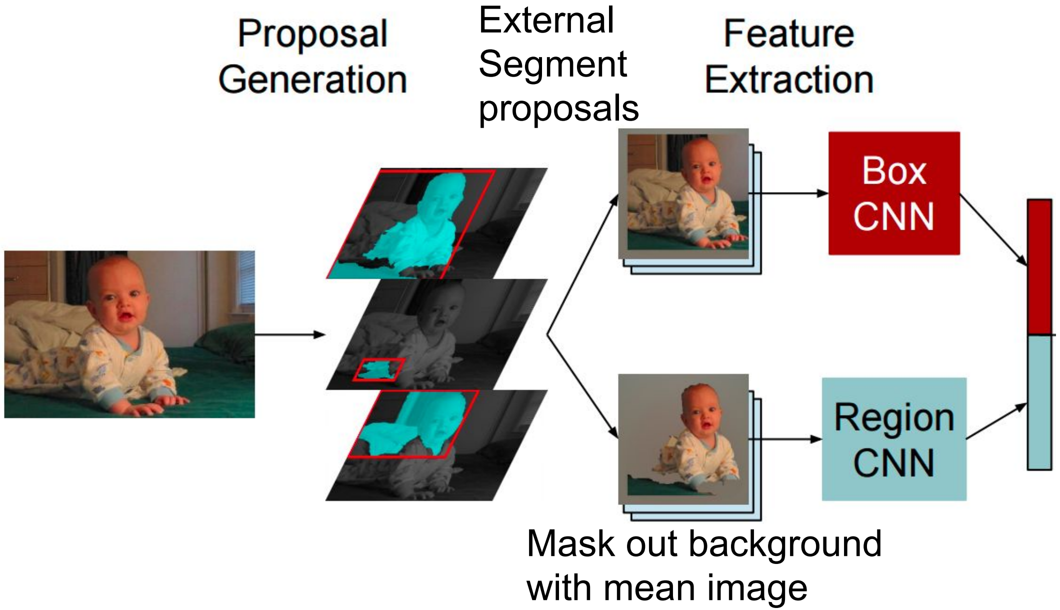
Similar to R-CNN, but with segments



Hariharan et al, "Simultaneous Detection and Segmentation", ECCV 2014

Instance Segmentation

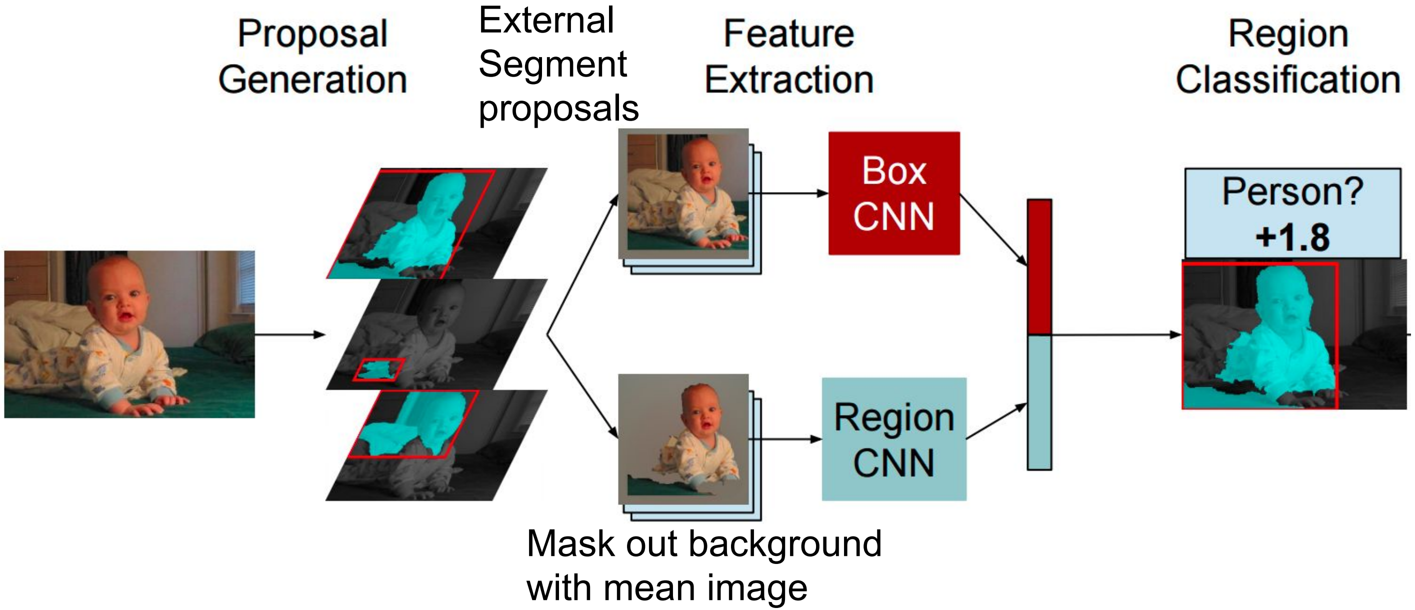
Similar to R-CNN, but with segments



Hariharan et al, “Simultaneous Detection and Segmentation”, ECCV 2014

Instance Segmentation

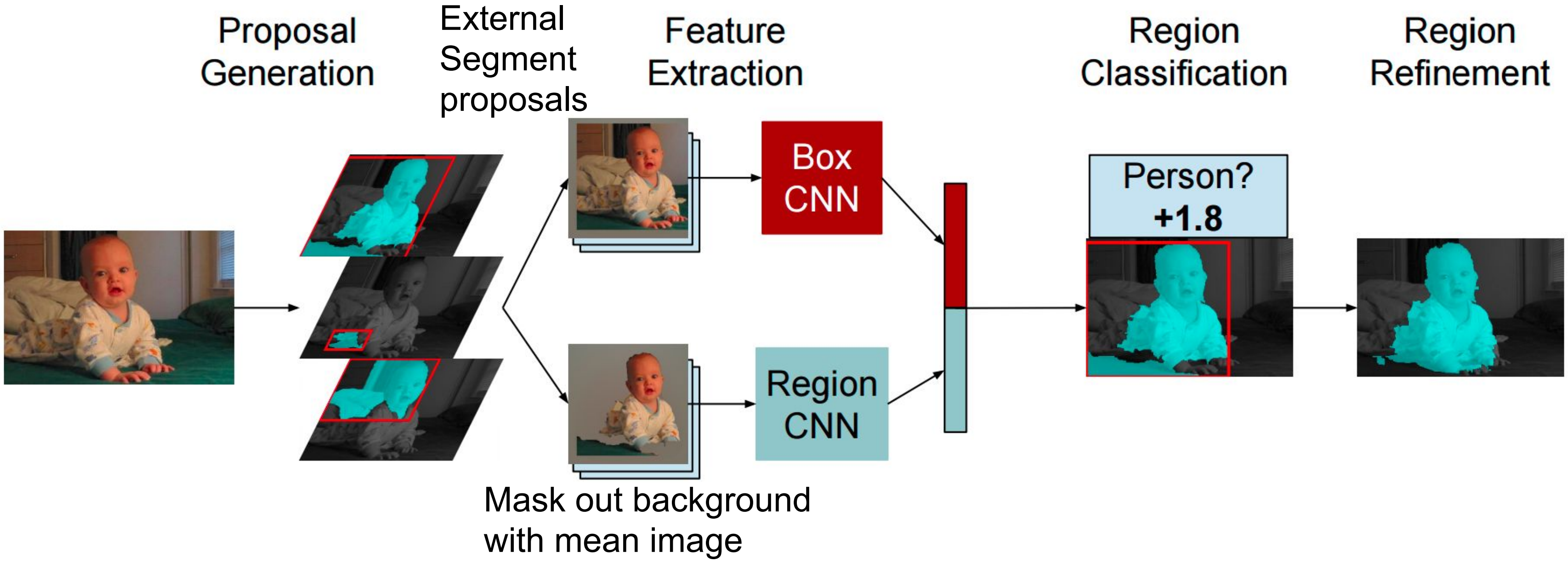
Similar to R-CNN, but with segments



Hariharan et al, “Simultaneous Detection and Segmentation”, ECCV 2014

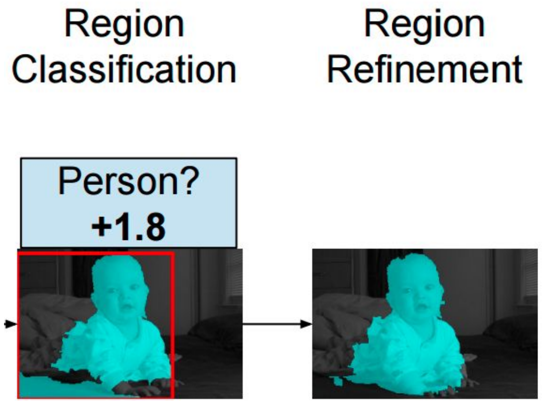
Instance Segmentation

Similar to R-CNN, but with segments



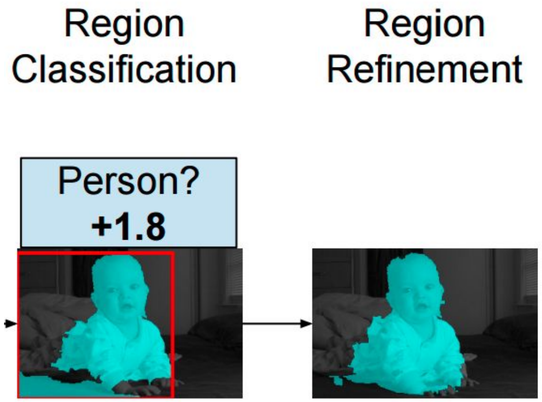
Hariharan et al, “Simultaneous Detection and Segmentation”, ECCV 2014

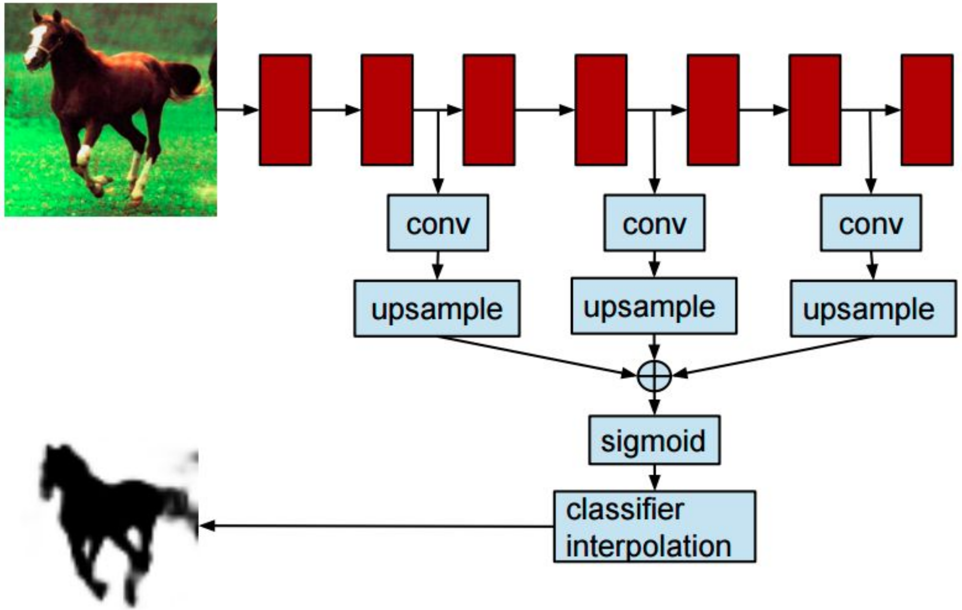
Instance Segmentation: Hypercolumns



Hariharan et al, “Hypercolumns for Object Segmentation and Fine-grained Localization”, CVPR 2015

Instance Segmentation: Hypercolumns





Hariharan et al, “Hypercolumns for Object Segmentation and Fine-grained Localization”, CVPR 2015

Instance Segmentation: Cascades

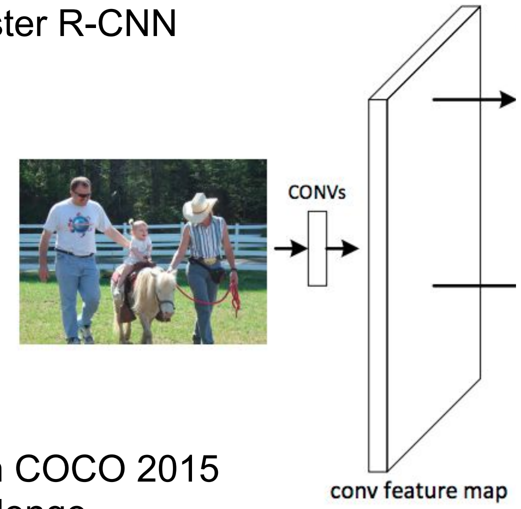
Similar to
Faster R-CNN

Won COCO 2015
challenge
(with ResNet)

Dai et al, "Instance-aware Semantic Segmentation via Multi-task Network Cascades", arXiv 2015

Instance Segmentation: Cascades

Similar to
Faster R-CNN



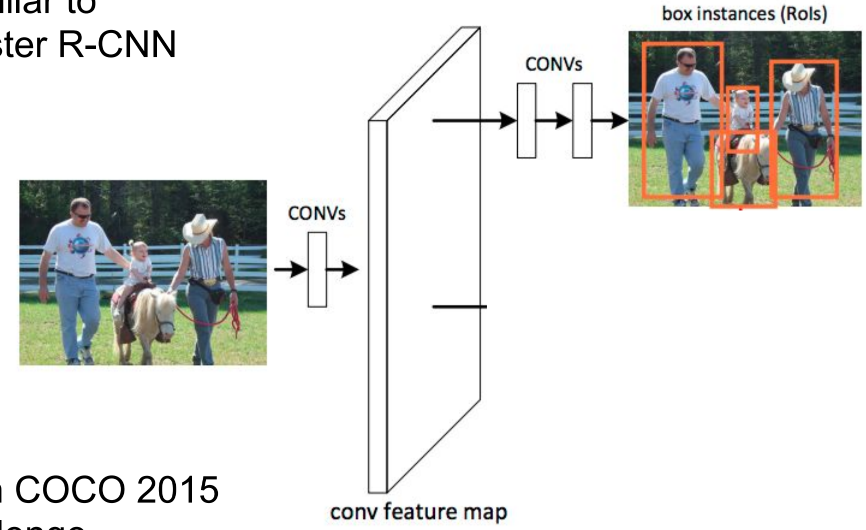
Won COCO 2015
challenge
(with ResNet)

Dai et al, "Instance-aware Semantic Segmentation via Multi-task Network Cascades", arXiv 2015

Instance Segmentation: Cascades

Similar to
Faster R-CNN

Region proposal network (RPN)

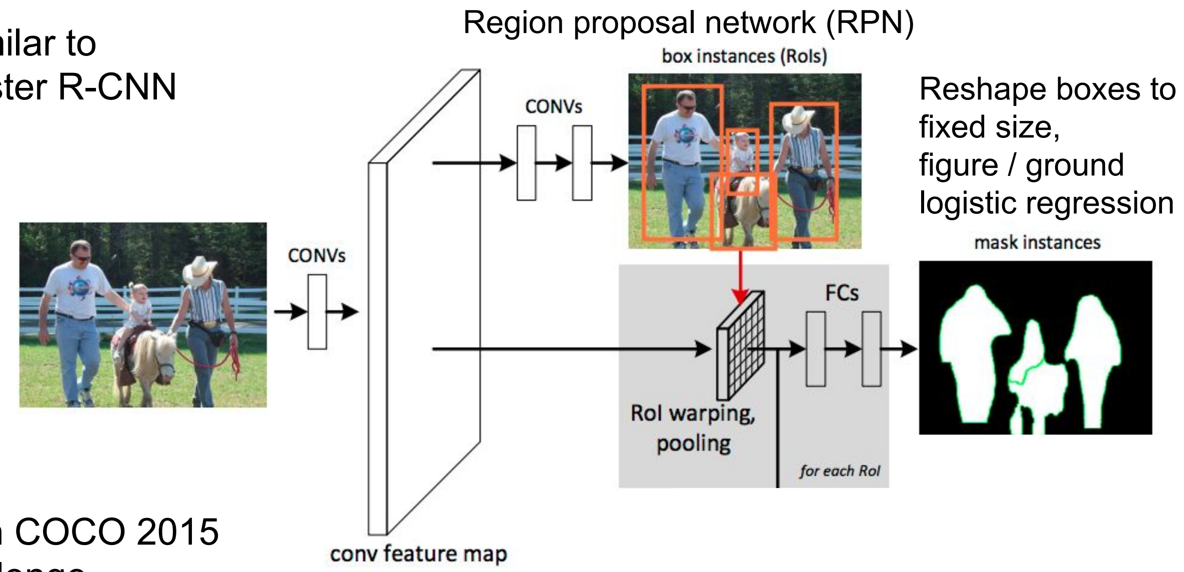


Won COCO 2015
challenge
(with ResNet)

Dai et al, "Instance-aware Semantic Segmentation via Multi-task Network Cascades", arXiv 2015

Instance Segmentation: Cascades

Similar to
Faster R-CNN



Won COCO 2015
challenge
(with ResNet)

Dai et al, “Instance-aware Semantic Segmentation via Multi-task Network Cascades”, arXiv 2015

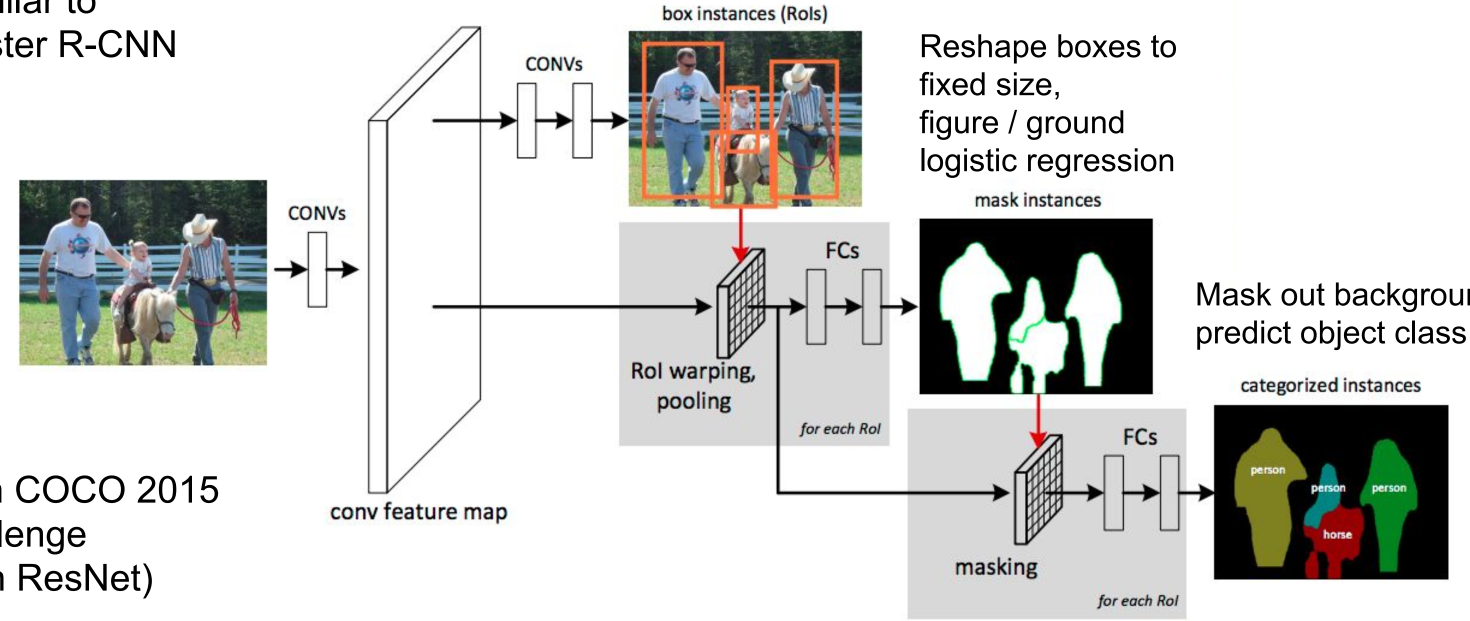
Instance Segmentation: Cascades

Similar to
Faster R-CNN

Region proposal network (RPN)

Reshape boxes to
fixed size,
figure / ground
logistic regression

Mask out background,
predict object class



Won COCO 2015
challenge
(with ResNet)

Dai et al, “Instance-aware Semantic Segmentation via Multi-task Network Cascades”, arXiv 2015

Instance Segmentation: Cascades

Similar to Faster R-CNN

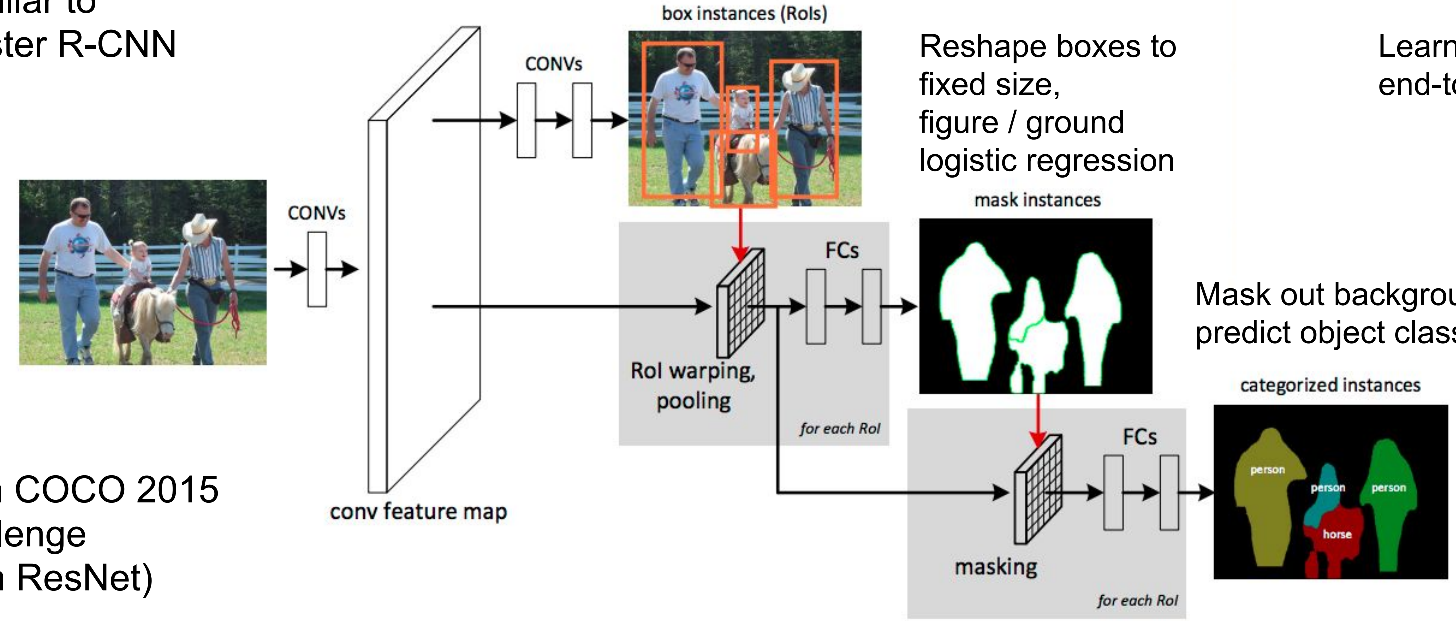
Region proposal network (RPN)

Reshape boxes to fixed size, figure / ground logistic regression

Learn entire model end-to-end!

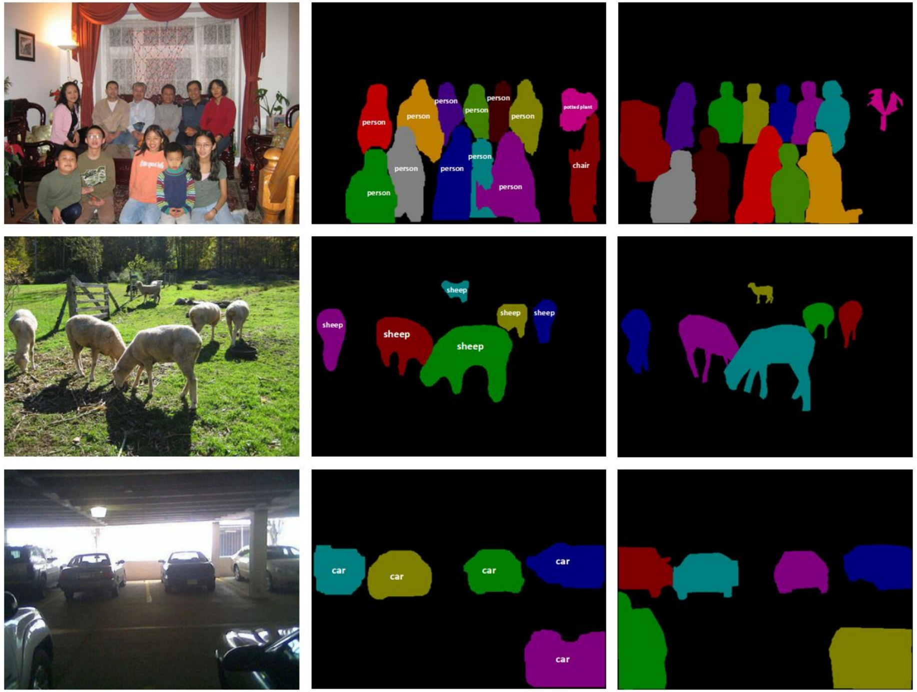
Mask out background, predict object class

Won COCO 2015 challenge (with ResNet)



Dai et al, “Instance-aware Semantic Segmentation via Multi-task Network Cascades”, arXiv 2015

Instance Segmentation: Cascades



Predictions **Ground truth**

Dai et al, “Instance-aware Semantic Segmentation via Multi-task Network Cascades”, arXiv 2015

Segmentation Overview

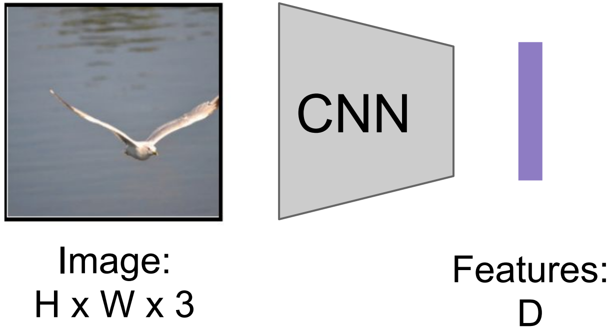
- Semantic segmentation
  - Classify all pixels
  - Fully convolutional models, downsample then upsample
  - Learnable upsampling: fractionally strided convolution
  - Skip connections can help
- Instance Segmentation
  - Detect instance, generate mask
  - Similar pipelines to object detection

Attention Models

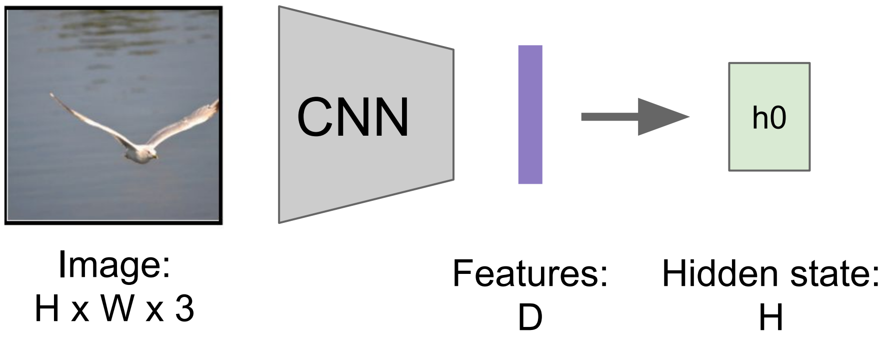
Recall: RNN for Captioning

Image:
H x W x 3

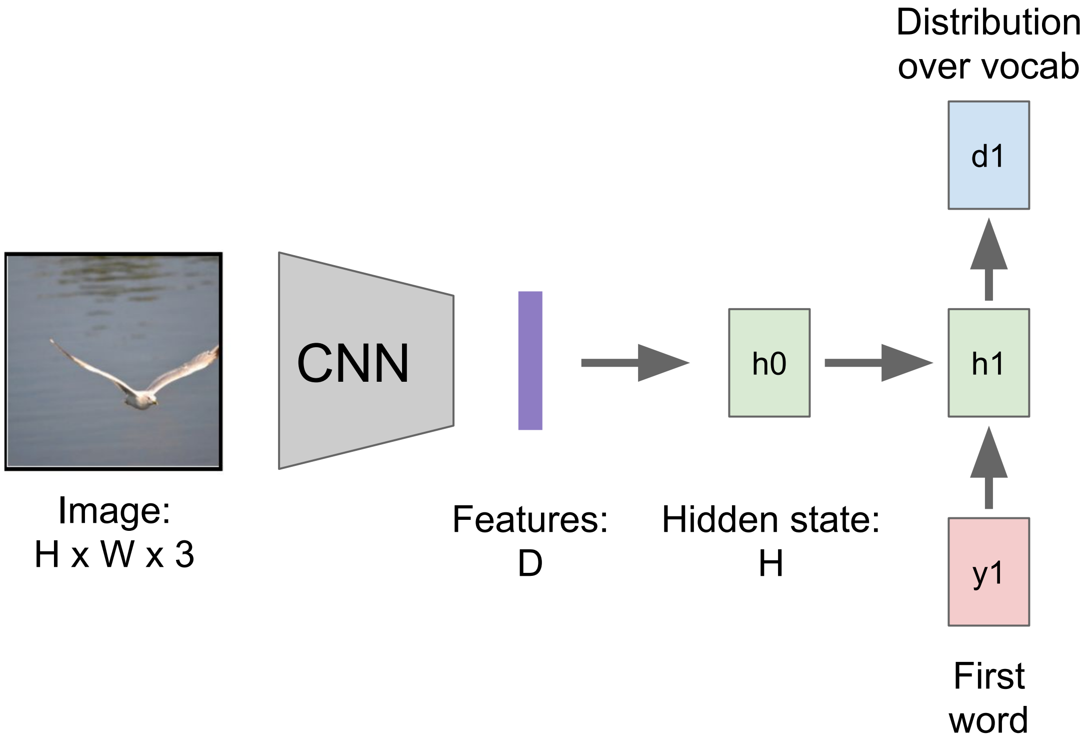
Recall: RNN for Captioning



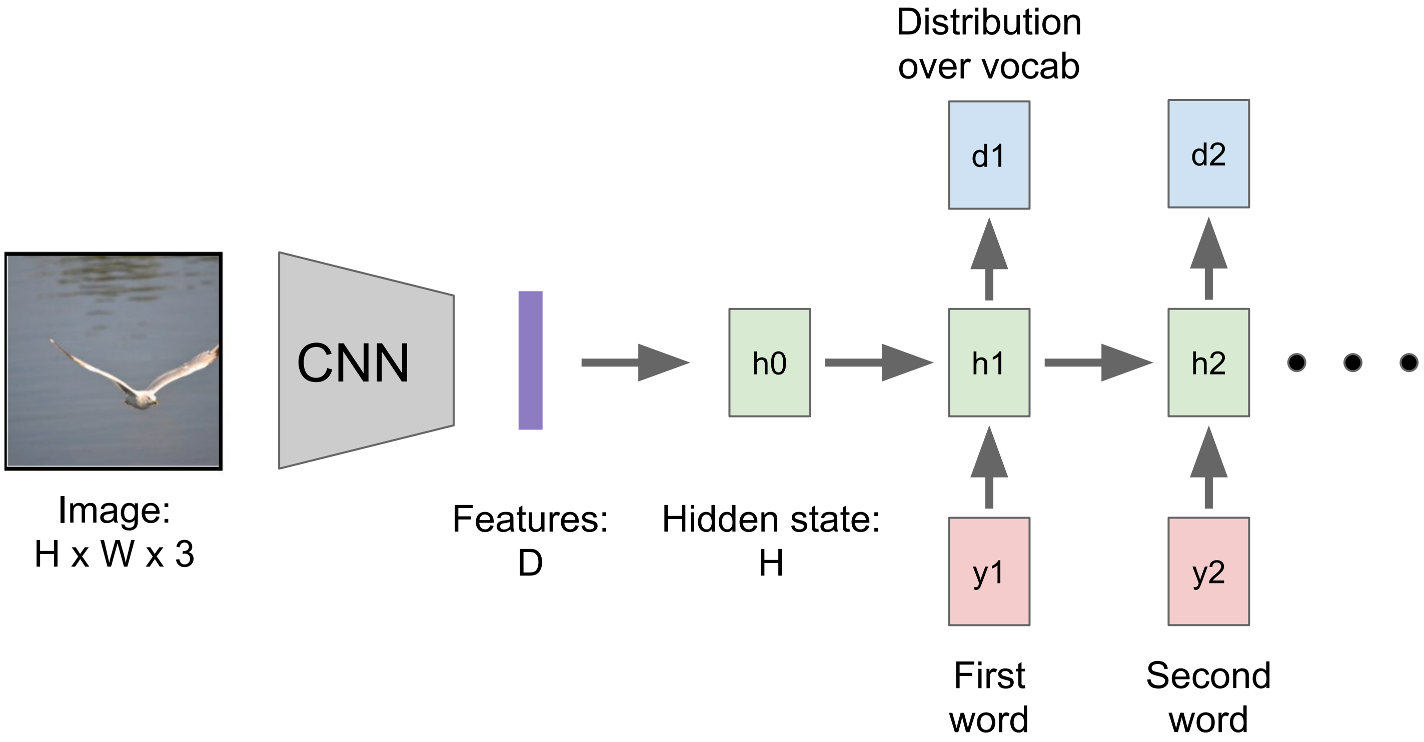
Recall: RNN for Captioning



Recall: RNN for Captioning

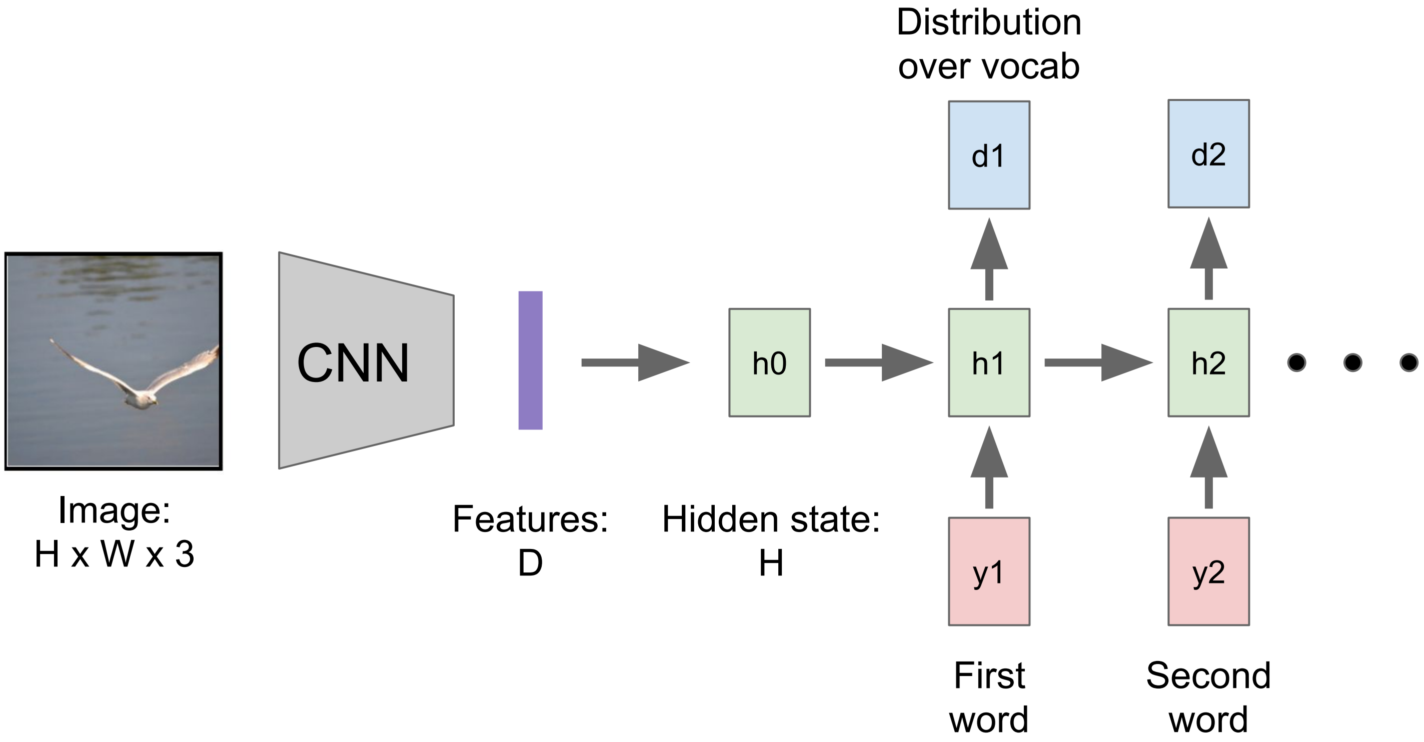


Recall: RNN for Captioning

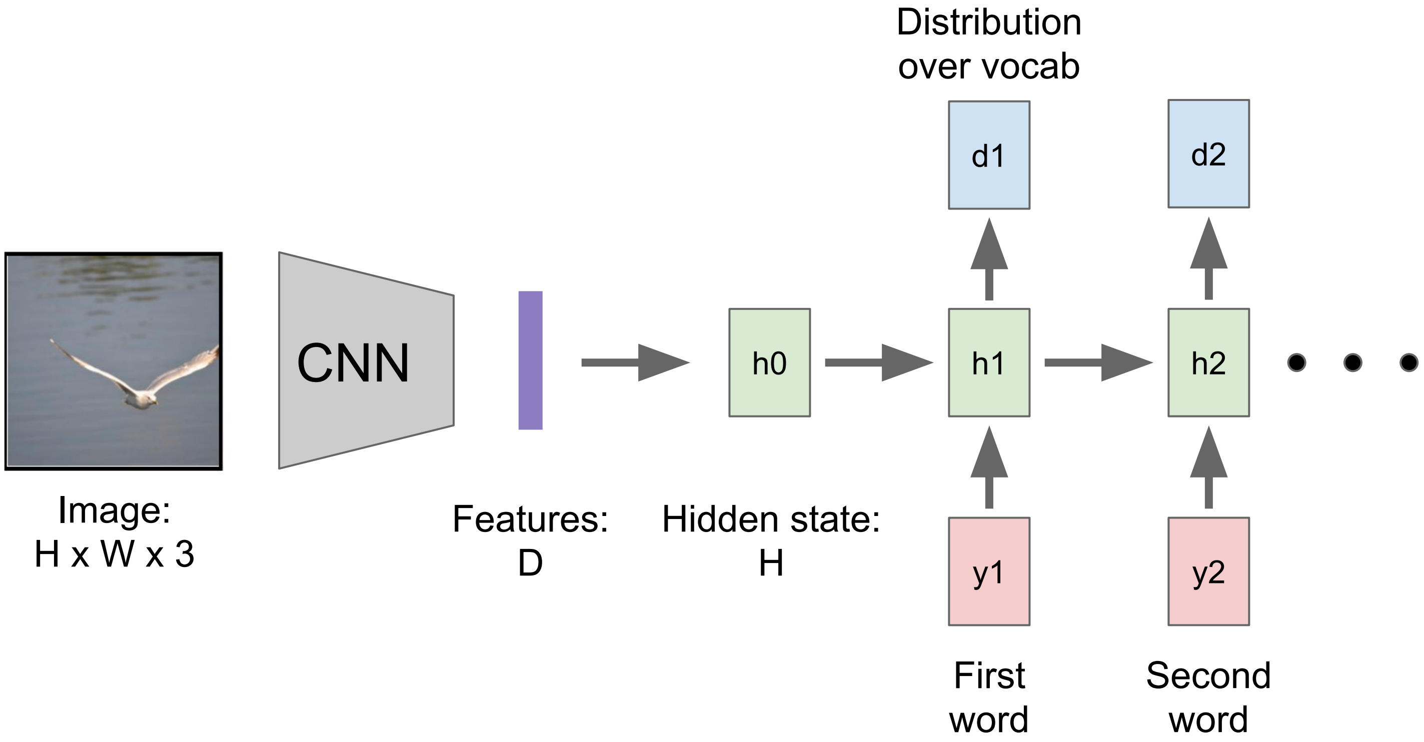


Recall: RNN for Captioning

RNN only looks at whole image, once



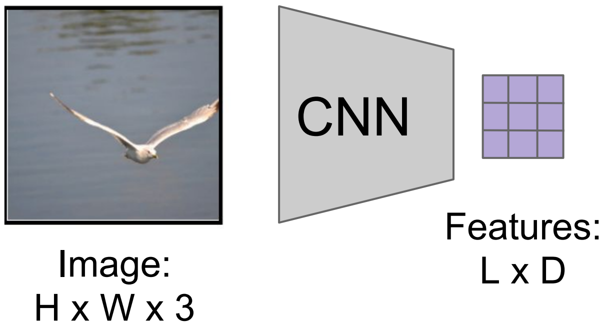
Recall: RNN for Captioning



RNN only looks at whole image, once

What if the RNN looks at different parts of the image at each timestep?

Soft Attention for Captioning



Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Soft Attention for Captioning

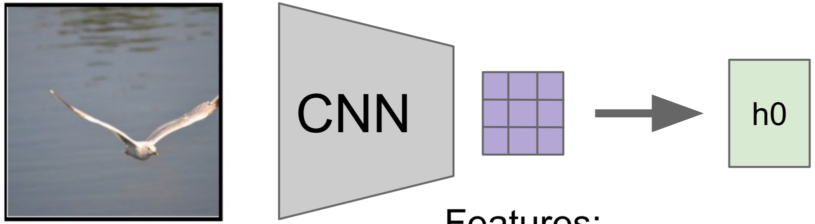
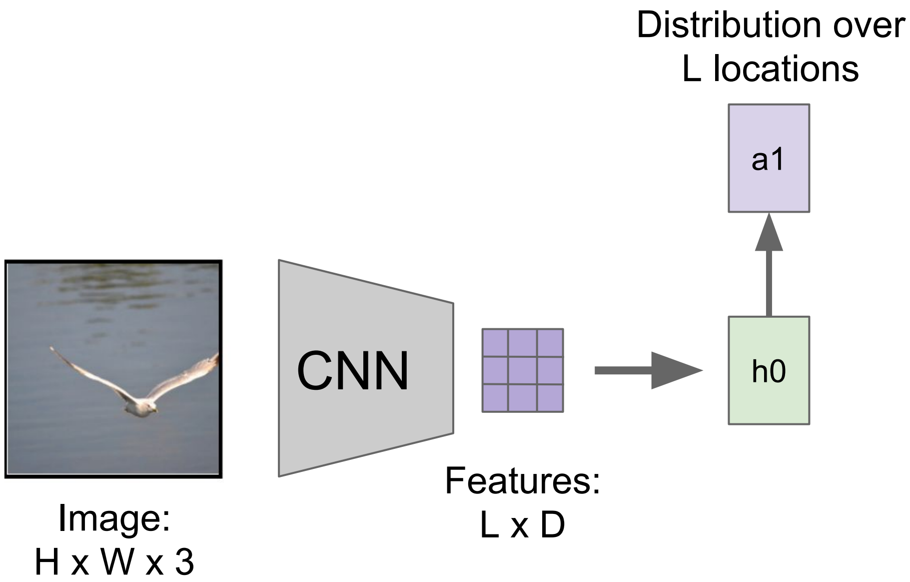


Image:
H x W x 3

Features:
L x D

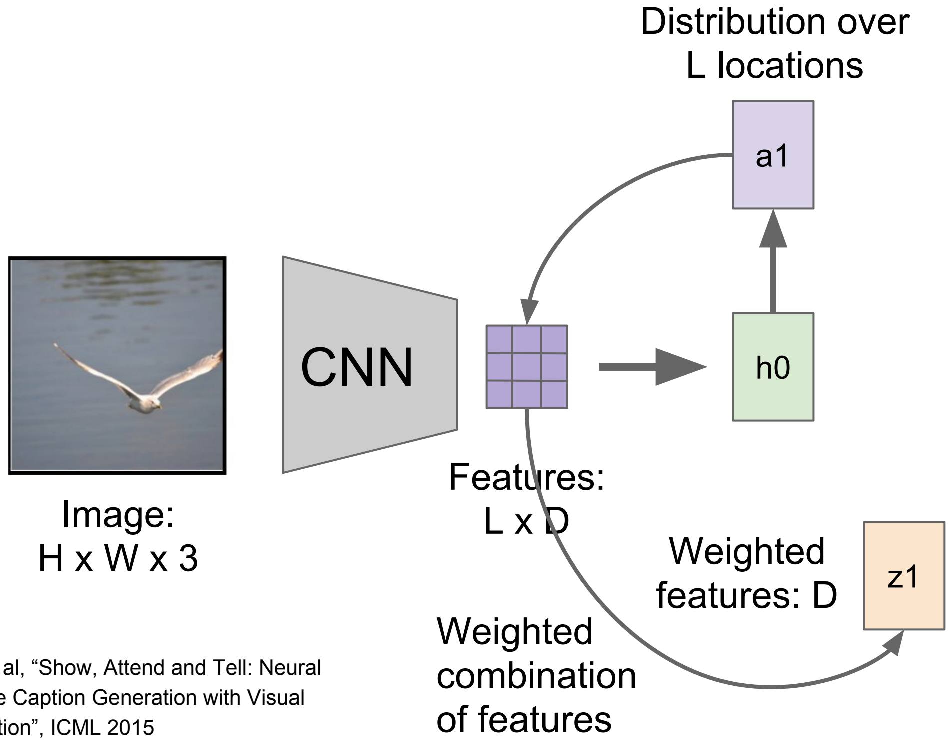
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Soft Attention for Captioning



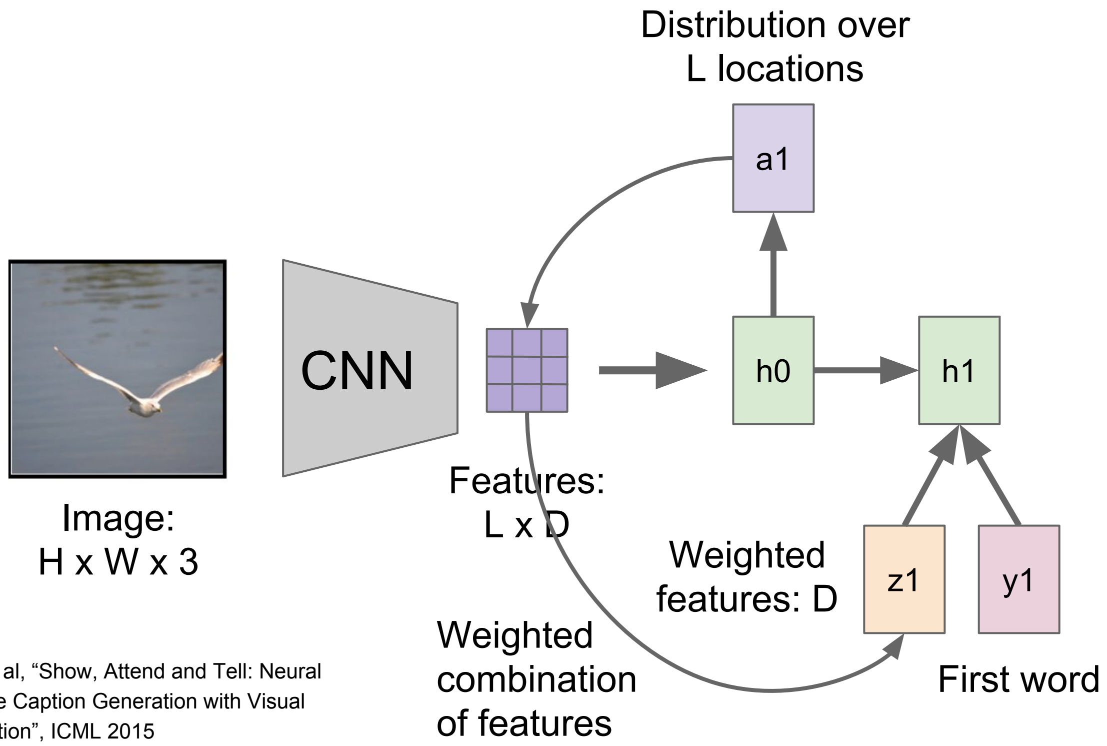
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Soft Attention for Captioning



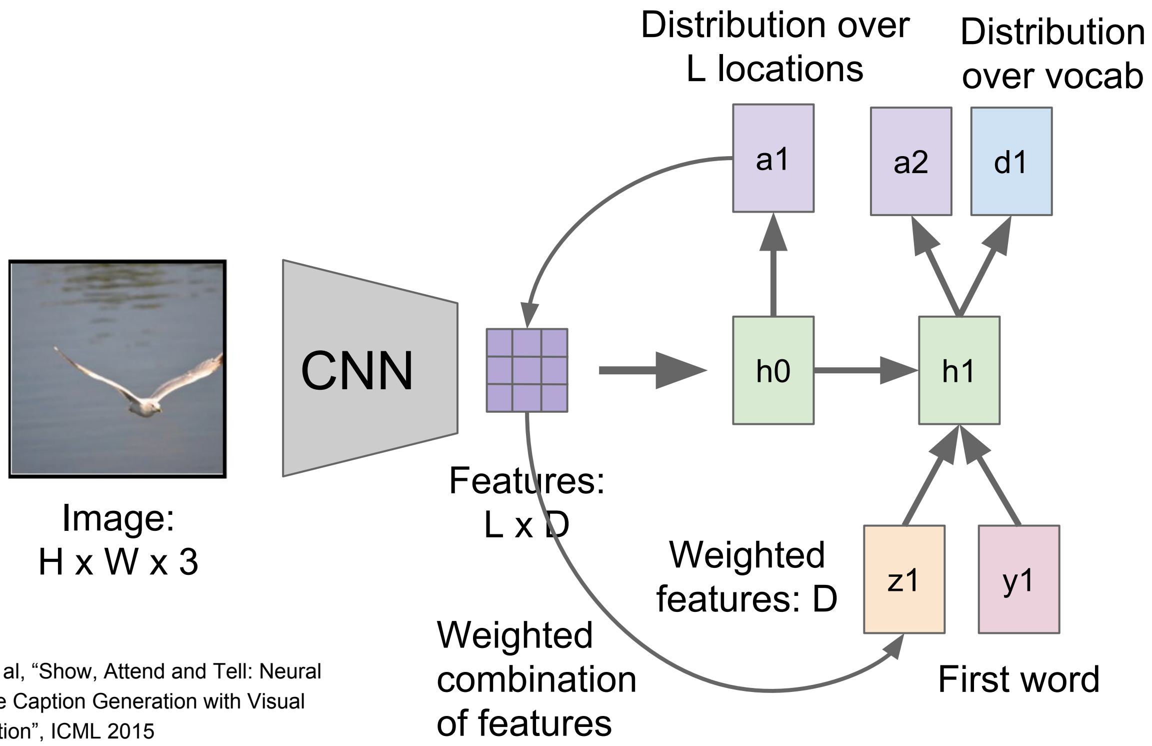
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Soft Attention for Captioning



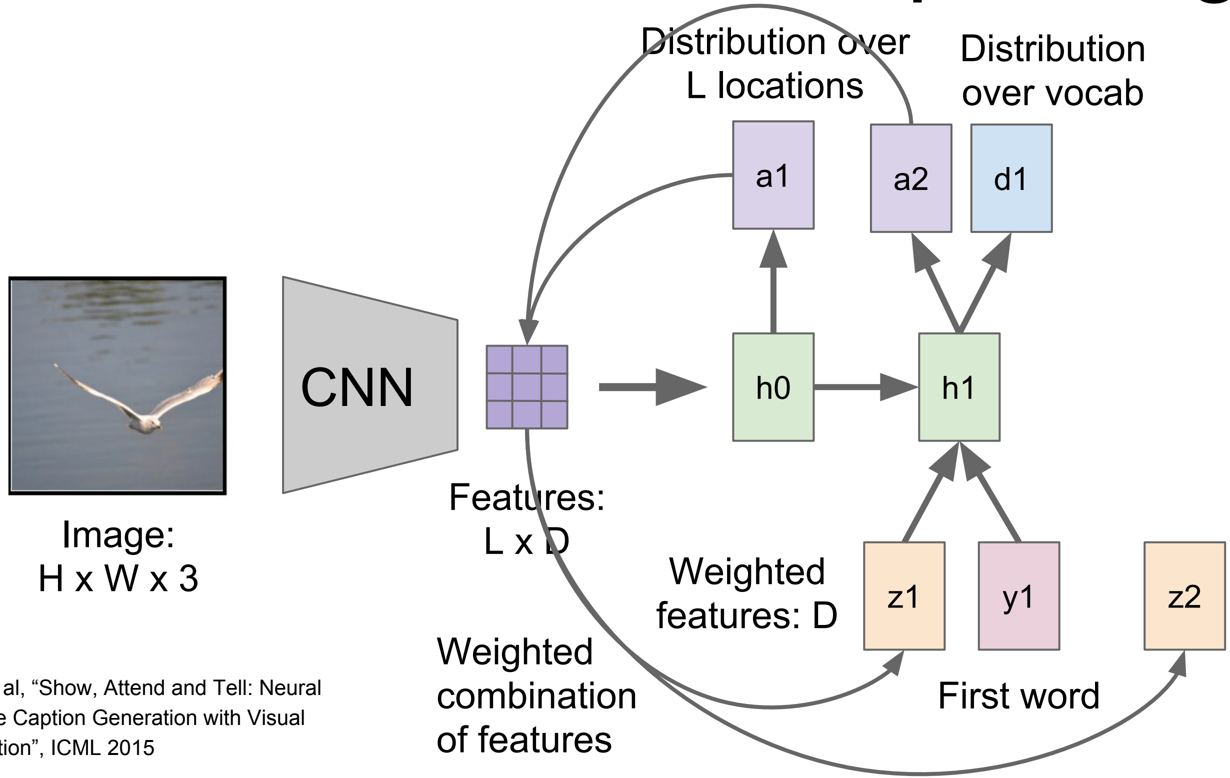
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Soft Attention for Captioning



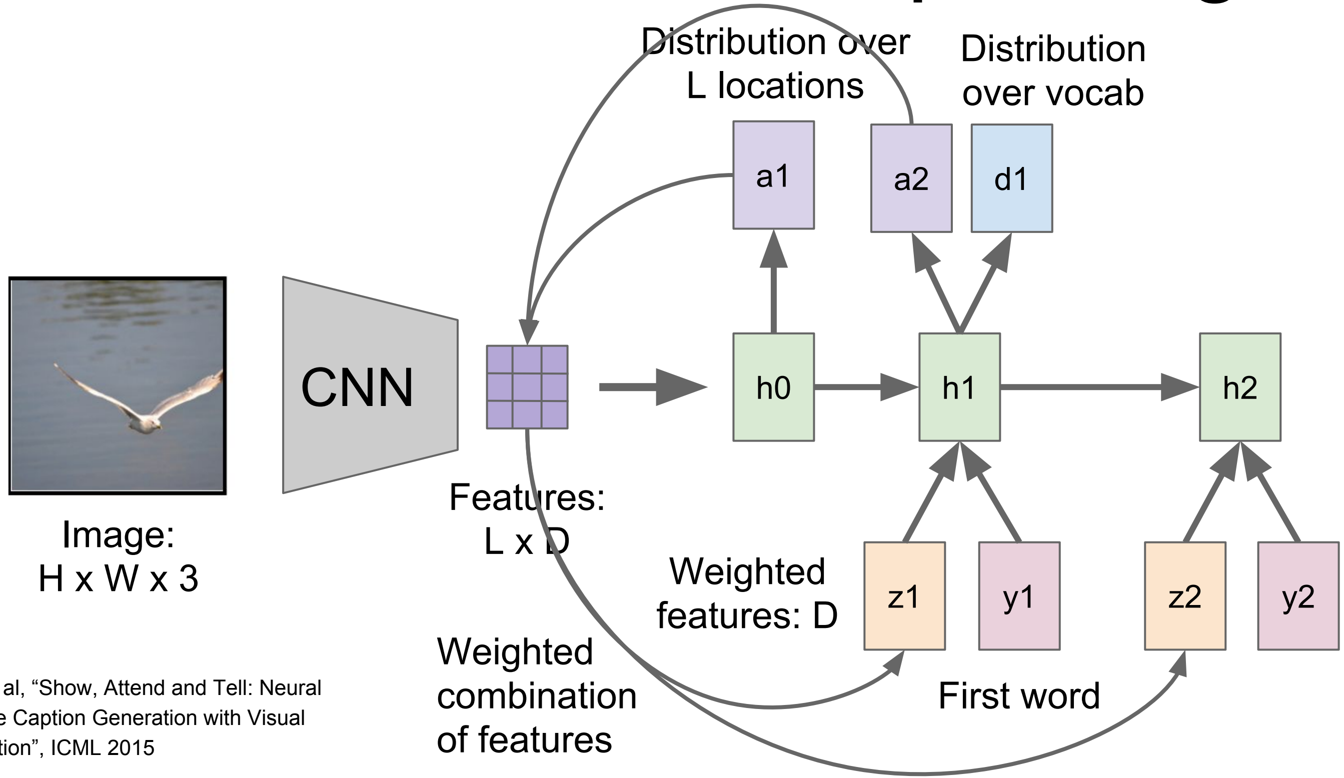
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Soft Attention for Captioning



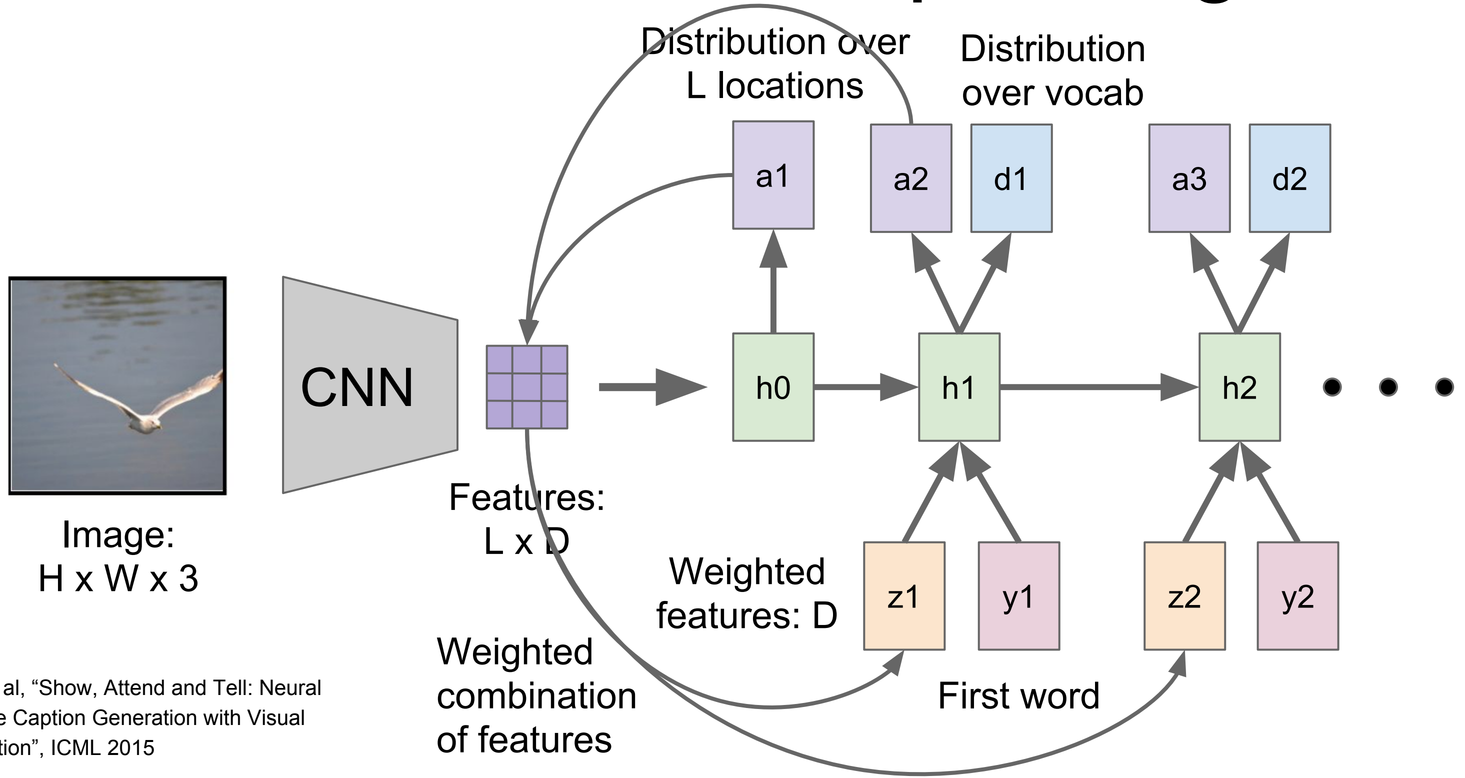
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Soft Attention for Captioning



Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

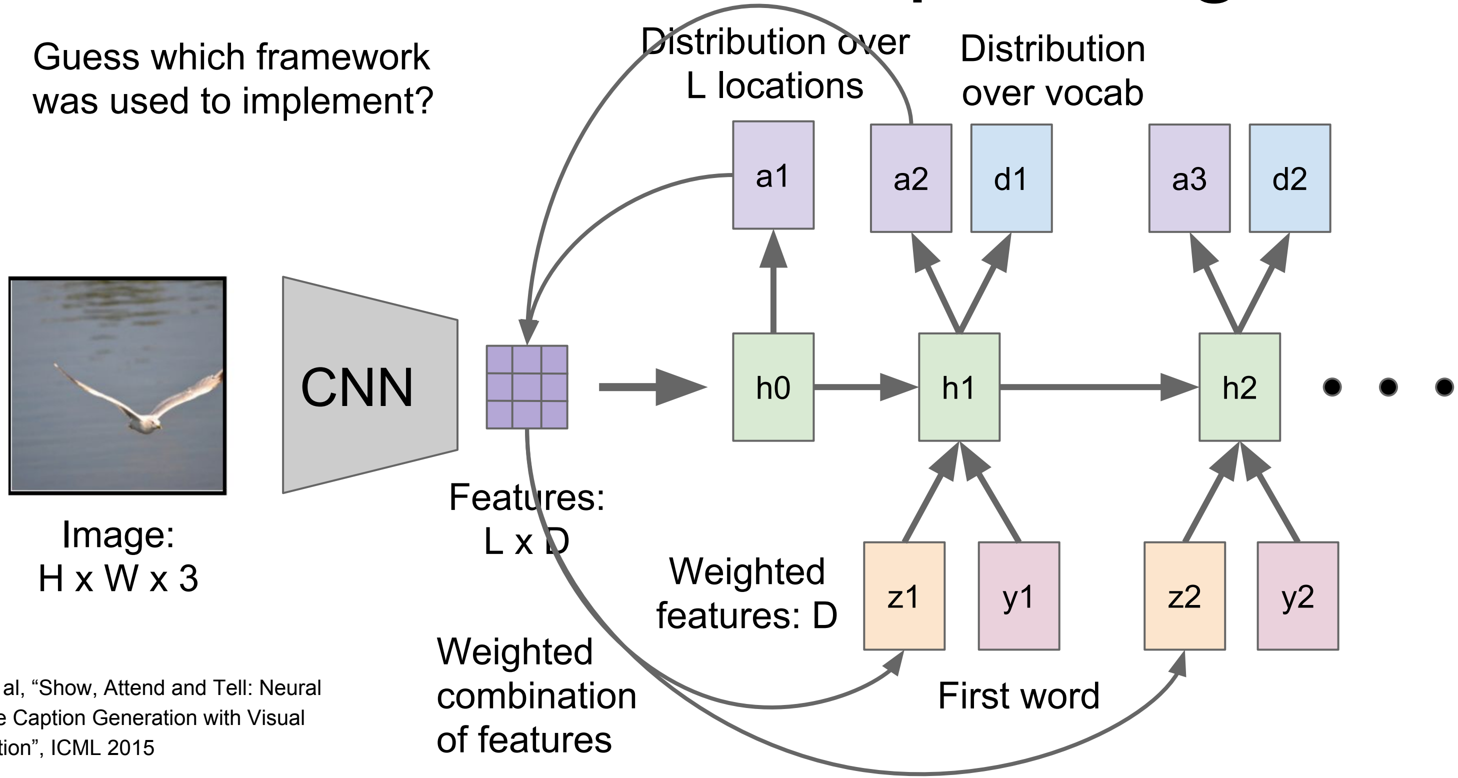
Soft Attention for Captioning



Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Soft Attention for Captioning

Guess which framework was used to implement?



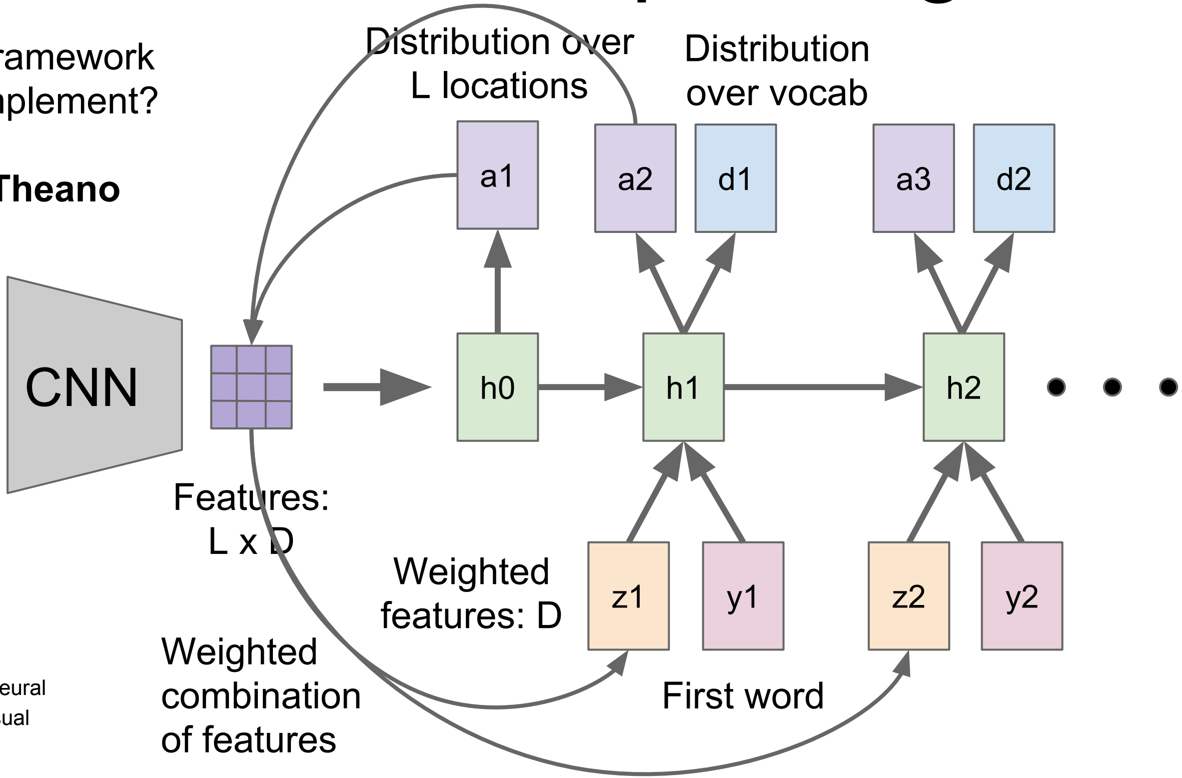
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Soft Attention for Captioning

Guess which framework was used to implement?

Crazy RNN = **Theano**

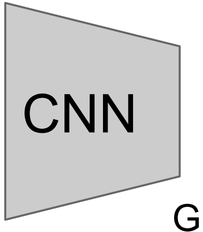
Image:
H x W x 3



Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Soft vs Hard Attention

Image:
H x W x 3



a	b
c	d

Grid of features
(Each D-
dimensional)

From
RNN:

p_a	p_b
p_c	p_d

Distribution over
grid locations
$p_a + p_b + p_c + p_c = 1$

Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Soft vs Hard Attention

Image:
H x W x 3

CNN

a	b
c	d

Grid of features
(Each D-dimensional)

Context vector z
(D-dimensional)

From RNN:

p_a	p_b
p_c	p_d

Distribution over grid locations
$p_a + p_b + p_c + p_c = 1$

Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Soft vs Hard Attention

Image:
H x W x 3

CNN

a	b
c	d

Grid of features
(Each D-dimensional)

From RNN:

p_a	p_b
p_c	p_d

Distribution over grid locations
$p_a + p_b + p_c + p_c = 1$

Context vector z
(D-dimensional)

Soft attention:
Summarize ALL locations
$z = p_a a + p_b b + p_c c + p_d d$

Derivative dz/dp is nice!
Train with gradient descent

Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Soft vs Hard Attention

Image:
H x W x 3

CNN

a	b
c	d

Grid of features
(Each D-dimensional)

From RNN:

p_a	p_b
p_c	p_d

Distribution over grid locations
$p_a + p_b + p_c + p_c = 1$

Context vector z
(D-dimensional)

Soft attention:
Summarize ALL locations
$z = p_a a + p_b b + p_c c + p_d d$

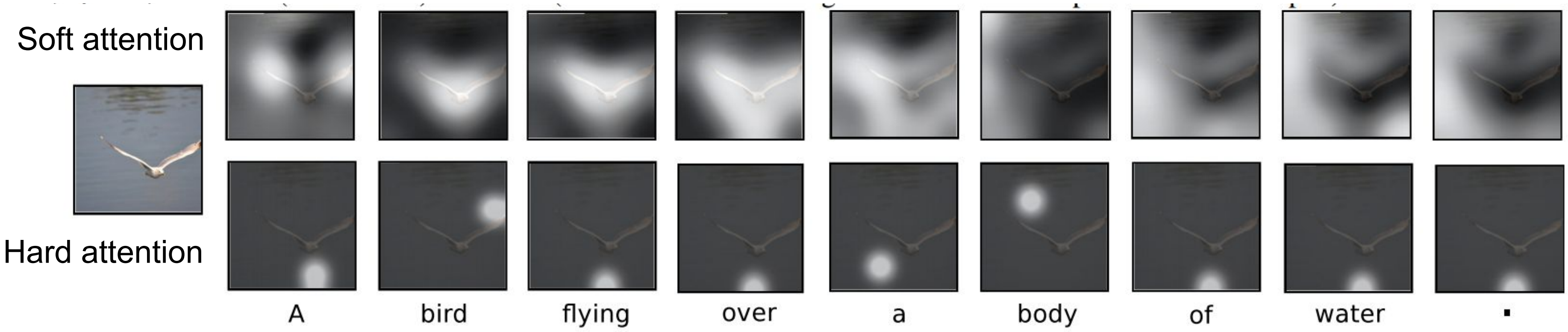
Derivative dz/dp is nice!
Train with gradient descent

Hard attention:
Sample ONE location according to p, z = that vector

With argmax, dz/dp is zero almost everywhere …
Can't use gradient descent; need reinforcement learning

Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Soft Attention for Captioning



Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Soft Attention for Captioning

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

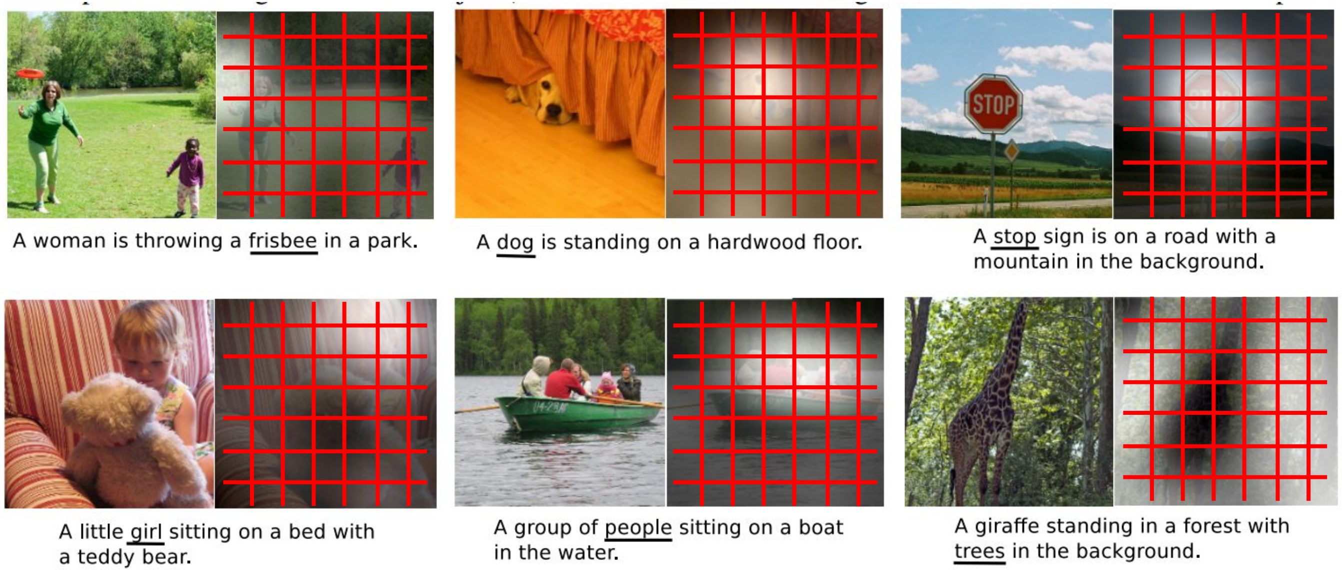
A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Soft Attention for Captioning

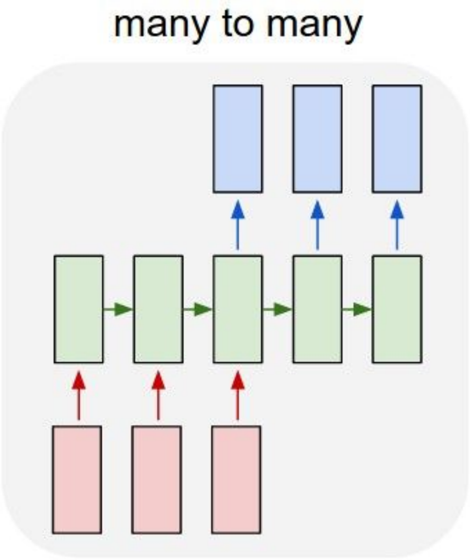
Attention constrained to fixed grid! We'll come back to this ….



Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Soft Attention for Translation

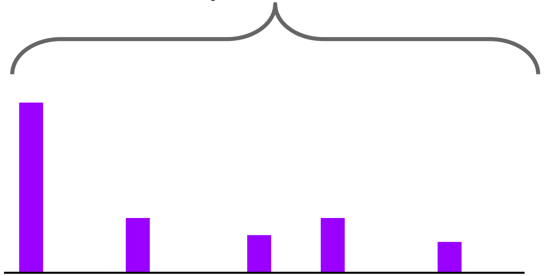
“Mi gato es el mejor” -> “My cat is the best”



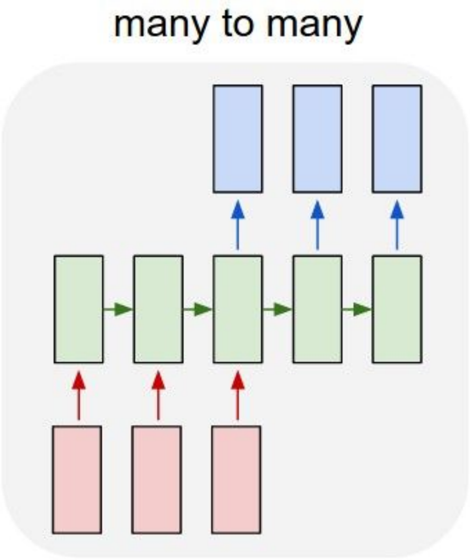
Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015

Soft Attention for Translation

Distribution over
input words



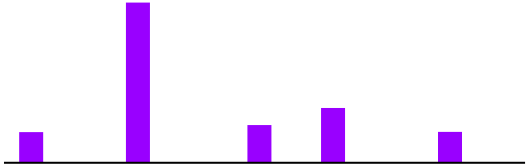
"Mi gato es el mejor" -> "My cat is the best"



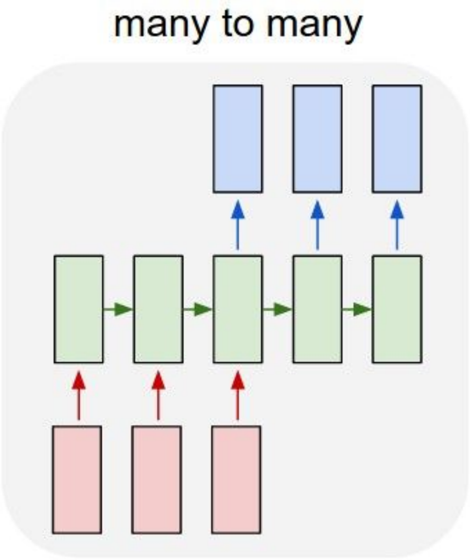
Bahdanau et al, "Neural Machine Translation by
Jointly Learning to Align and Translate", ICLR 2015

Soft Attention for Translation

Distribution over
input words

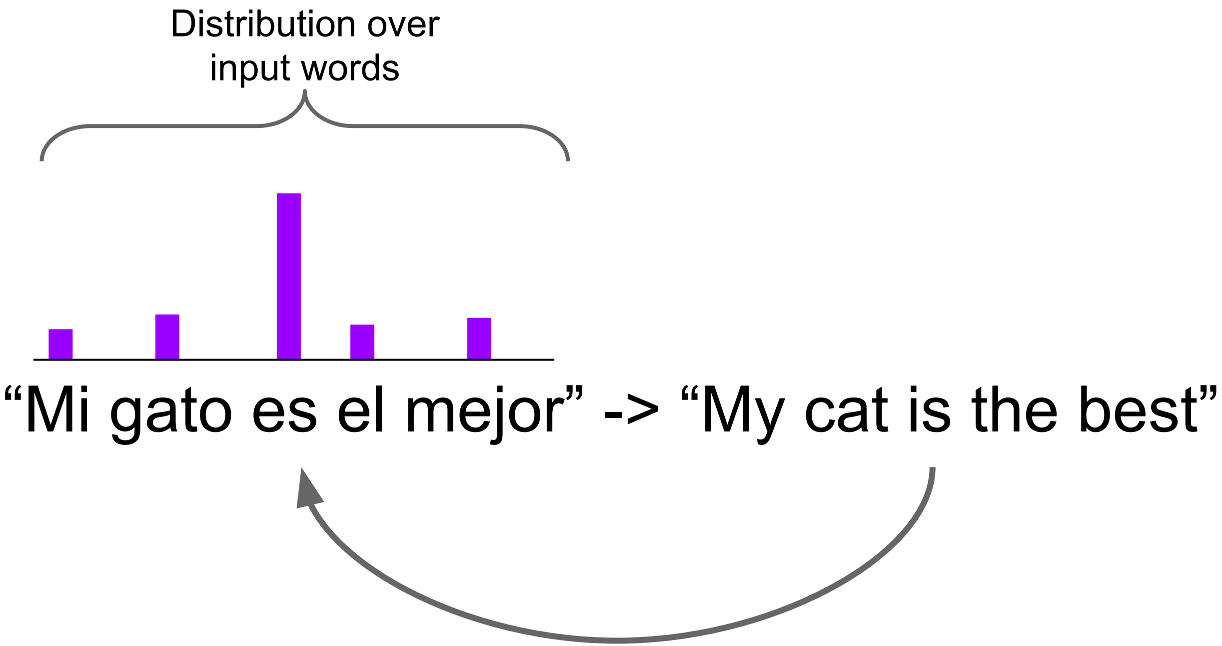


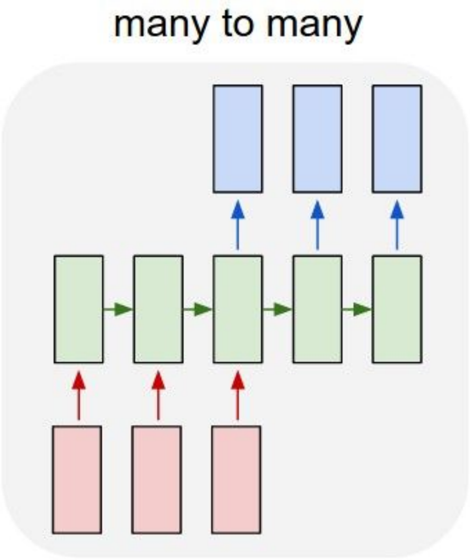
"Mi gato es el mejor" -> "My cat is the best"



Bahdanau et al, "Neural Machine Translation by
Jointly Learning to Align and Translate", ICLR 2015

Soft Attention for Translation





Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015

Soft Attention for Everything!

Machine Translation, attention over input:

- Luong et al, "Effective Approaches to Attention-based Neural Machine Translation," EMNLP 2015

Speech recognition, attention over input sounds:

- Chan et al, "Listen, Attend, and Spell", arXiv 2015
- Chorowski et al, "Attention-based models for Speech Recognition", NIPS 2015

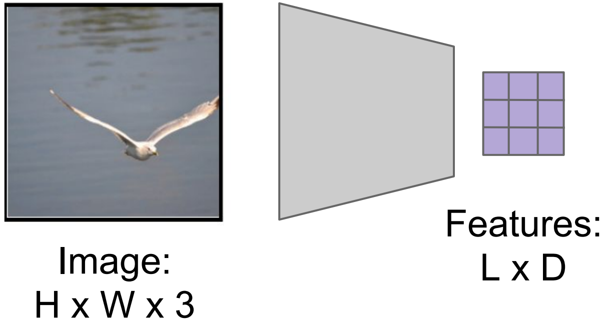
Video captioning, attention over input frames:

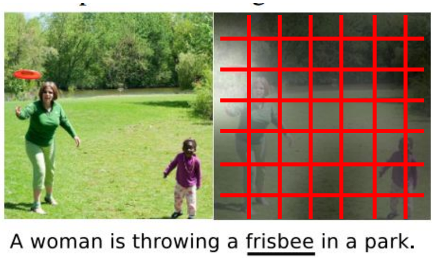
- Yao et al, "Describing Videos by Exploiting Temporal Structure", ICCV 2015

Image, question to answer, attention over image:

- Xu and Saenko, "Ask, Attend and Answer: Exploring Question-Guided Spatial Attention for Visual Question Answering", arXiv 2015
- Zhu et al, "Visual7W: Grounded Question Answering in Images", arXiv 2015

Attending to arbitrary regions?

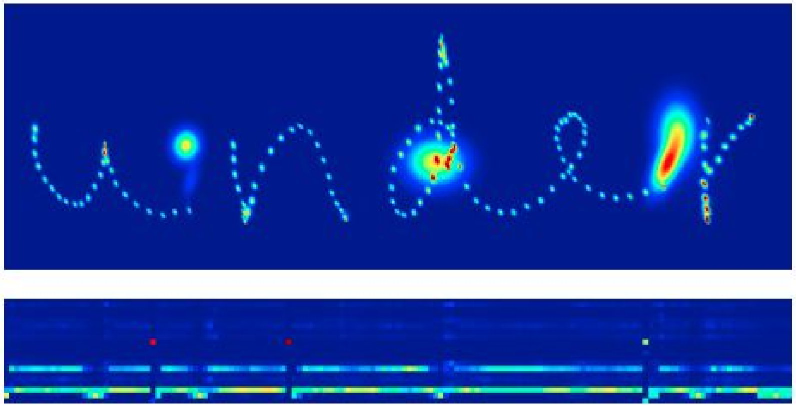




Attention mechanism from Show, Attend, and Tell only lets us softly attend to fixed grid positions … can we do better?

Attending to Arbitrary Regions

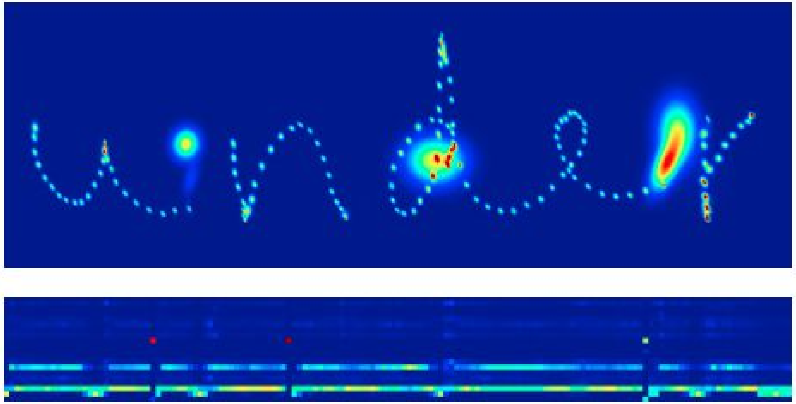
- Read text, generate handwriting using an RNN
- Attend to arbitrary regions of the **output** by predicting params of a mixture model



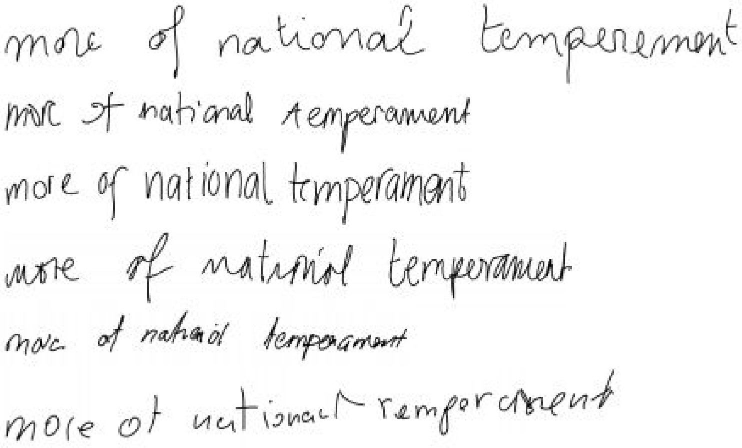
Graves, "Generating Sequences with Recurrent Neural Networks", arXiv 2013

Attending to Arbitrary Regions

- Read text, generate handwriting using an RNN
- Attend to arbitrary regions of the **output** by predicting params of a mixture model



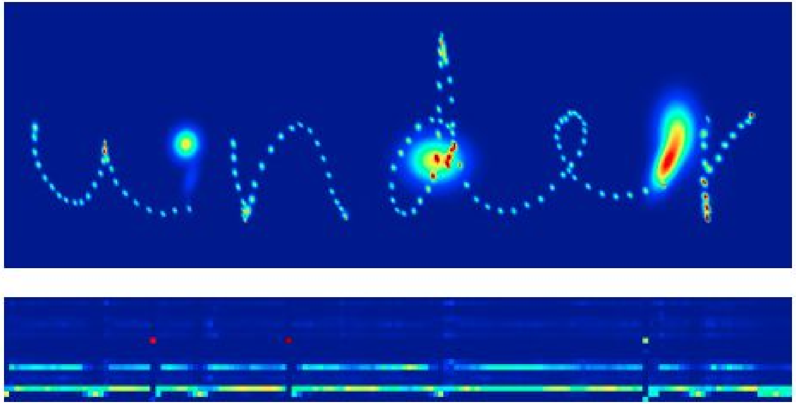
Which are real and which are generated?



Graves, “Generating Sequences with Recurrent Neural Networks”, arXiv 2013

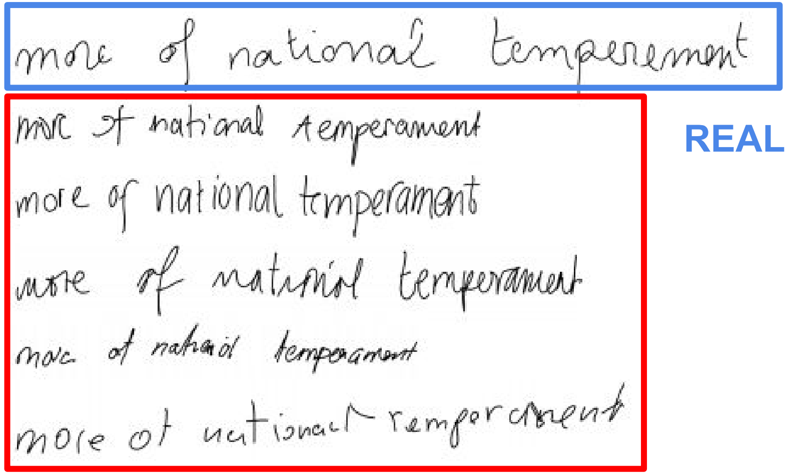
Attending to Arbitrary Regions

- Read text, generate handwriting using an RNN
- Attend to arbitrary regions of the **output** by predicting params of a mixture model



Graves, “Generating Sequences with Recurrent Neural Networks”, arXiv 2013

Which are real and which are generated?

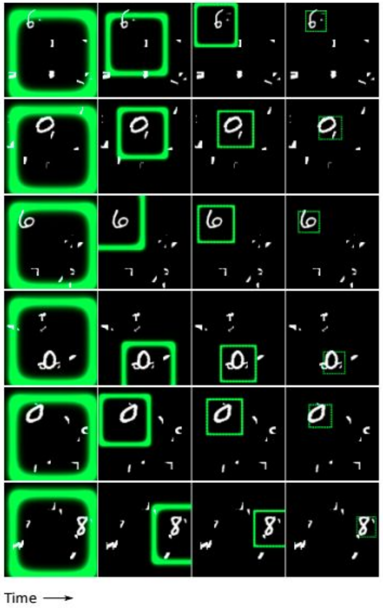


REAL

GENERATED

Attending to Arbitrary Regions: DRAW

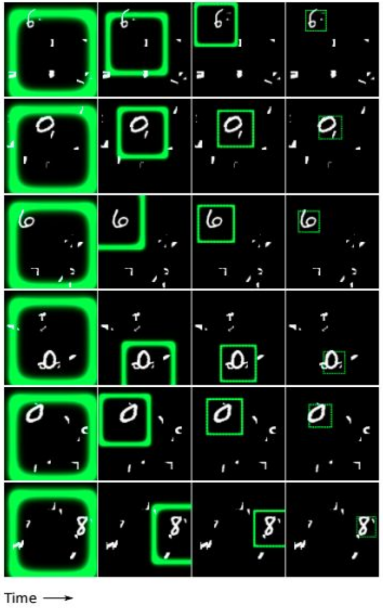
Classify images by attending to arbitrary regions of the *input*



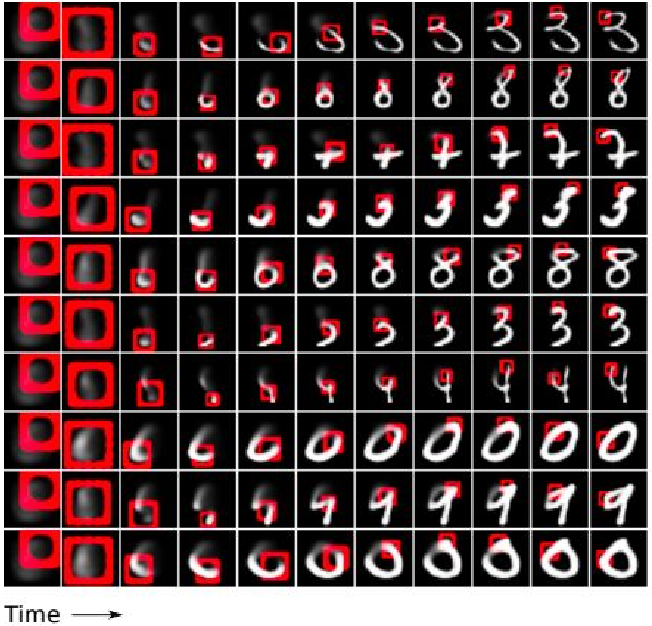
Gregor et al, “DRAW: A Recurrent Neural Network For Image Generation”, ICML 2015

Attending to Arbitrary Regions: DRAW

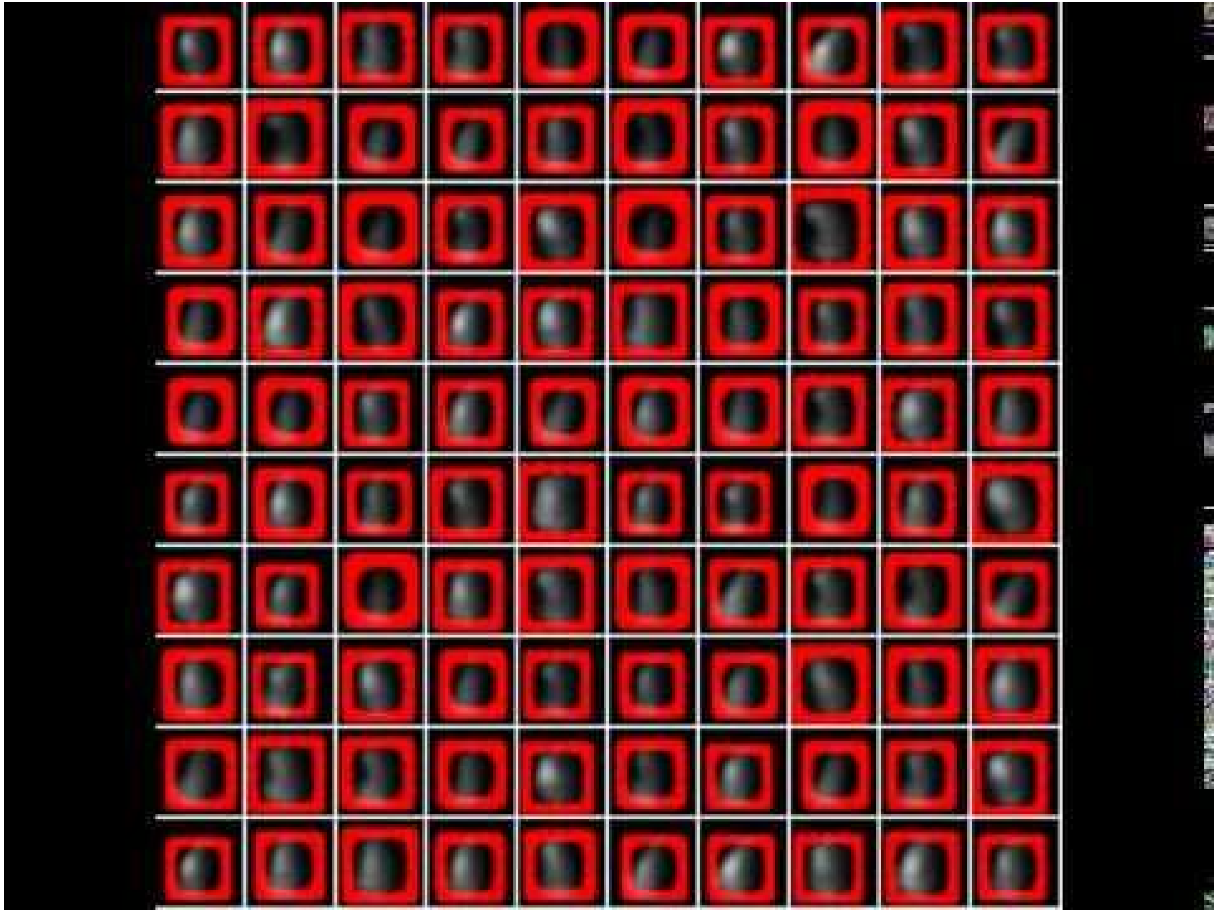
Classify images by attending to arbitrary regions of the *input*



Generate images by attending to arbitrary regions of the *output*



Gregor et al, "DRAW: A Recurrent Neural Network For Image Generation", ICML 2015



Attending to Arbitrary Regions: Spatial Transformer Networks

Attention mechanism similar to DRAW, but easier to explain

Jaderberg et al, “Spatial Transformer Networks”, NIPS 2015

Spatial Transformer Networks

Input image:
H x W x 3

Cropped and
rescaled image:
X x Y x 3

Box Coordinates:
(xc, yc, w, h)

Jaderberg et al, “Spatial Transformer Networks”, NIPS 2015

Spatial Transformer Networks

Input image:
H x W x 3

Can we make this function differentiable?

Cropped and rescaled image:
X x Y x 3

Box Coordinates:
(xc, yc, w, h)

Jaderberg et al, “Spatial Transformer Networks”, NIPS 2015

Spatial Transformer Networks

Input image:
H x W x 3

Box Coordinates:
(xc, yc, w, h)

Can we make this function differentiable?

Cropped and rescaled image:
X x Y x 3

Idea: Function mapping *pixel coordinates* (xt, yt) of output to *pixel coordinates* (xs, ys) of input

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

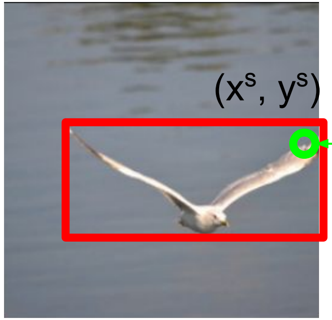
Jaderberg et al, “Spatial Transformer Networks”, NIPS 2015

Spatial Transformer Networks

Idea: Function mapping *pixel coordinates* (xt, yt) of output to *pixel coordinates* (xs, ys) of input

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

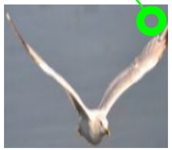
(x^s, y^s)



Input image:
H x W x 3

Can we make this function differentiable?

(x^t, y^t)

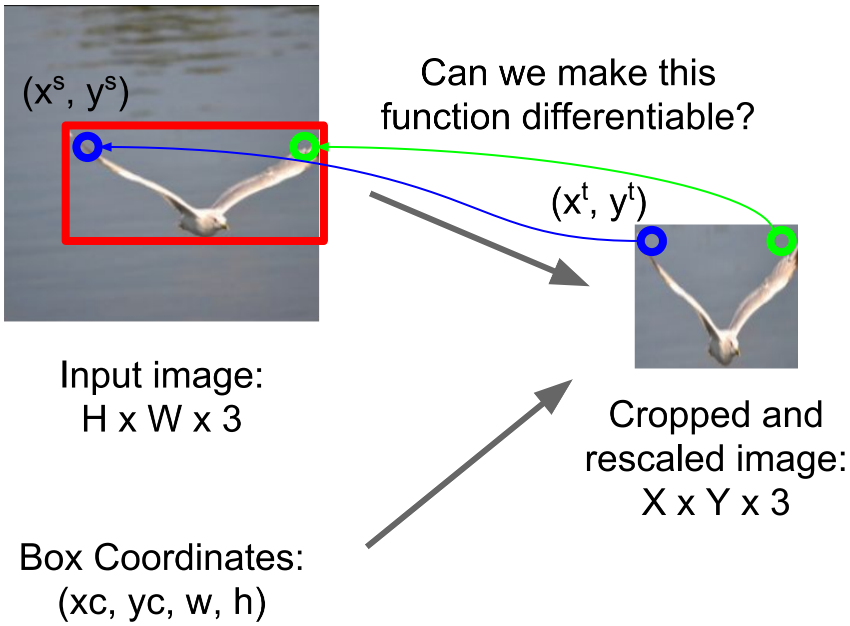


Cropped and rescaled image:
X x Y x 3

Box Coordinates:
(xc, yc, w, h)

Jaderberg et al, “Spatial Transformer Networks”, NIPS 2015

Spatial Transformer Networks

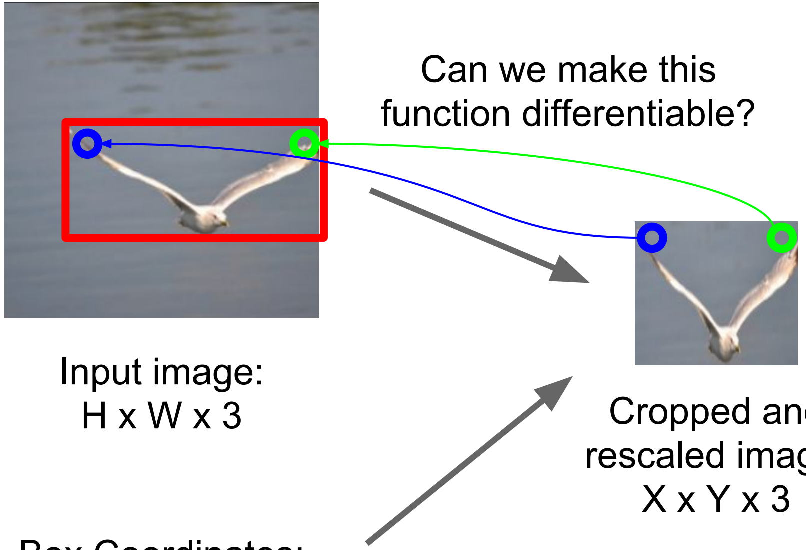


Idea: Function mapping *pixel coordinates* (xt, yt) of output to *pixel coordinates* (xs, ys) of input

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

Jaderberg et al, “Spatial Transformer Networks”, NIPS 2015

Spatial Transformer Networks



Can we make this function differentiable?

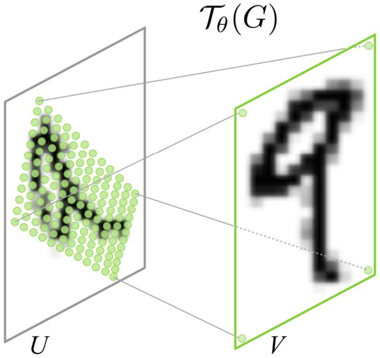
Input image:
H x W x 3

Cropped and rescaled image:
X x Y x 3

Box Coordinates:
(xc, yc, w, h)

Idea: Function mapping *pixel coordinates* (xt, yt) of output to *pixel coordinates* (xs, ys) of input

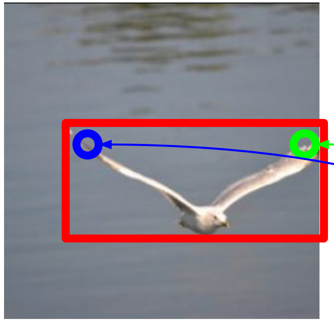
$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$



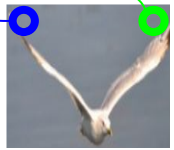
Repeat for all pixels in *output* to get a **sampling grid**

Jaderberg et al, "Spatial Transformer Networks", NIPS 2015

Spatial Transformer Networks



Can we make this function differentiable?



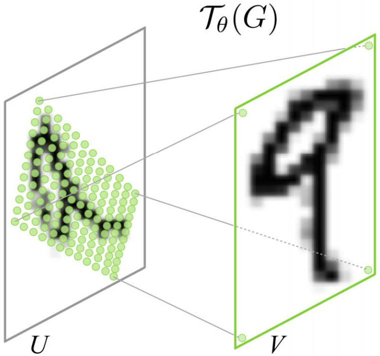
Input image:
H x W x 3

Cropped and rescaled image:
X x Y x 3

Box Coordinates:
(xc, yc, w, h)

Idea: Function mapping *pixel coordinates* (xt, yt) of output to *pixel coordinates* (xs, ys) of input

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

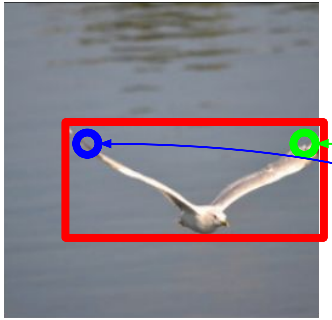


Repeat for all pixels in *output* to get a **sampling grid**

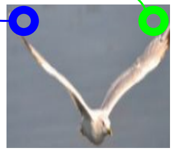
Then use **bilinear interpolation** to compute output

Jaderberg et al, “Spatial Transformer Networks”, NIPS 2015

Spatial Transformer Networks



Can we make this function differentiable?



Input image:
H x W x 3

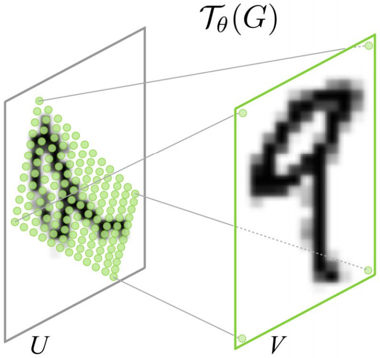
Cropped and rescaled image:
X x Y x 3

Box Coordinates:
(xc, yc, w, h)

Idea: Function mapping *pixel coordinates* (xt, yt) of output to *pixel coordinates* (xs, ys) of input

Network attends to input by predicting θ

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$



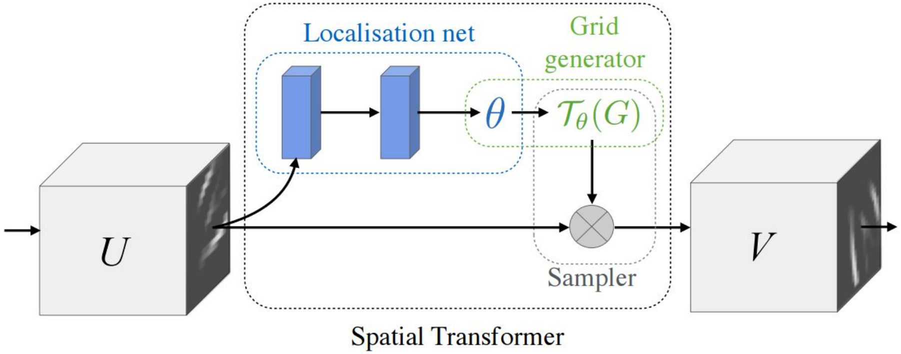
Repeat for all pixels in *output* to get a **sampling grid**

Then use **bilinear interpolation** to compute output

Jaderberg et al, "Spatial Transformer Networks", NIPS 2015

Spatial Transformer Networks

Input:
Full image

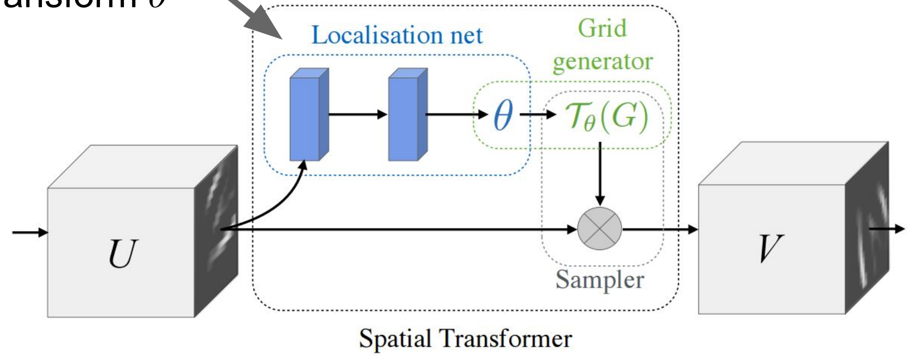


Output: Region of
interest from input

Spatial Transformer Networks

A small
Localization network
predicts transform θ

Input:
Full image



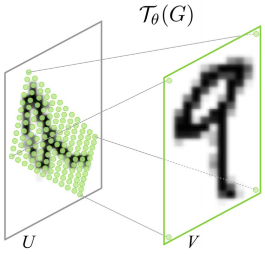
Output: Region of interest from input

Spatial Transformer Networks

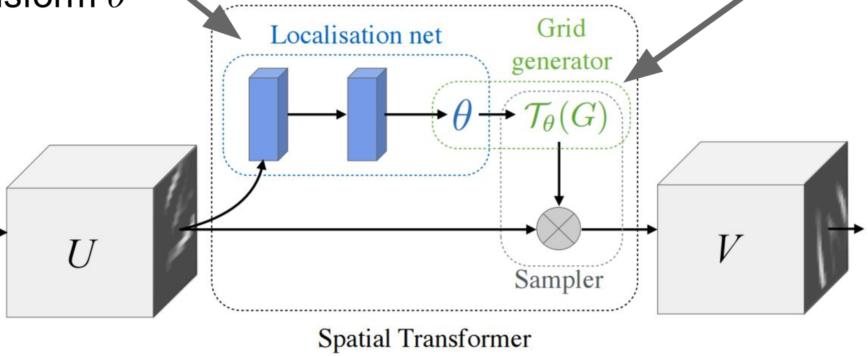
A small
Localization network
predicts transform θ

Grid generator uses θ to
compute sampling grid

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$



Input:
Full image



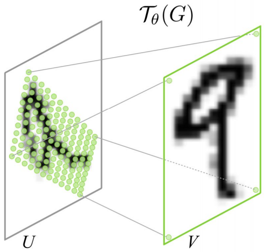
Output: Region of
interest from input

Spatial Transformer Networks

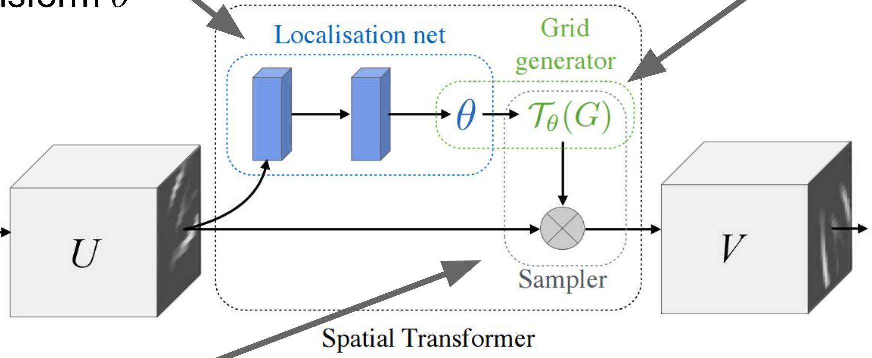
A small **Localization network** predicts transform θ

Grid generator uses θ to compute sampling grid

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$



Input: Full image



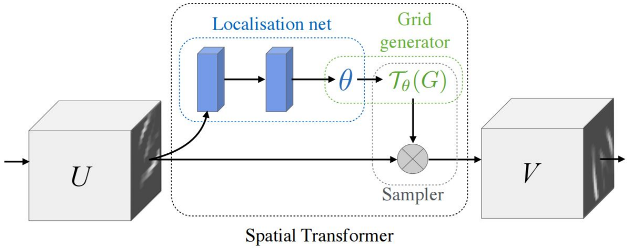
Output: Region of interest from input

Sampler uses bilinear interpolation to produce output

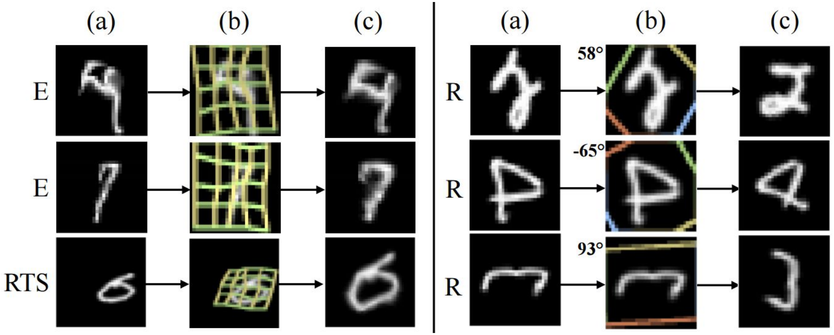
$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

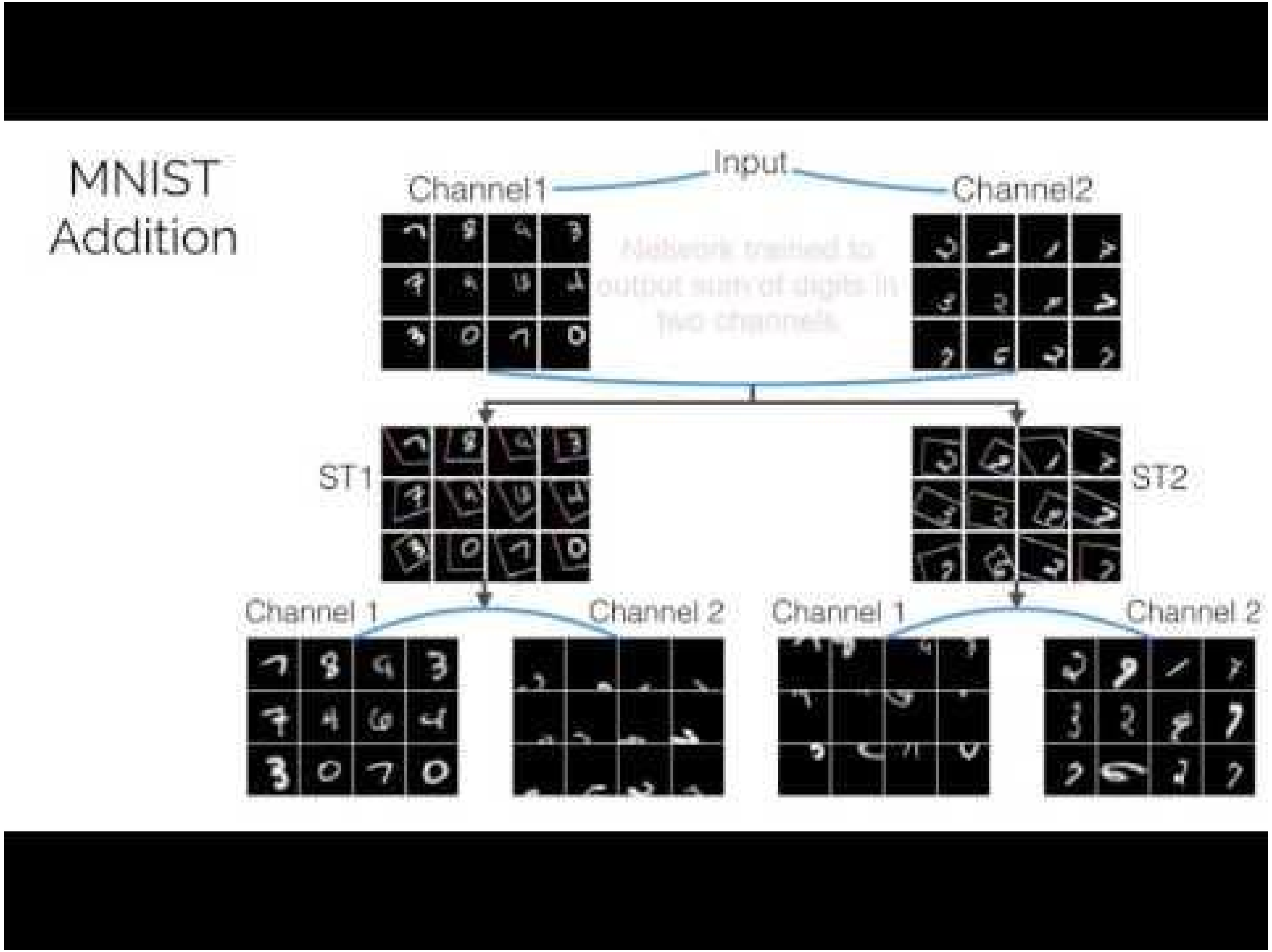
Spatial Transformer Networks

Differentiable "attention / transformation" module



Insert spatial transformers into a classification network and it learns to attend and transform the input




MNIST
Addition
Input
Channel1
Channel2
ST1
ST2
Channel 1
Channel 2
Channel 1
Channel 2

Attention Recap

- Soft attention:
    - Easy to implement: produce distribution over input locations, reweight features and feed as input
    - Attend to arbitrary input locations using spatial transformer networks
- Hard attention:
    - Attend to a single input location
    - Can’t use gradient descent!
    - Need **reinforcement learning!**