

# 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)

## TensorFlow

```
1 import tensorflow as tf
2 import numpy as np
3
4 N, D, H, C = 64, 1000, 100, 10
5
6 x = tf.placeholder(tf.float32, shape=[None, D])
7 y = tf.placeholder(tf.float32, shape=[None, C])
8
9 w1 = tf.Variable(tf.random.randn(D, H).astype(np.float32))
10 w2 = tf.Variable(tf.random.randn(H, C).astype(np.float32))
11
12 a = tf.matmul(x, w1)
13 a_relu = tf.nn.relu(a)
14 scores = tf.matmul(a_relu, w2)
15 probs = tf.nn.softmax(scores)
16 loss = -tf.reduce_sum(y * tf.log(probs))
17
18 learning_rate = 1e-2
19 train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
20
21 xx = np.random.randn(N, D).astype(np.float32)
22 yy = np.zeros((N, C)).astype(np.float32)
23 yy[np.arange(N), np.random.randint(C, size=N)] = 1
24
25 with tf.Session() as sess:
26     sess.run(tf.initialize_all_variables())
27
28     for t in xrange(100):
29         _, loss_value = sess.run([train_step, loss],
30                                feed_dict={x: xx, y: yy})
31
32     print loss_value
```

## Torch

```
1 require 'torch'
2 require 'nn'
3 require 'optim'
4
5 -- Batch size, input dim, hidden dim, num classes
6 local N, D, H, C = 100, 1000, 100, 10
7
8 -- Build a one-layer ReLU network
9 local net = nn.Sequential()
10 net:add(nn.Linear(D, H))
11 net: add(nn.ReLU())
12 net: add(nn.Linear(H, C))
13
14 -- Collect all weights and gradients in a single Tensor
15 local weights, grad_weights = net:getParameters()
16
17 -- Loss functions are called "criterions"
18 local crit = nn.CrossEntropyCriterion() -- Softmax loss
19
20 -- Callback to interface with optim methods
21 local function f(w)
22     assert(w == weights)
23
24     -- Generate some random input data
25     local x = torch.randn(N, D)
26     local y = torch.Tensor(N):random(C)
27
28     -- Forward pass: Compute scores and loss
29     local scores = net:forward(x)
30     local loss = crit:forward(scores, y)
31
32     -- Backward pass: compute gradients
33     grad_weights:zero()
34     local dscores = crit:backward(scores, y)
35     local dx = net:backward(x, dscores)
36
37     return loss, grad_weights
38 end
39
40 -- Make a step using Adam
41 local state = {learningRate=1e-3}
42 optim.adam(f, weights, state)
43
```

## Theano

```
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

```
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(shapes=(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=0.9)

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
```

## Keras

```
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)

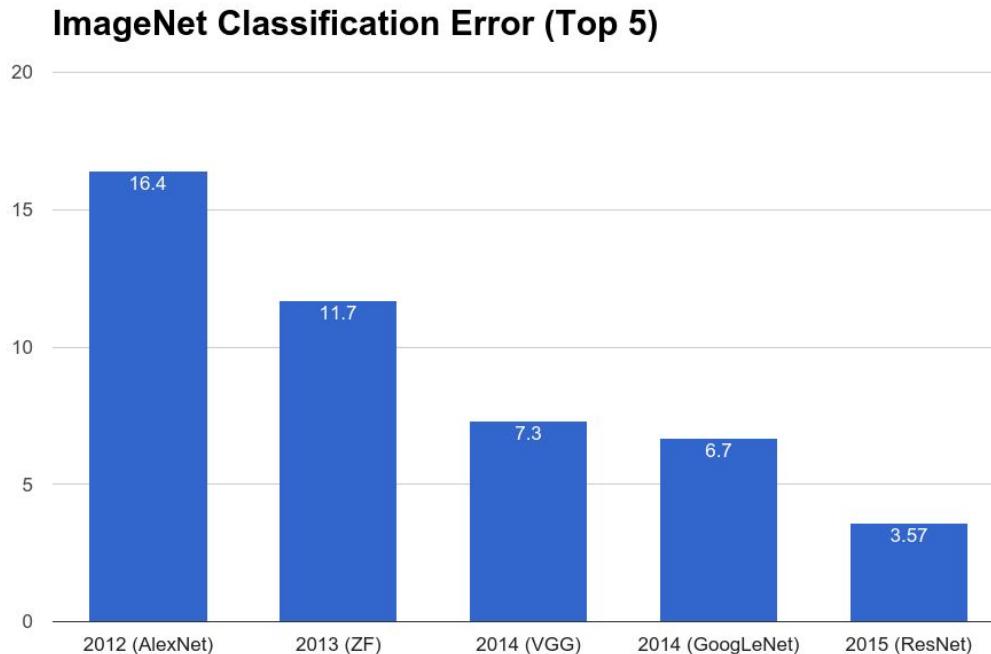
N, N_batch = 1000, 32
X = np.random.randn(N, D)
y = np.random.randint(C, size=N)
y = np_utils.to_categorical(y)

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

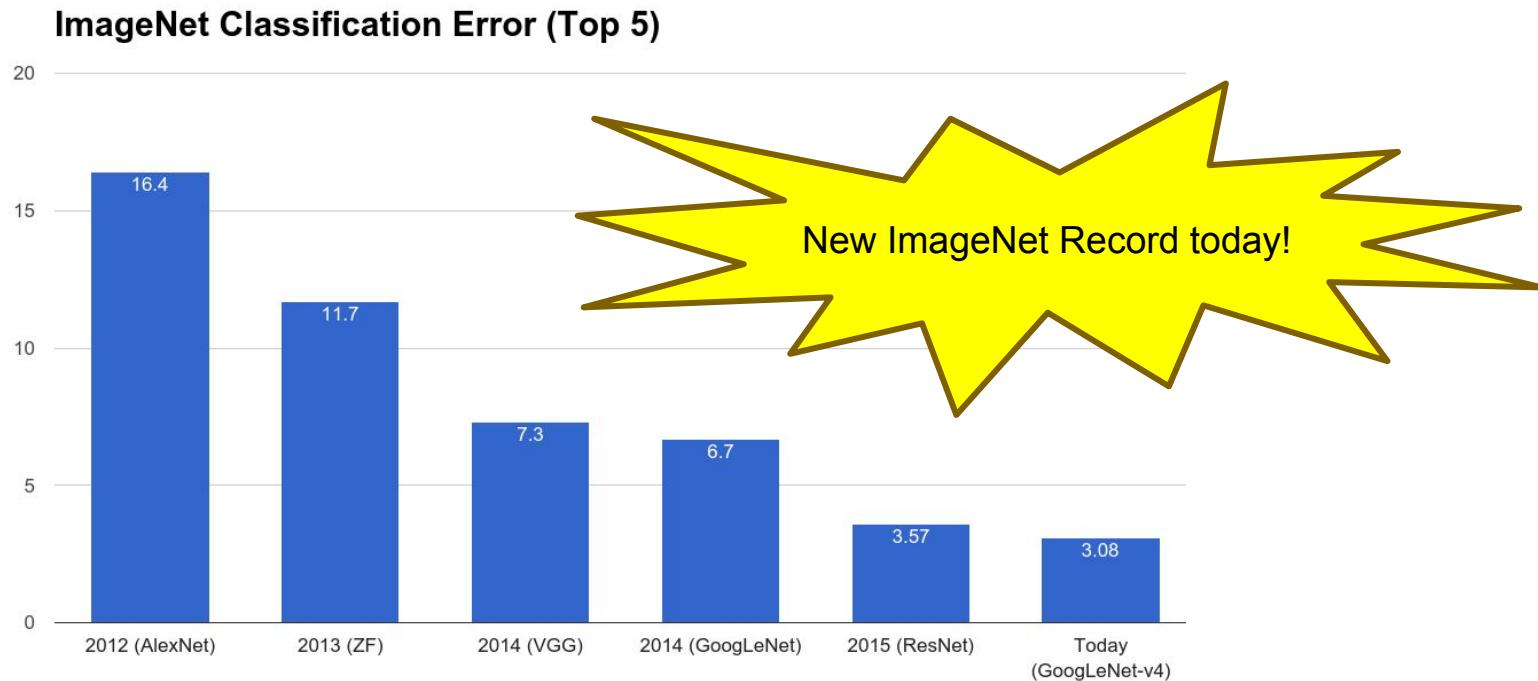
# Today

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

# But first....

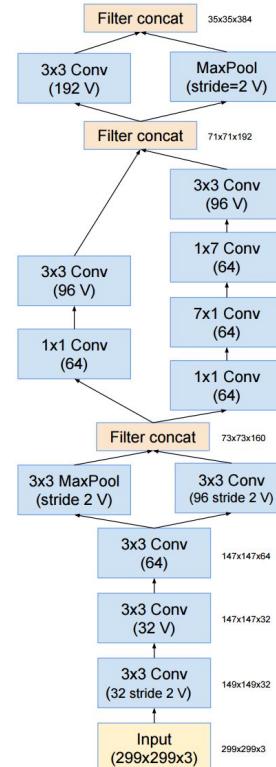
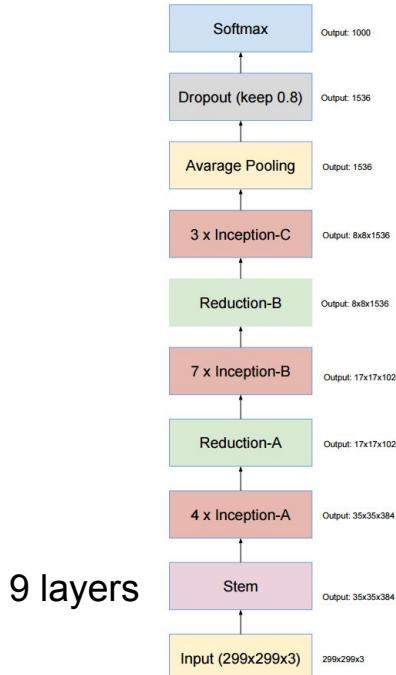


# But first....



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

# Inception-v4



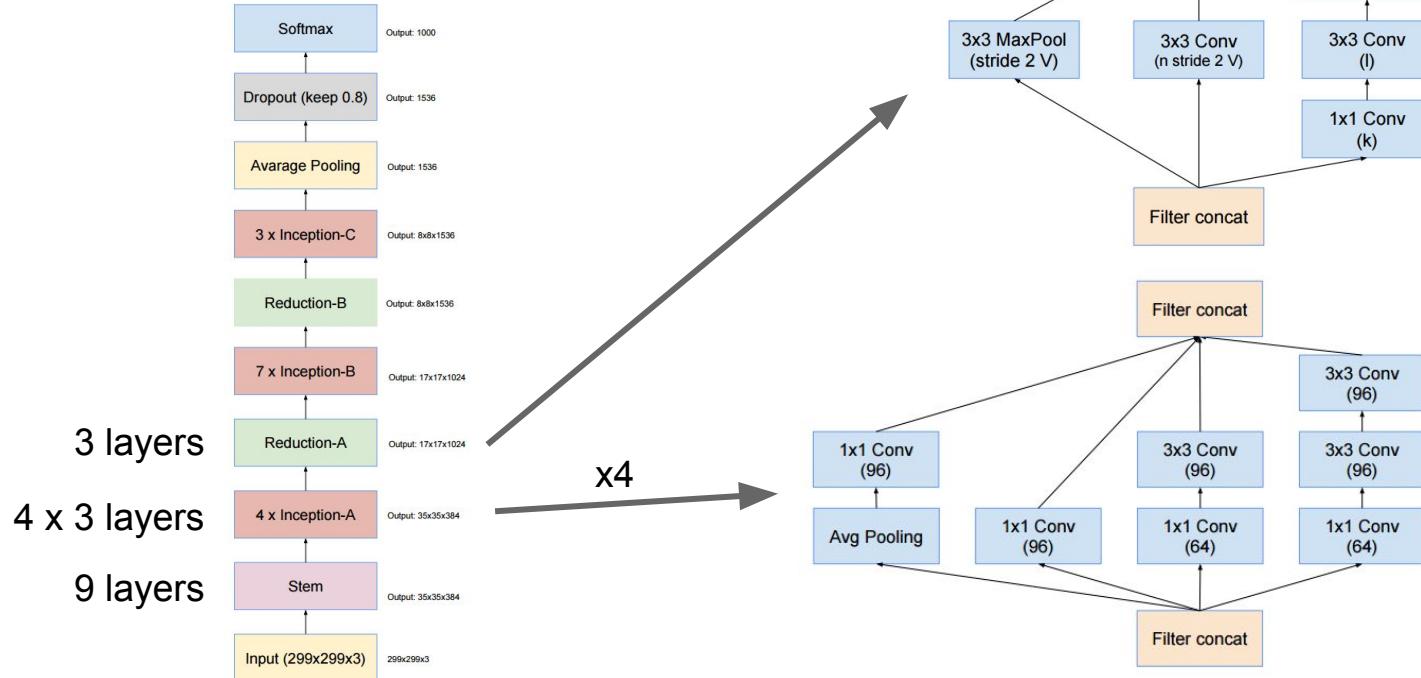
$1 \times 7, 7 \times 1$  filters

Strided convolution  
AND max pooling

$V =$  Valid convolutions  
(no padding)

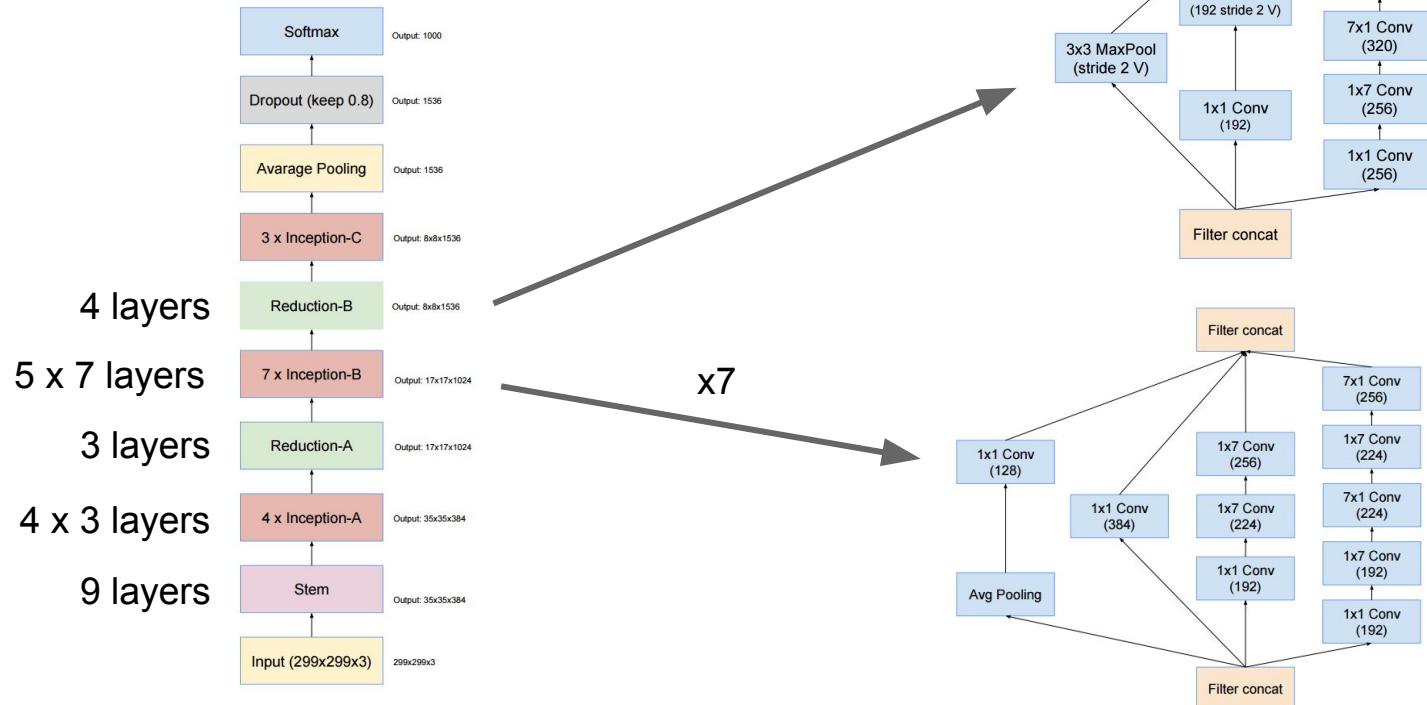
Szegedy et al, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, arXiv 2016

# Inception-v4



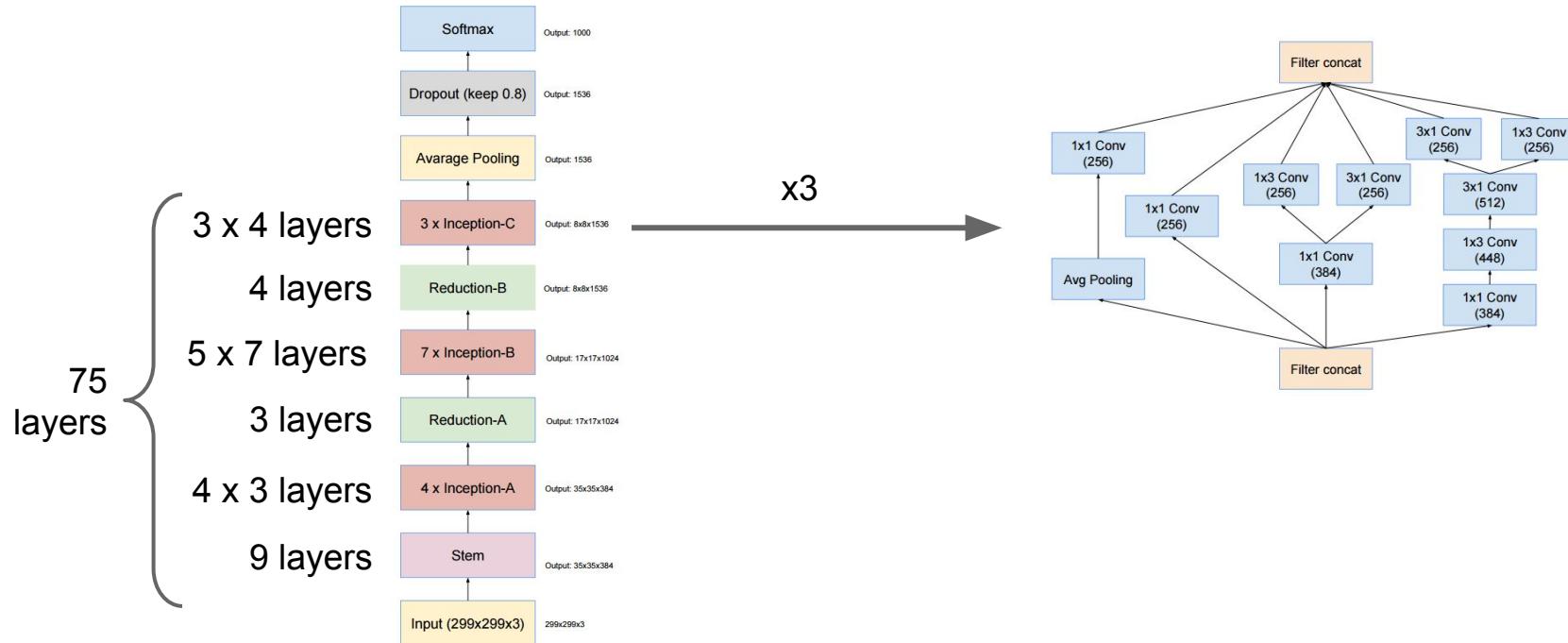
Szegedy et al, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, arXiv 2016

# Inception-v4



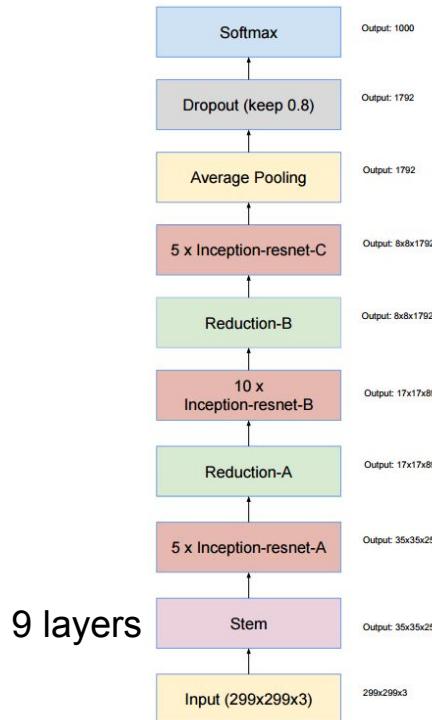
Szegedy et al, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, arXiv 2016

# Inception-v4

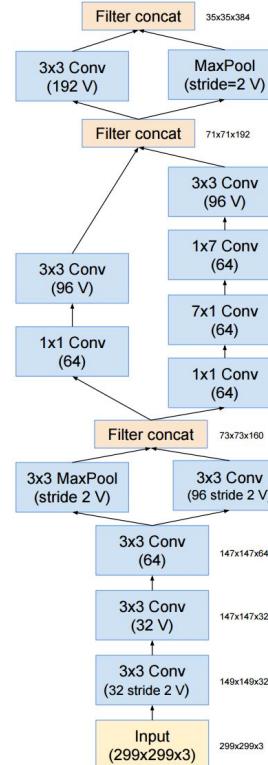


Szegedy et al, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, arXiv 2016

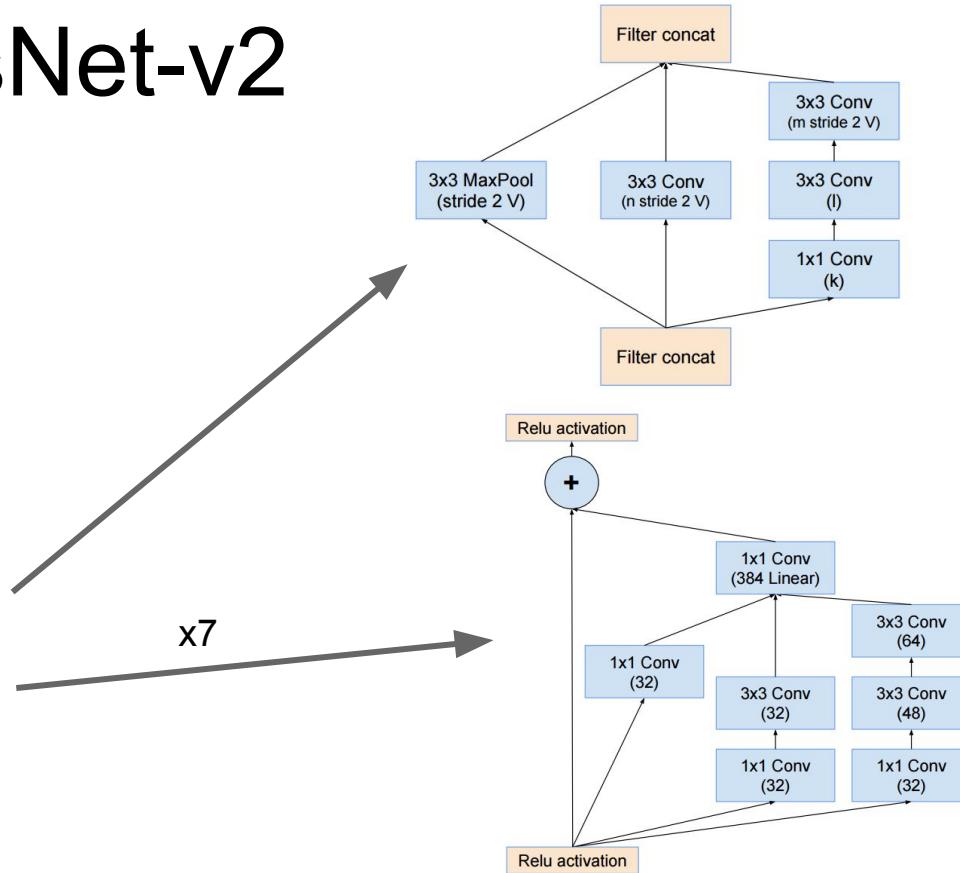
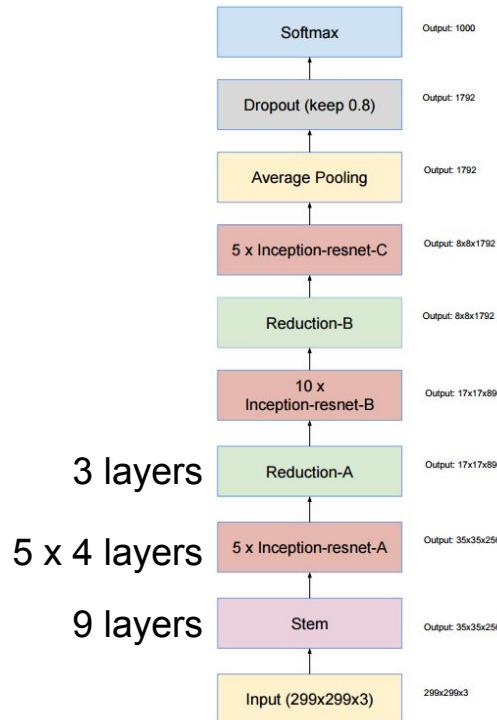
# Inception-ResNet-v2



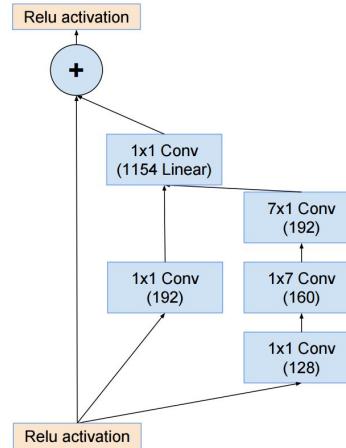
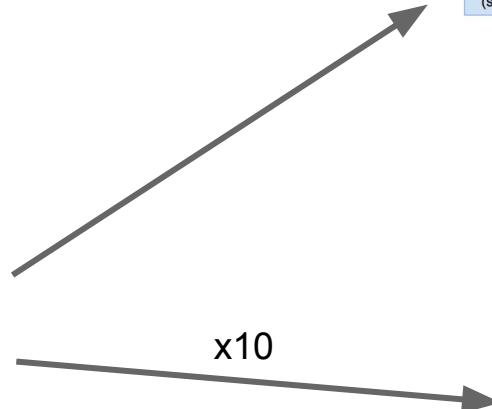
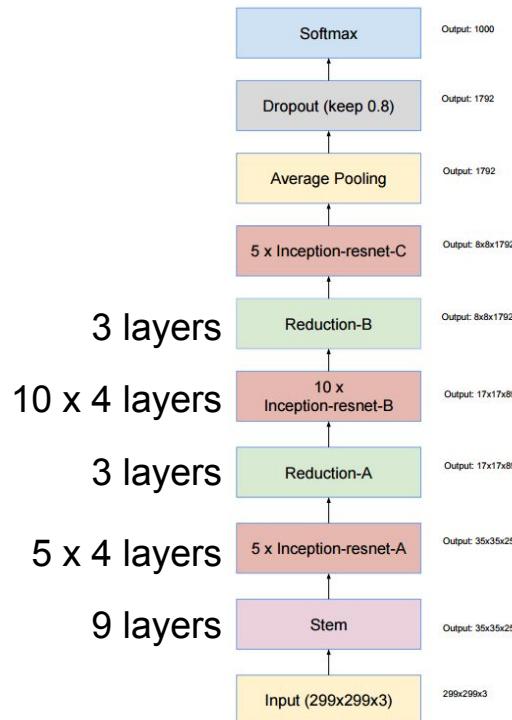
9 layers



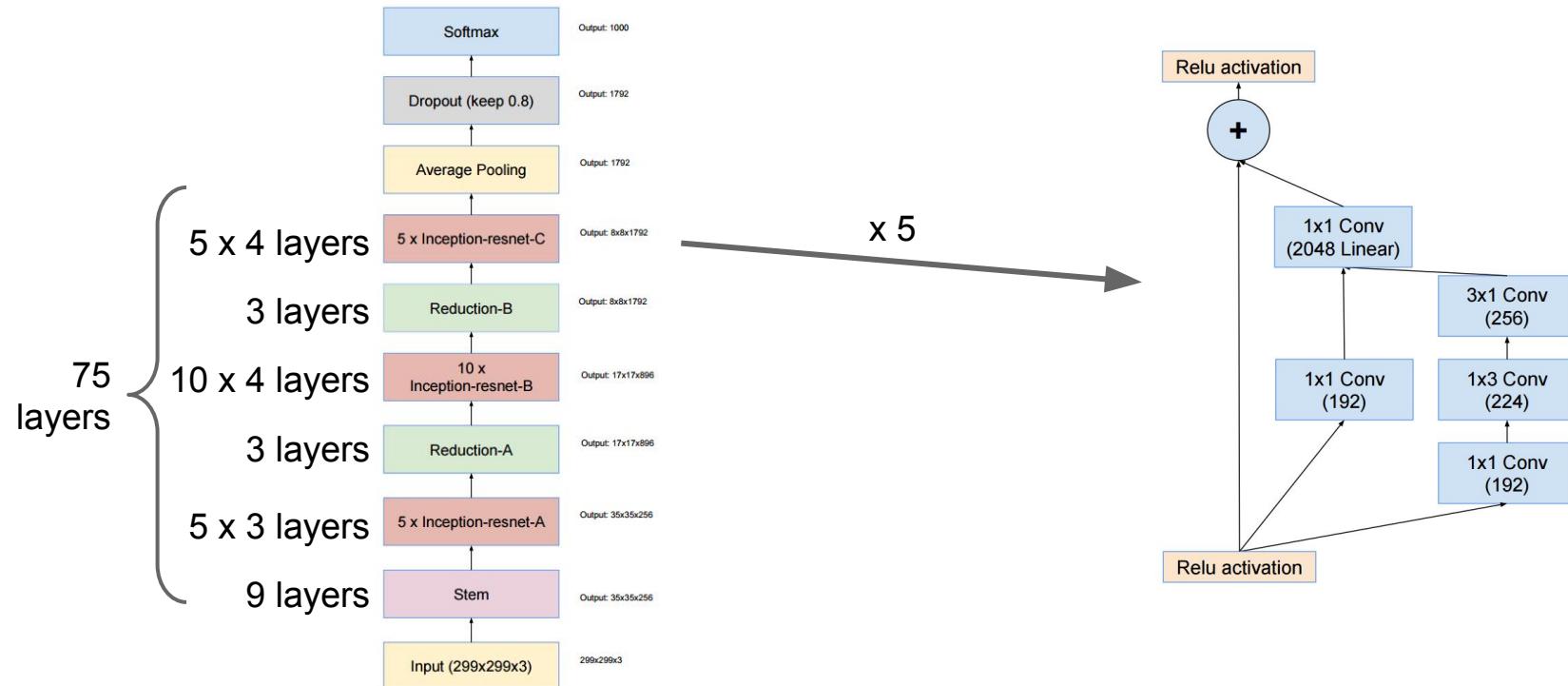
# Inception-ResNet-v2



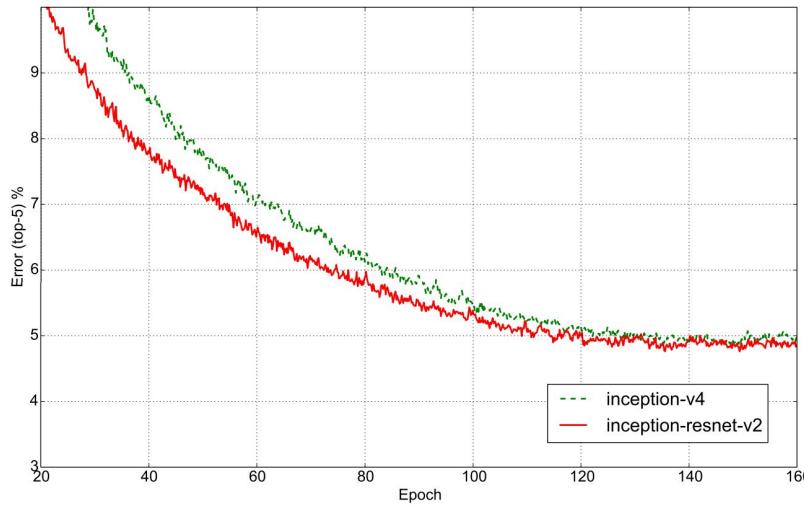
# Inception-ResNet-v2



# 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**



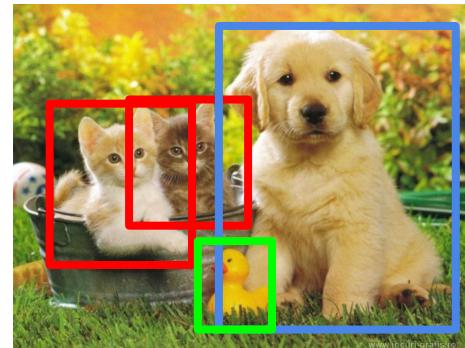
CAT

**Classification + Localization**



CAT

**Object Detection**



CAT, DOG, DUCK

**Segmentation**



CAT, DOG, DUCK

Single object

Multiple objects

# Computer Vision Tasks

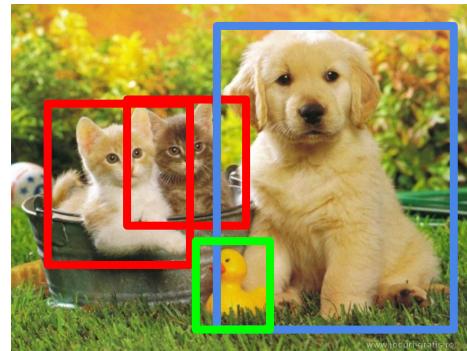
Classification



**Classification  
+ Localization**



Object Detection



Segmentation



Lecture 8

# Computer Vision Tasks

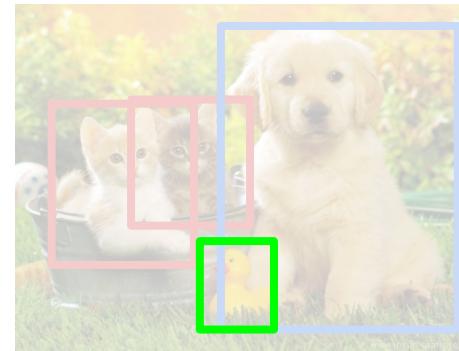
Classification



Classification  
+ Localization



Object Detection



Segmentation



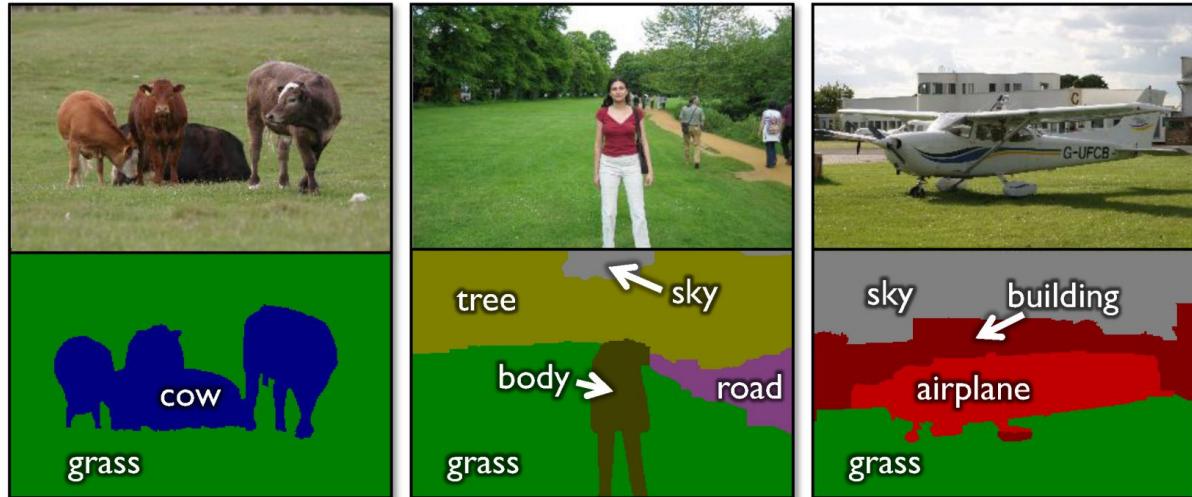
Today

# Semantic Segmentation

Label every pixel!

Don't differentiate instances (cows)

Classic computer vision problem



object classes	building	grass	tree	cow	sheep	sky	airplane	water	face	car
bicycle	flower	sign	bird	book	chair	road	cat	dog	body	boat

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)

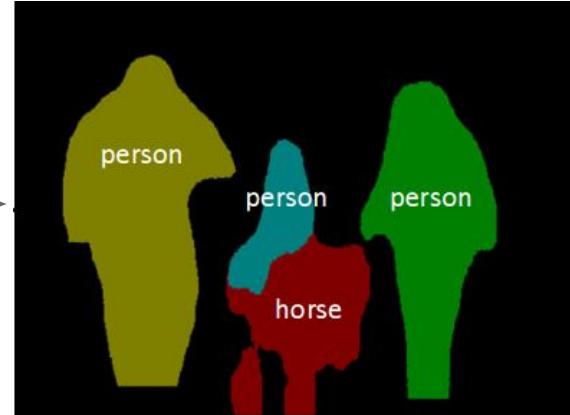


Figure credit: Dai et al, “Instance-aware Semantic Segmentation via Multi-task Network Cascades”, arXiv 2015

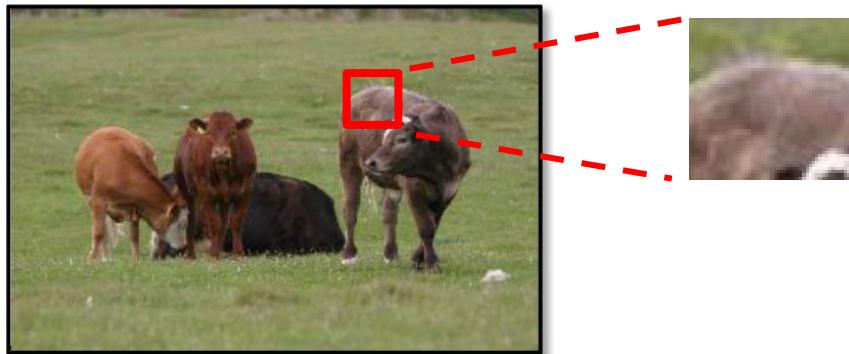
# Semantic Segmentation

# Semantic Segmentation

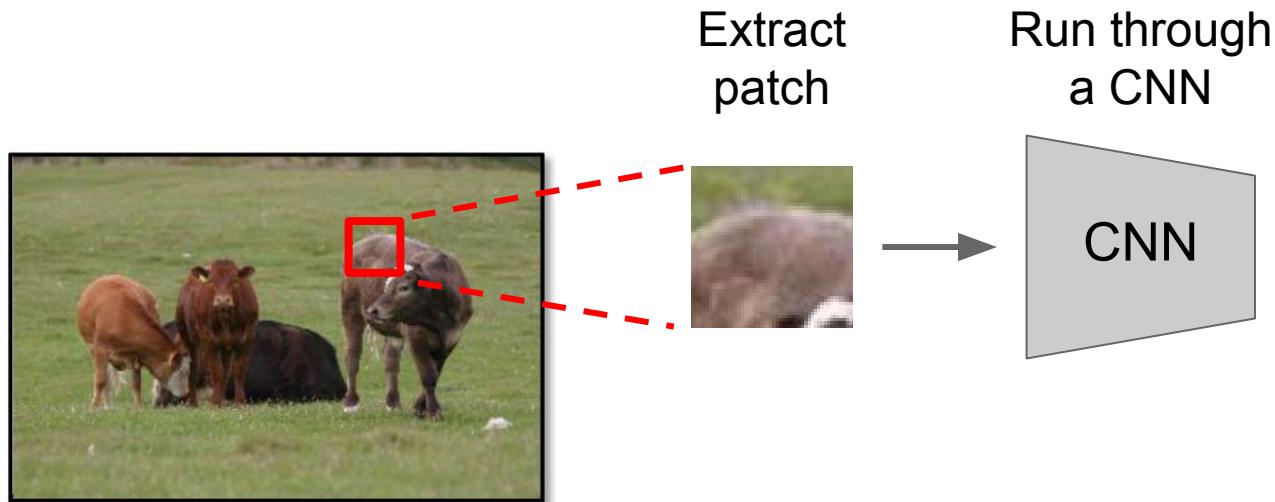


# Semantic Segmentation

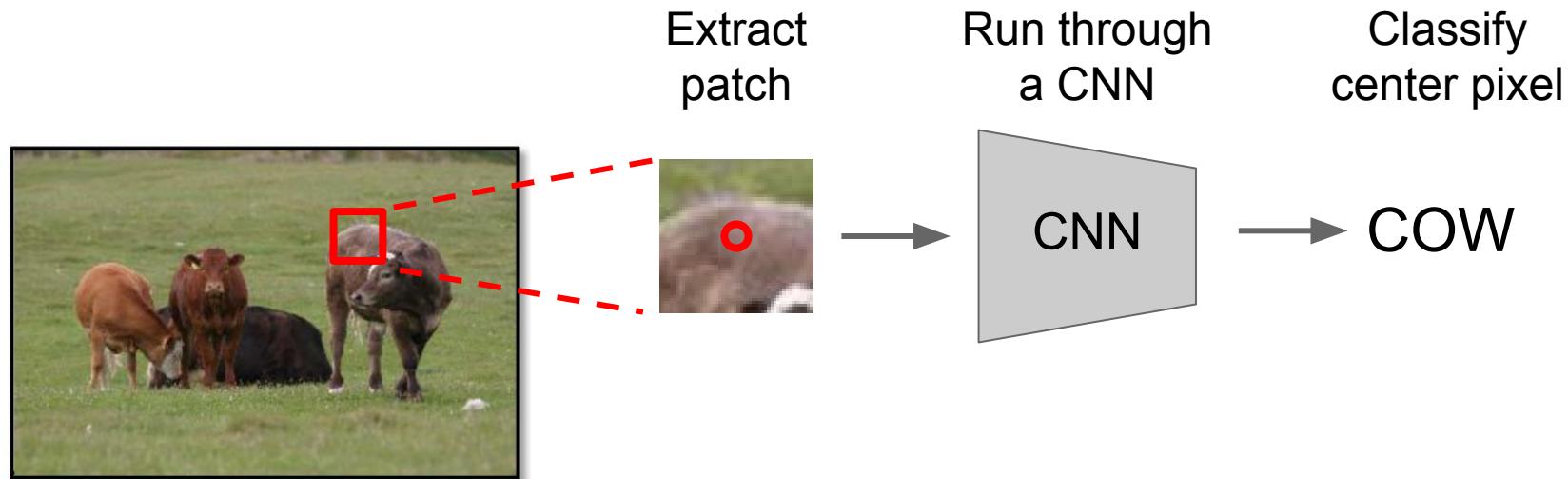
Extract  
patch



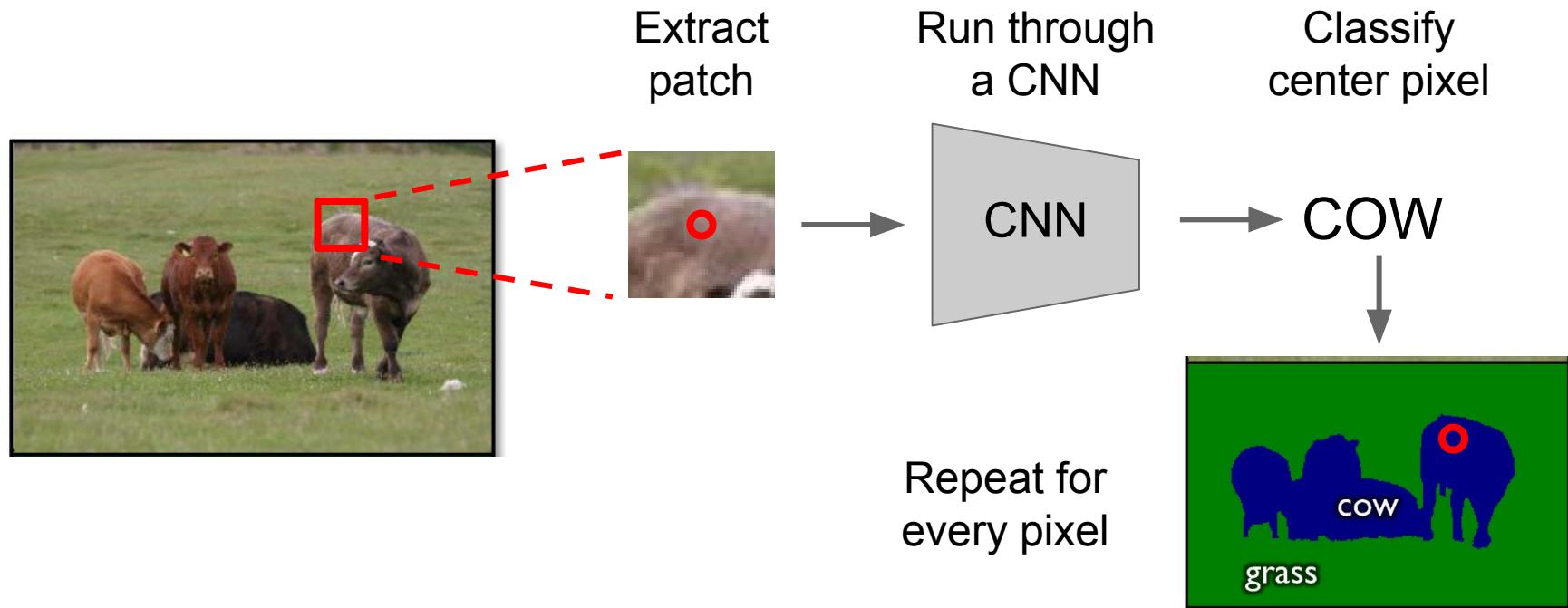
# Semantic Segmentation



# Semantic Segmentation

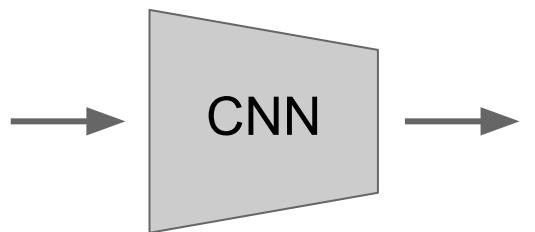


# Semantic Segmentation



# Semantic Segmentation

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



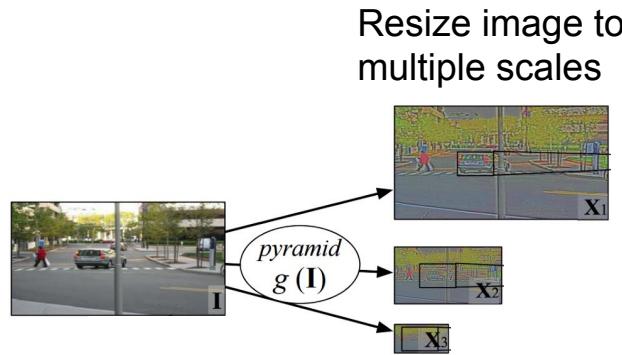
Smaller output  
due to pooling

# Semantic Segmentation: Multi-Scale



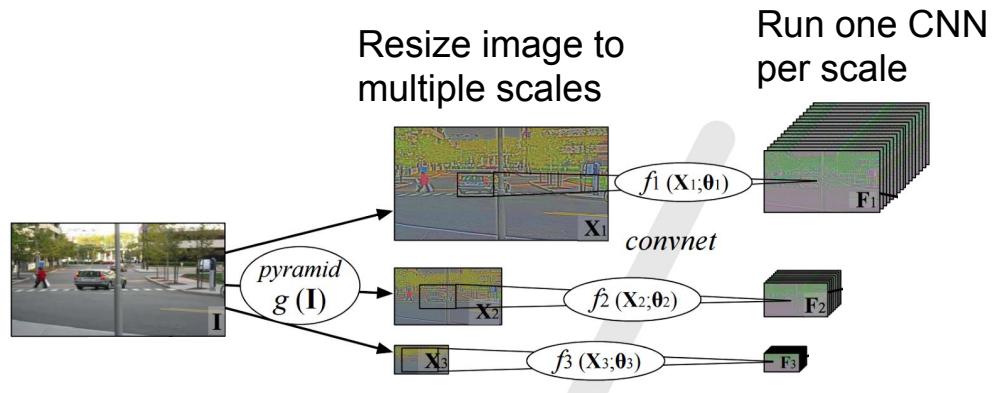
Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013

# Semantic Segmentation: Multi-Scale



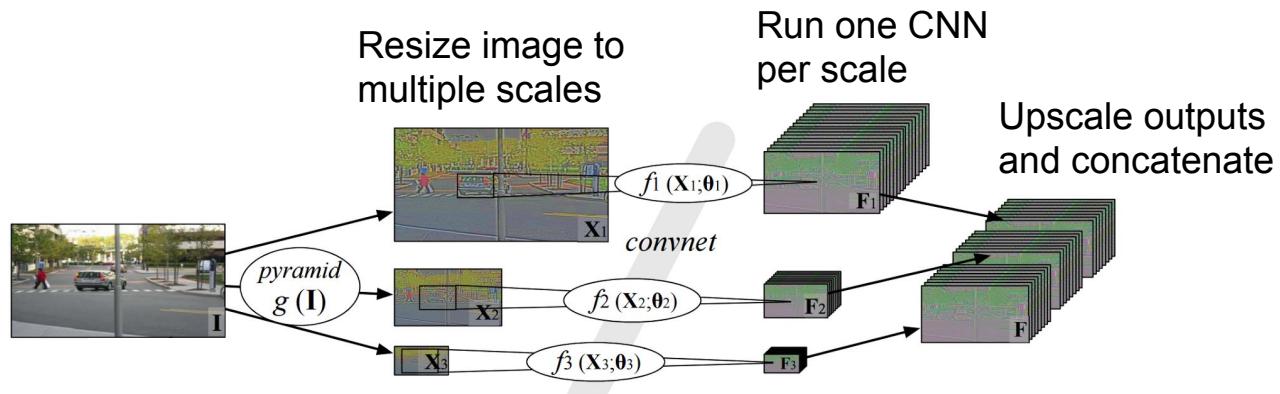
Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013

# Semantic Segmentation: Multi-Scale



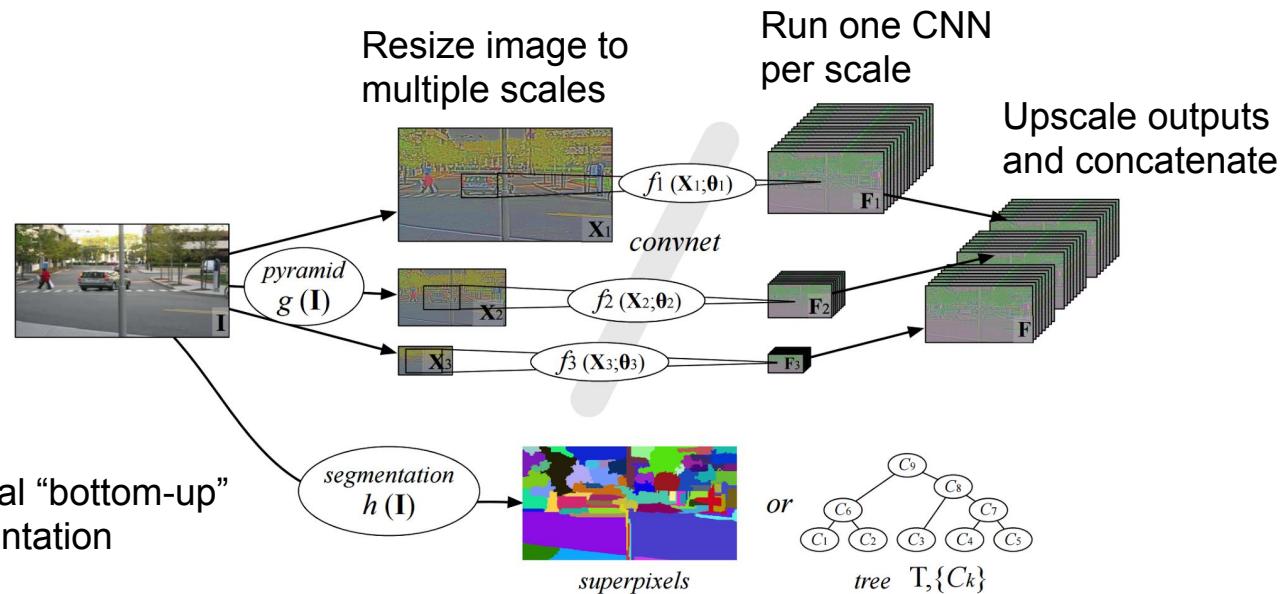
Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013

# Semantic Segmentation: Multi-Scale



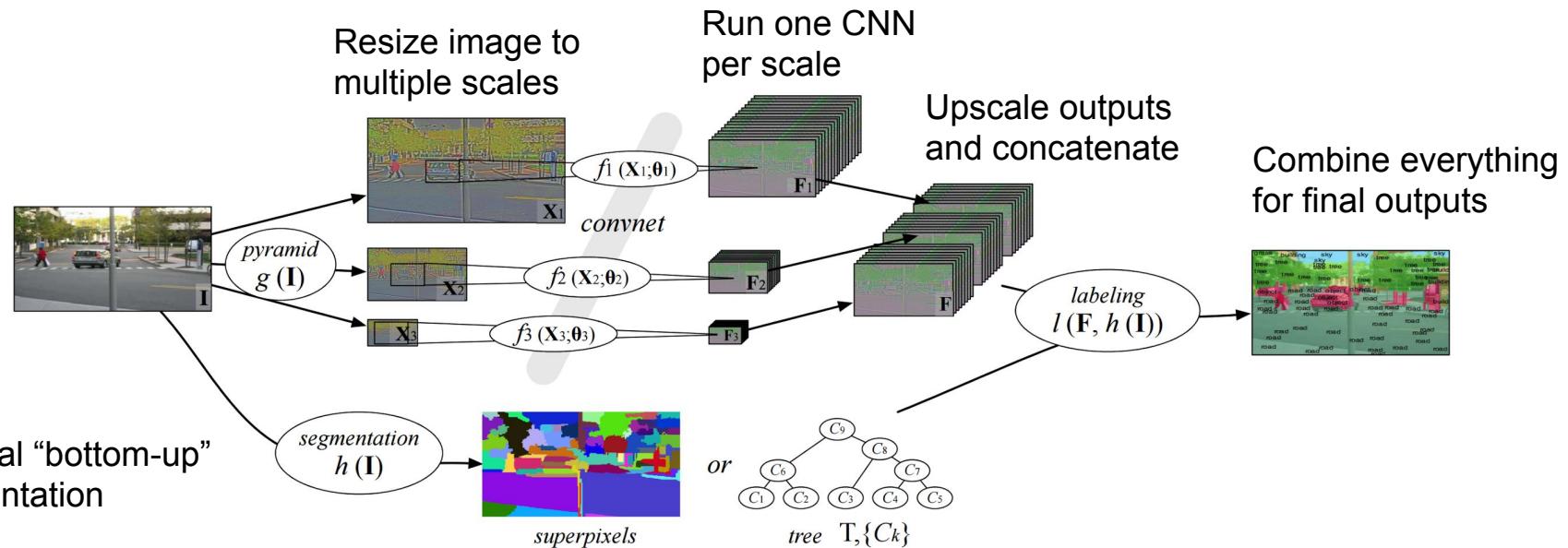
Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013

# Semantic Segmentation: Multi-Scale



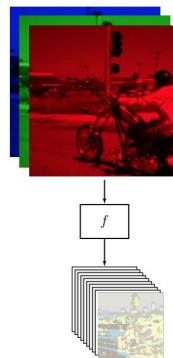
Farabet et al, “Learning Hierarchical Features for Scene Labeling,” TPAMI 2013

# Semantic Segmentation: Multi-Scale



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

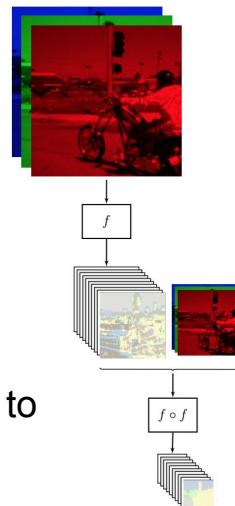
# Semantic Segmentation: Refinement



Apply CNN once  
to get labels

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

# Semantic Segmentation: Refinement

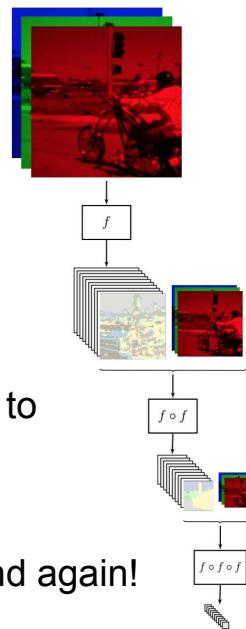


Apply CNN once  
to get labels

Apply AGAIN to  
refine labels

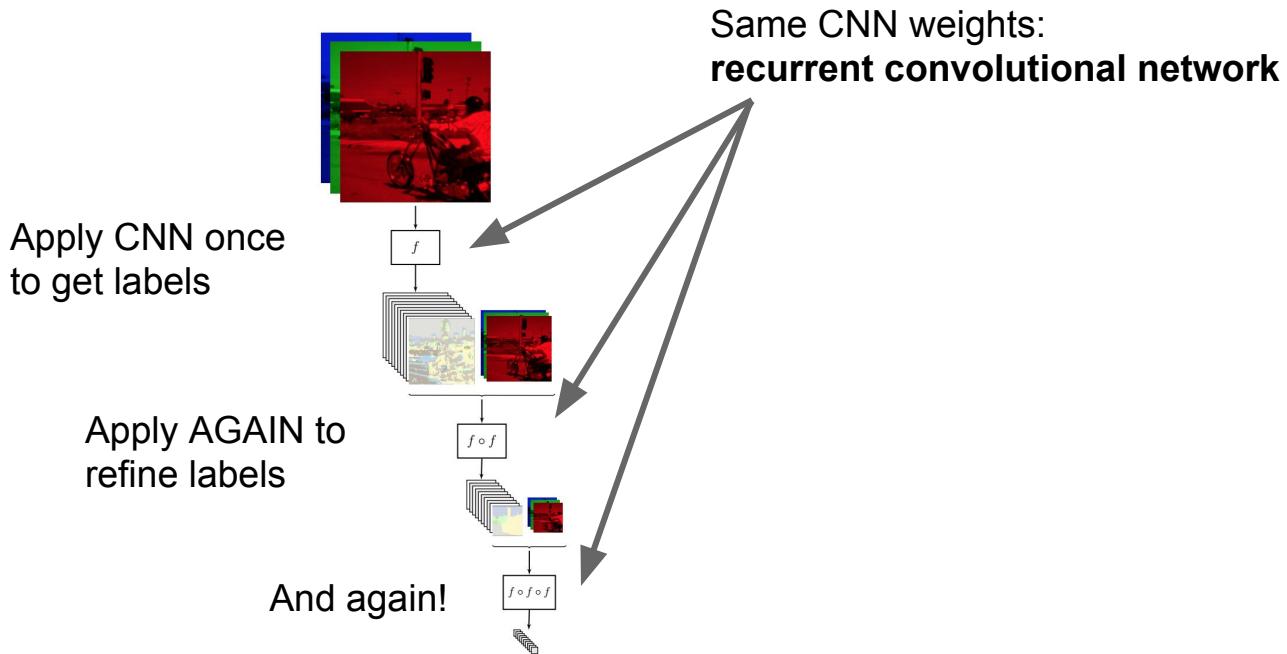
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

# Semantic Segmentation: Refinement



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

# Semantic Segmentation: Refinement



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

# Semantic Segmentation: Refinement

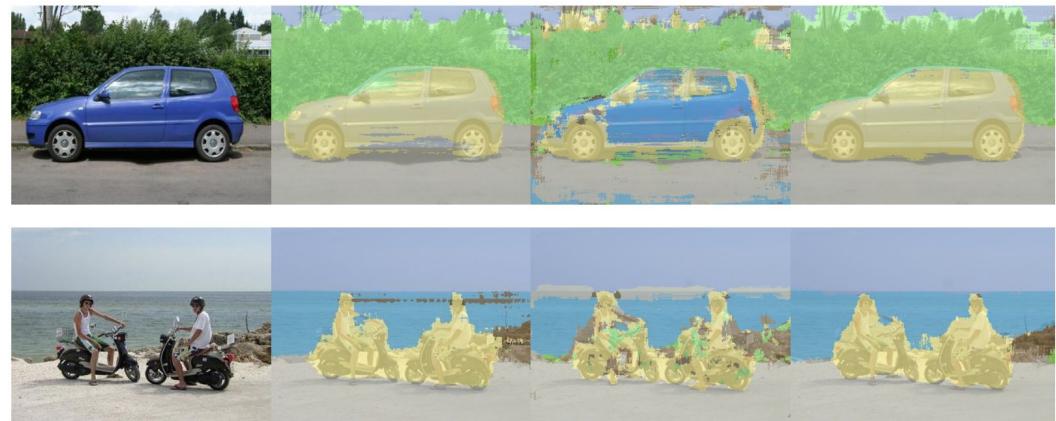
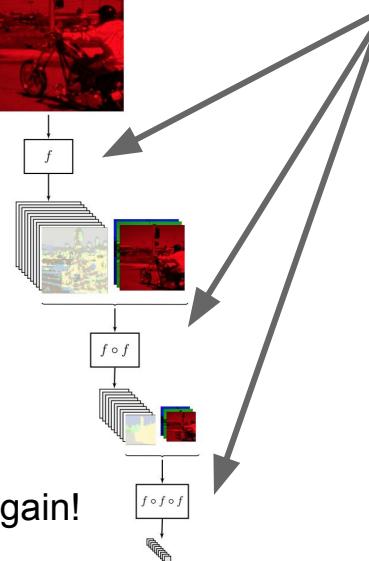
Apply CNN once  
to get labels



Same CNN weights:  
**recurrent convolutional network**

Apply AGAIN to  
refine labels

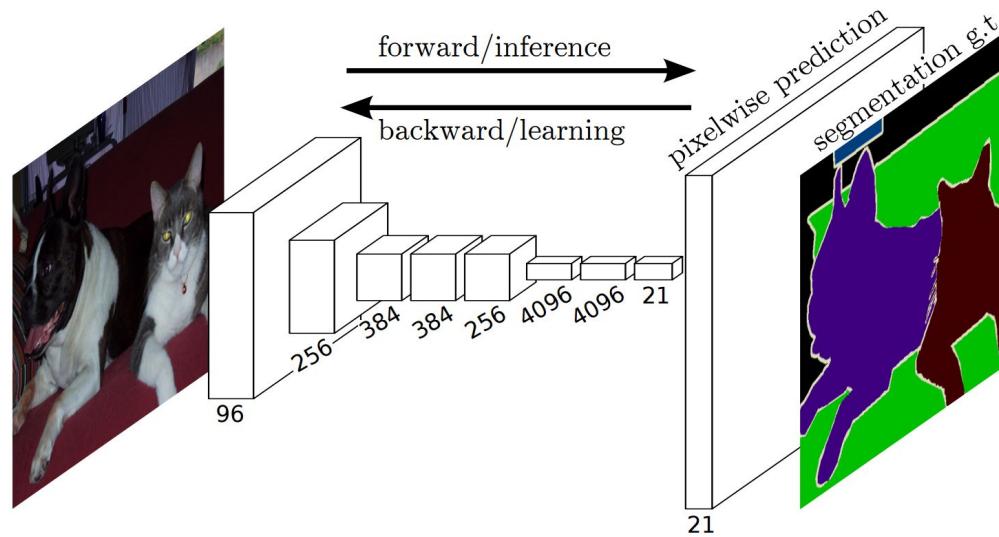
And again!



More iterations improve results

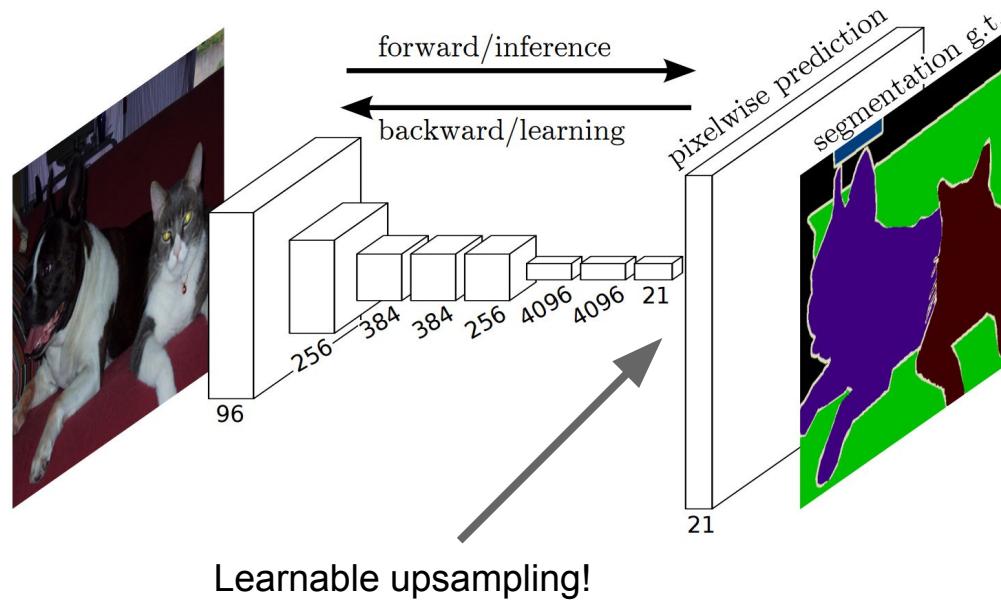
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

# Semantic Segmentation: Upsampling



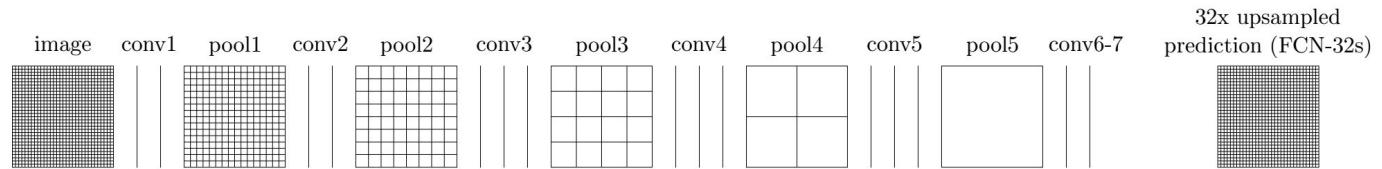
Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015

# Semantic Segmentation: Upsampling



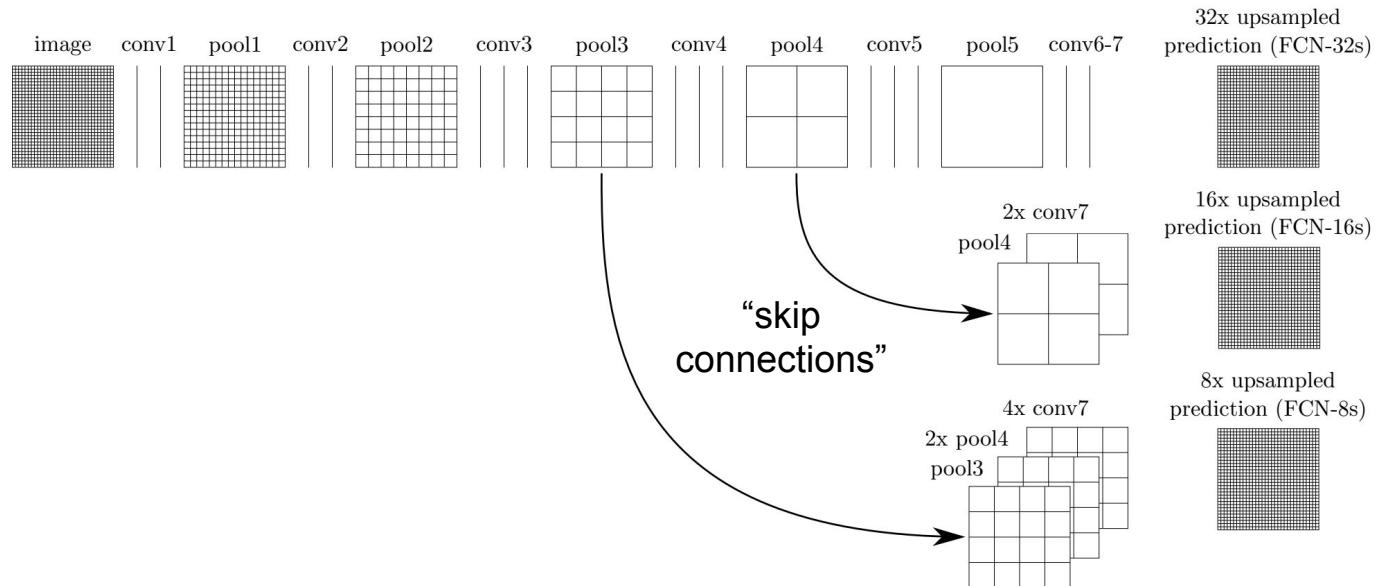
Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015

# Semantic Segmentation: Upsampling



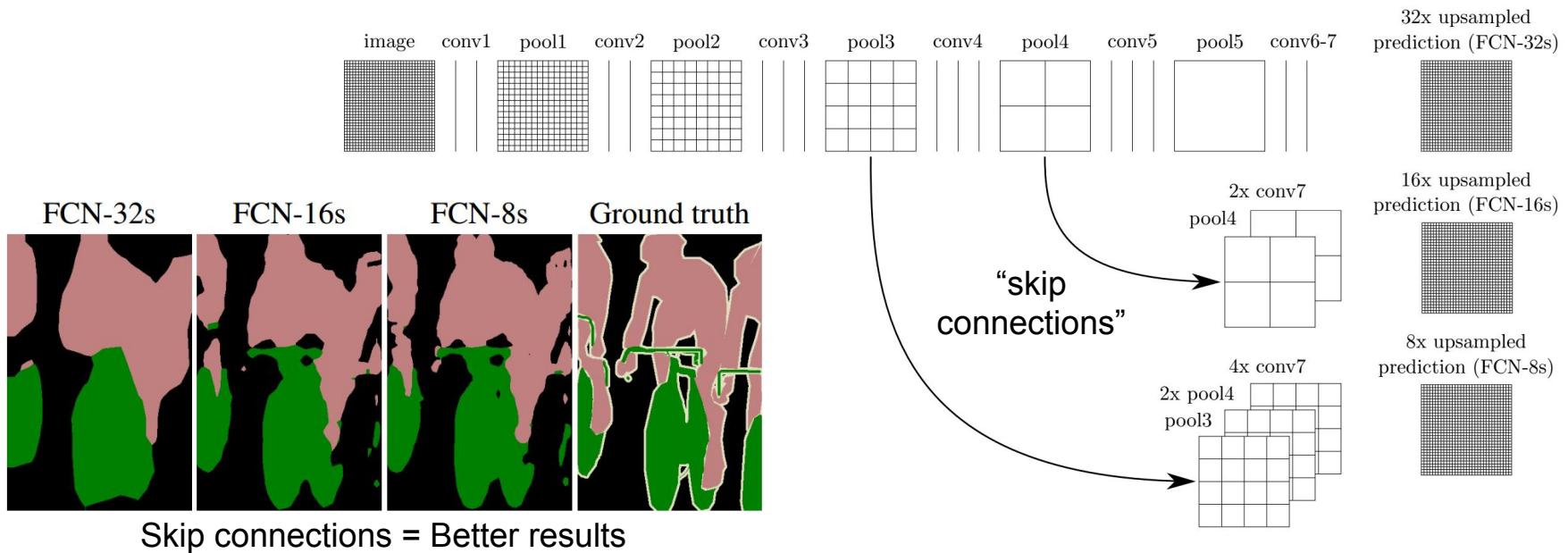
Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015

# Semantic Segmentation: Upsampling



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

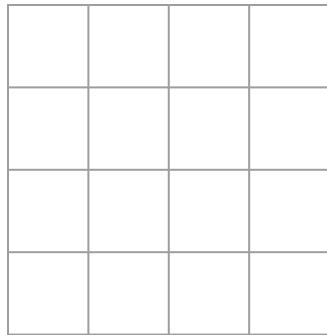
# Semantic Segmentation: Upsampling



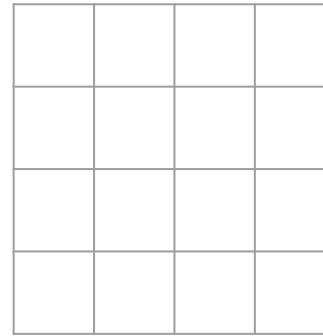
Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015

# Learnable Upsampling: “Deconvolution”

Typical  $3 \times 3$  convolution, stride 1 pad 1



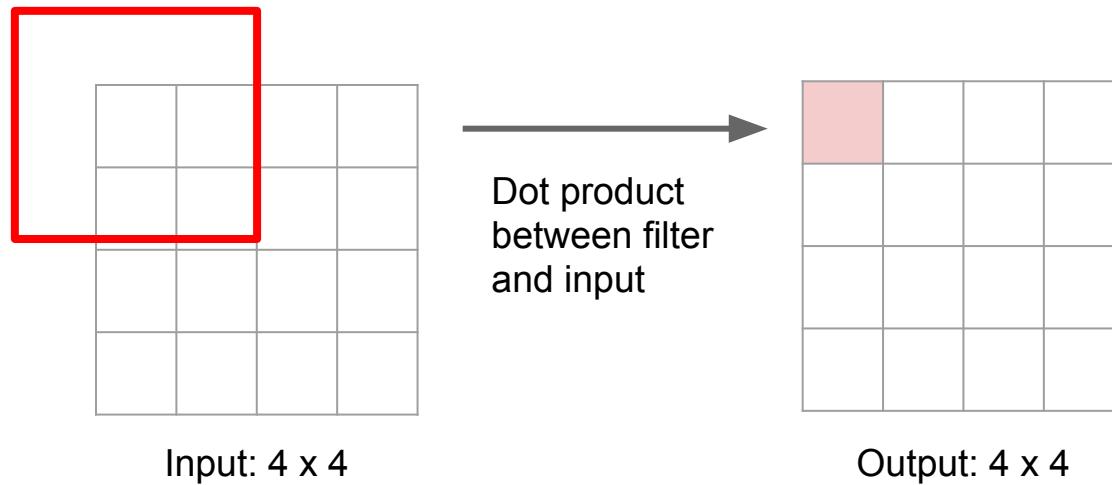
Input:  $4 \times 4$



Output:  $4 \times 4$

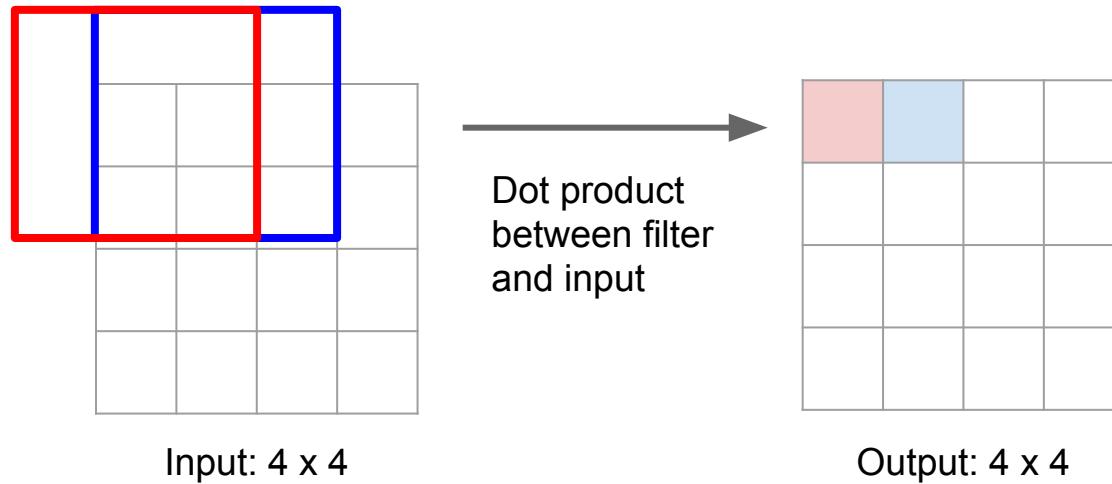
# Learnable Upsampling: “Deconvolution”

Typical  $3 \times 3$  convolution, stride 1 pad 1



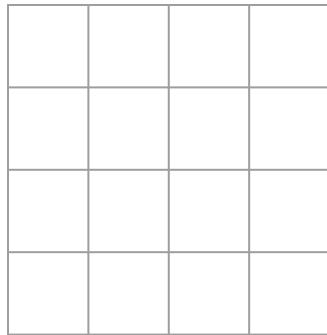
# Learnable Upsampling: “Deconvolution”

Typical  $3 \times 3$  convolution, stride 1 pad 1

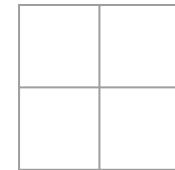


# Learnable Upsampling: “Deconvolution”

Typical  $3 \times 3$  convolution, **stride 2** pad 1



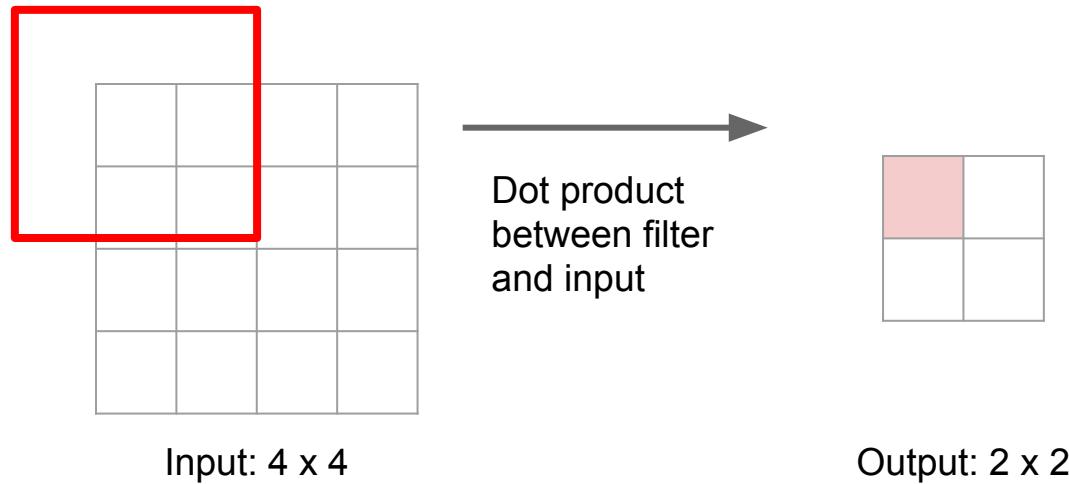
Input:  $4 \times 4$



Output:  $2 \times 2$

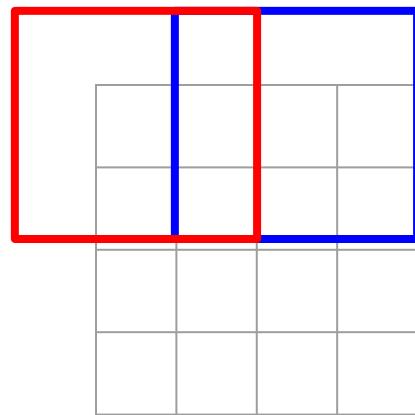
# Learnable Upsampling: “Deconvolution”

Typical  $3 \times 3$  convolution, stride 2 pad 1



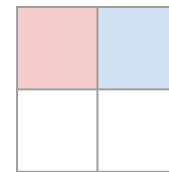
# Learnable Upsampling: “Deconvolution”

Typical  $3 \times 3$  convolution, stride 2 pad 1



Input:  $4 \times 4$

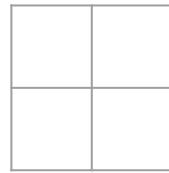
Dot product  
between filter  
and input



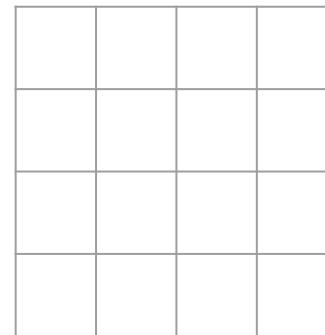
Output:  $2 \times 2$

# Learnable Upsampling: “Deconvolution”

3 x 3 “deconvolution”, stride 2 pad 1



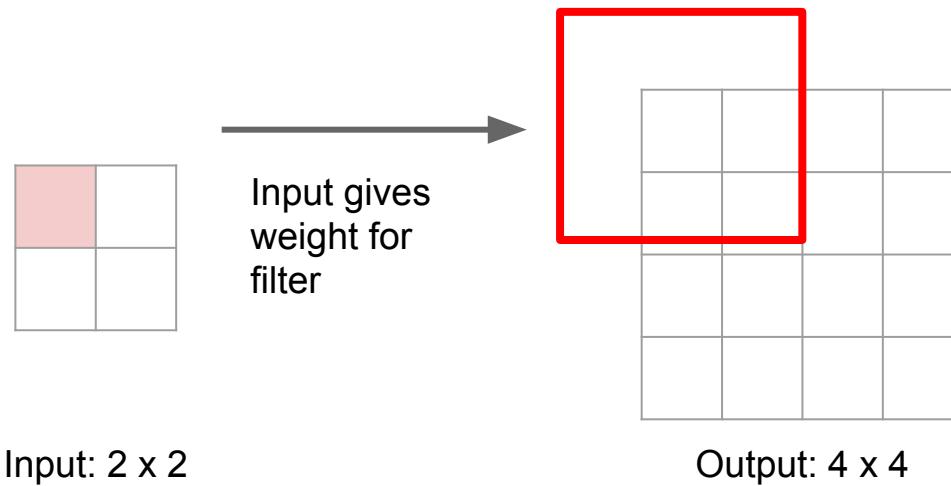
Input: 2 x 2



Output: 4 x 4

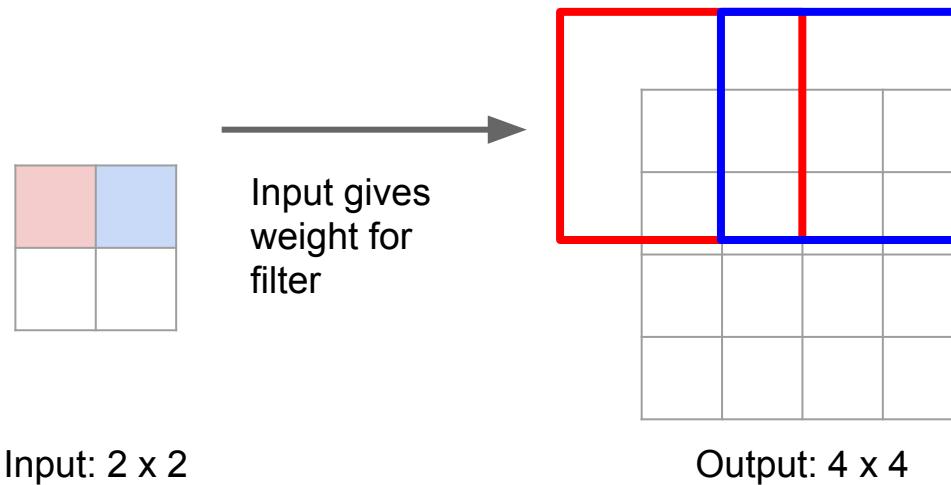
# Learnable Upsampling: “Deconvolution”

3 x 3 “deconvolution”, stride 2 pad 1

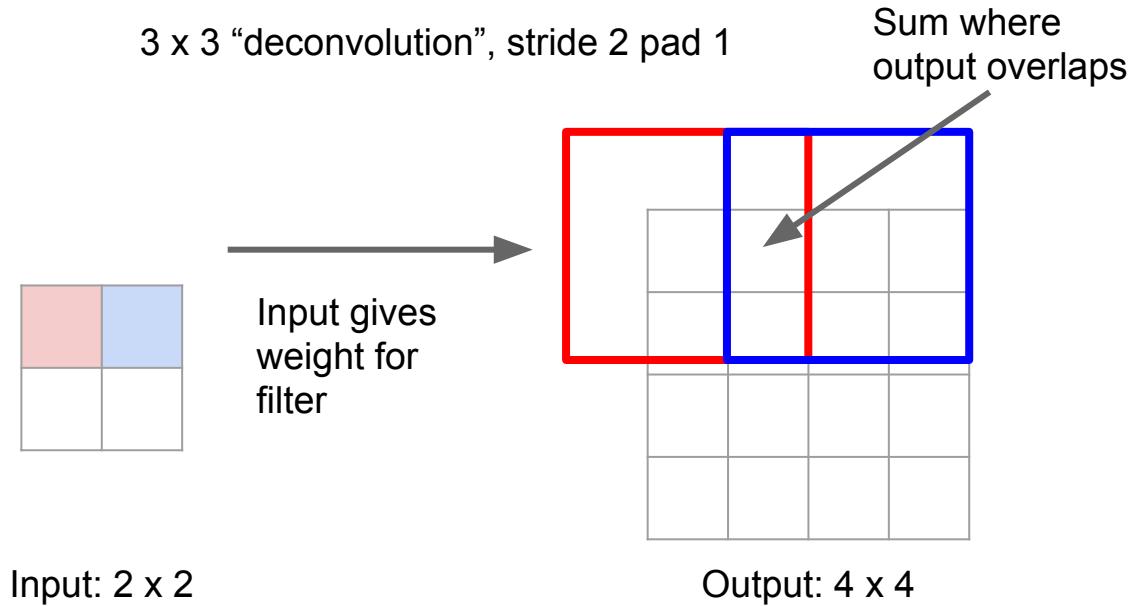


# Learnable Upsampling: “Deconvolution”

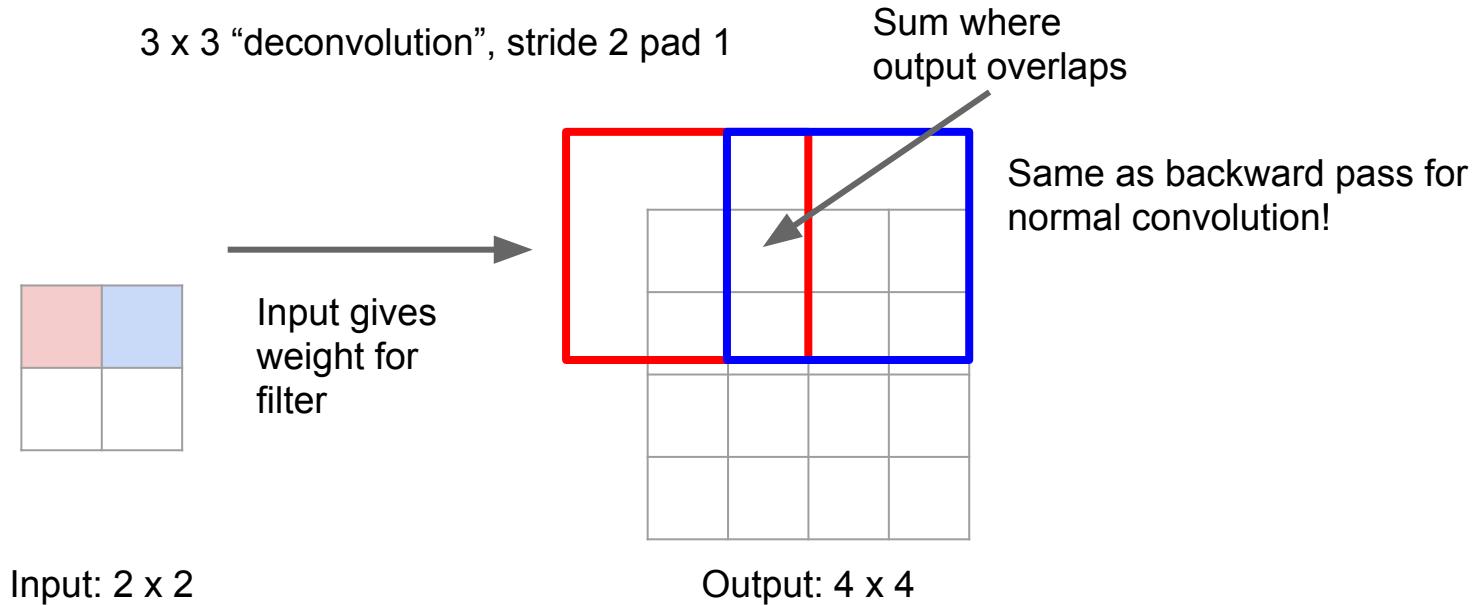
3 x 3 “deconvolution”, stride 2 pad 1



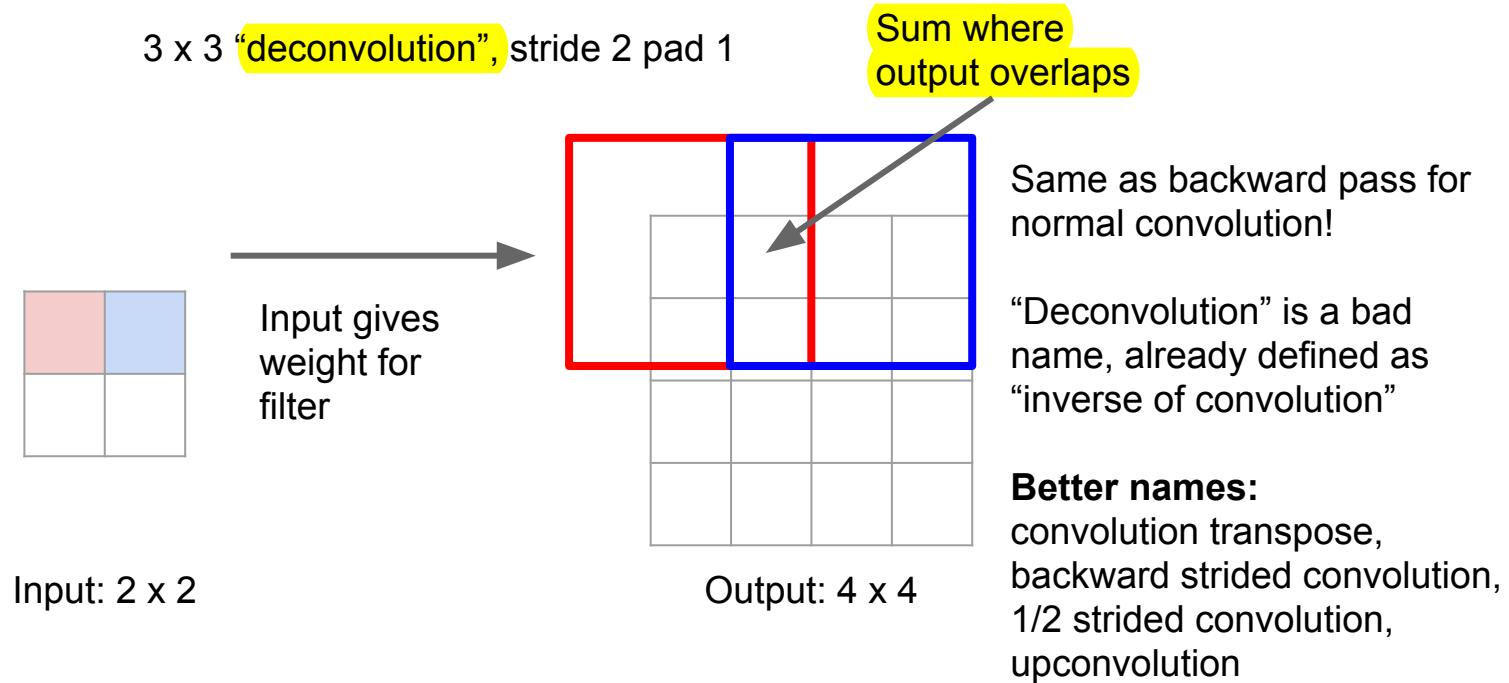
# Learnable Upsampling: “Deconvolution”



# Learnable Upsampling: “Deconvolution”



# Learnable Upsampling: “Deconvolution”



# Learnable Upsampling: “Deconvolution”

<sup>1</sup>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”

<sup>1</sup>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

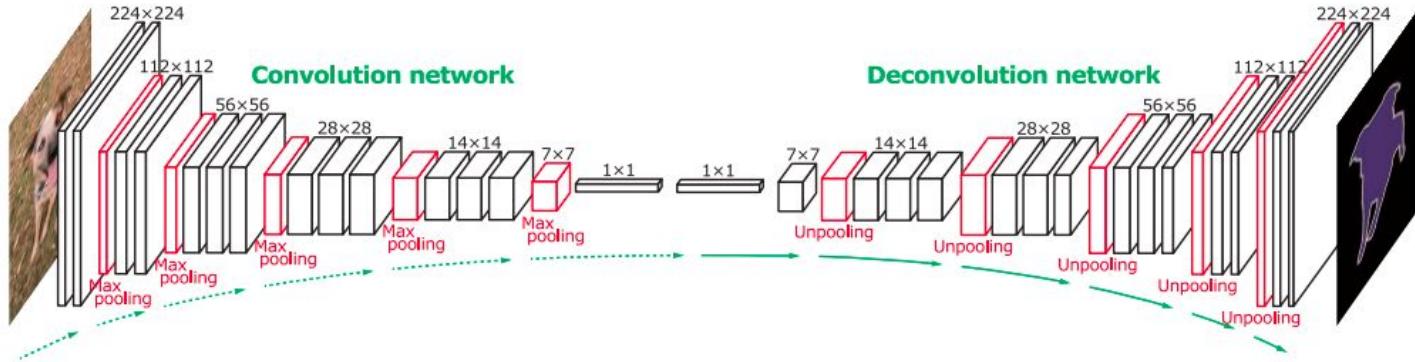
Great explanation  
in appendix



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

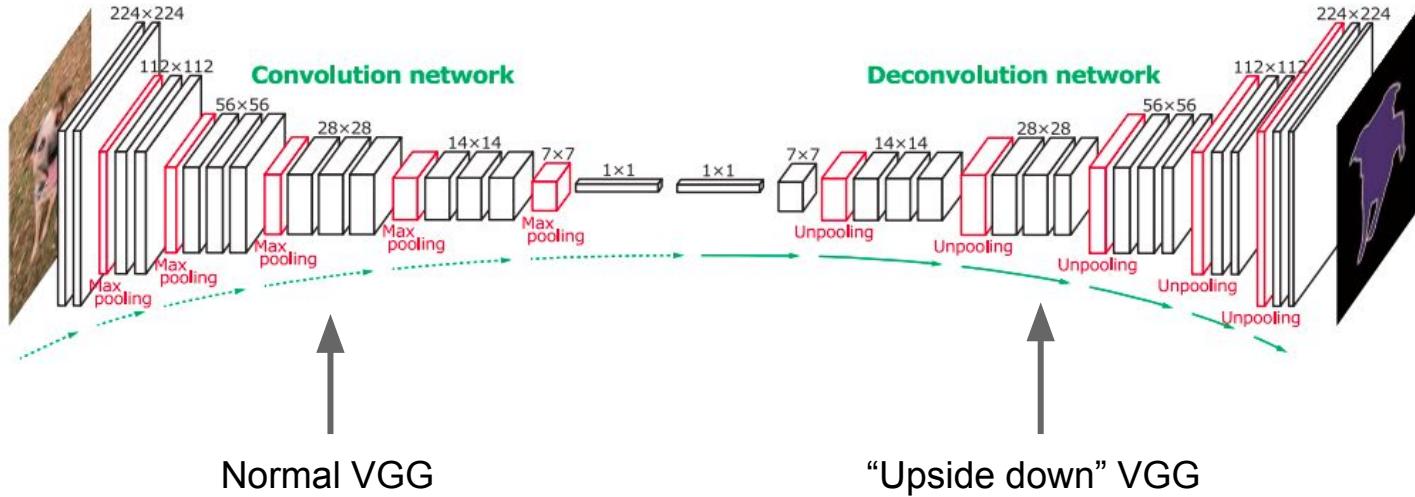
**Better names:**  
convolution transpose,  
backward strided convolution,  
1/2 strided convolution,  
upconvolution

# Semantic Segmentation: Upsampling



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

# Semantic Segmentation: Upsampling



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)

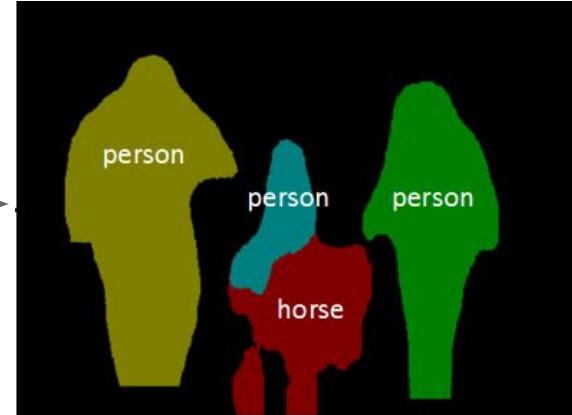


Figure credit: Dai et al, “Instance-aware Semantic Segmentation via Multi-task Network Cascades”, arXiv 2015

# Instance Segmentation

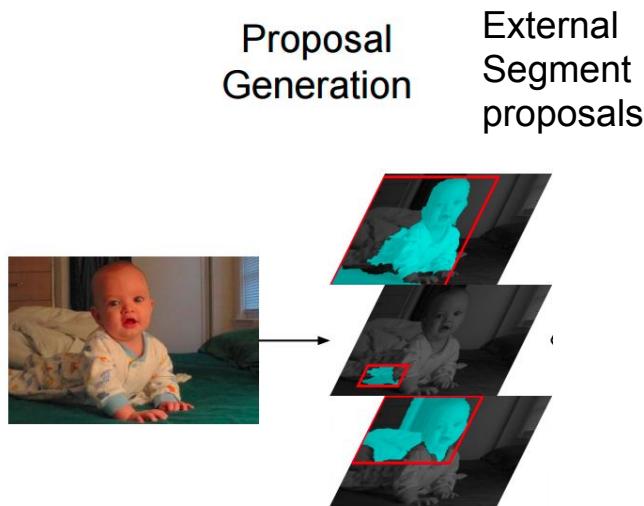
Similar to R-CNN, but  
with segments



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

# Instance Segmentation

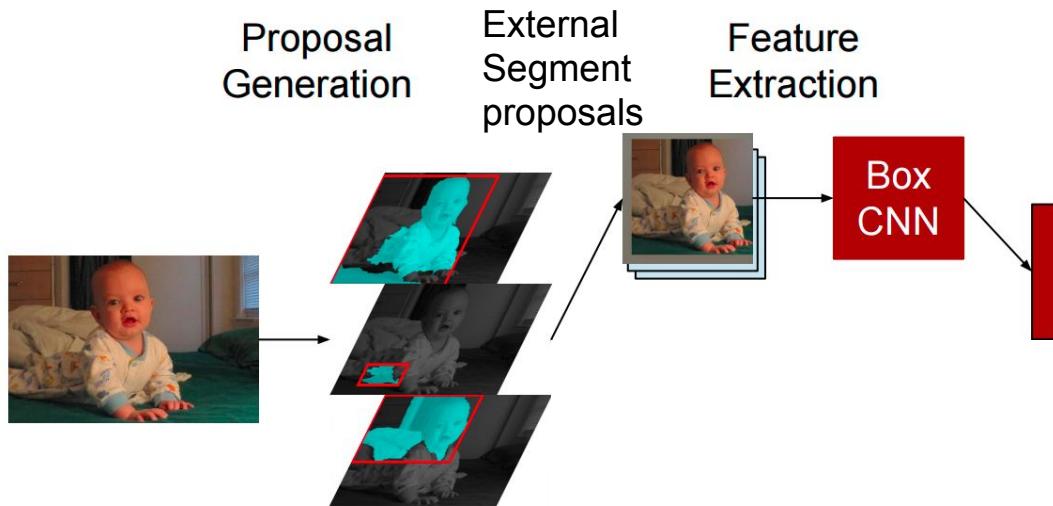
Similar to R-CNN, but  
with segments



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

# Instance Segmentation

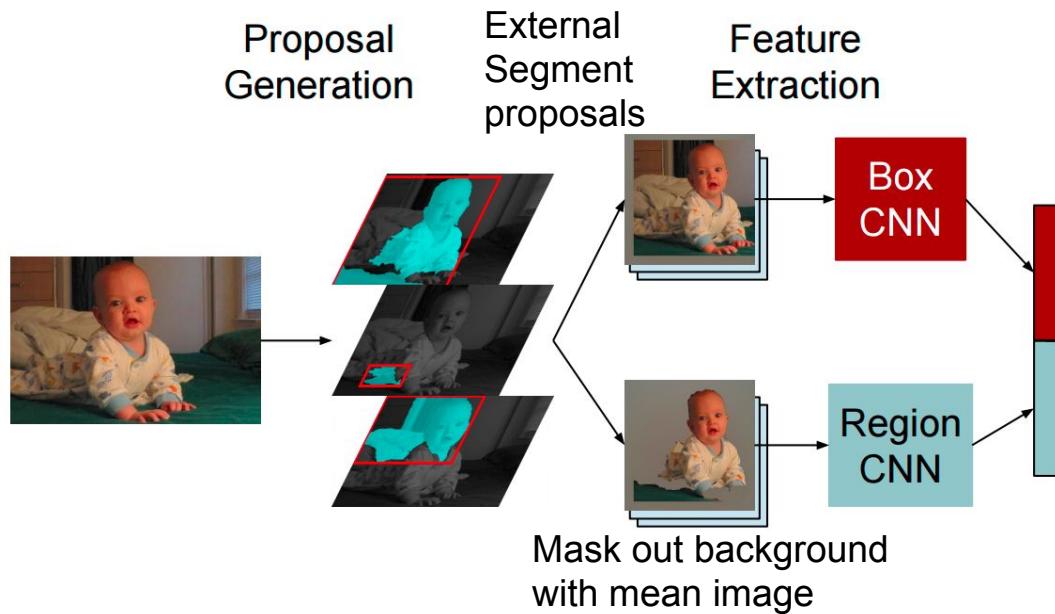
Similar to R-CNN, but with segments



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

# Instance Segmentation

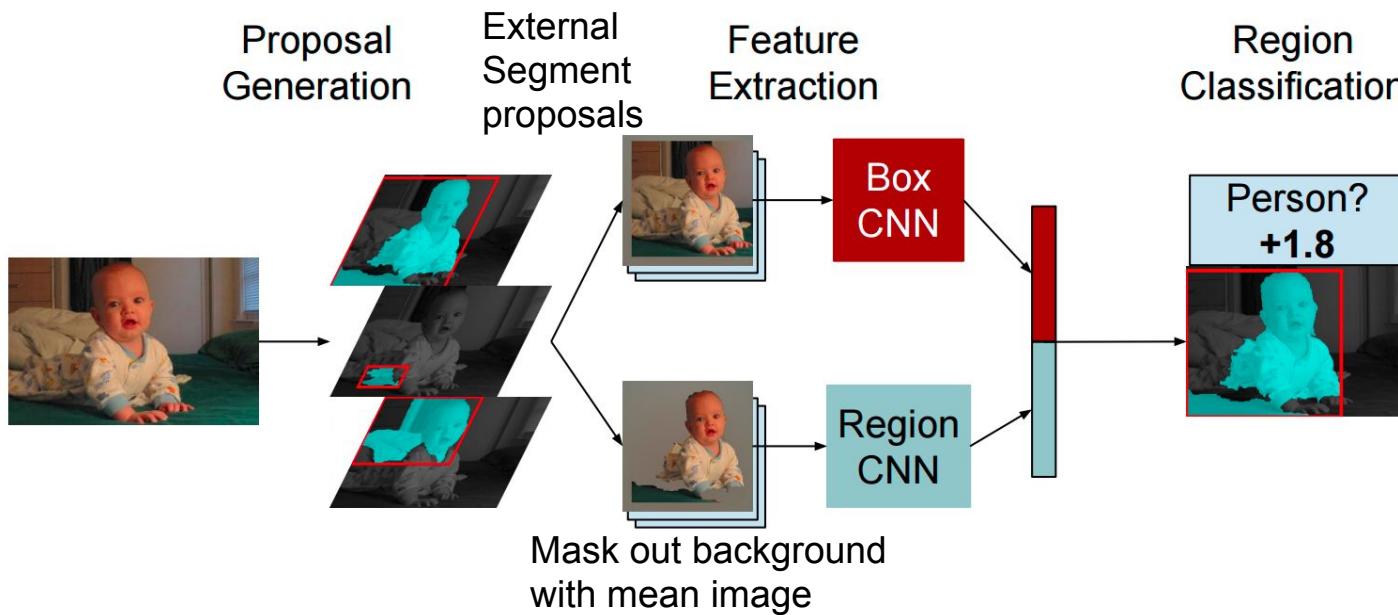
Similar to R-CNN, but with segments



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

# Instance Segmentation

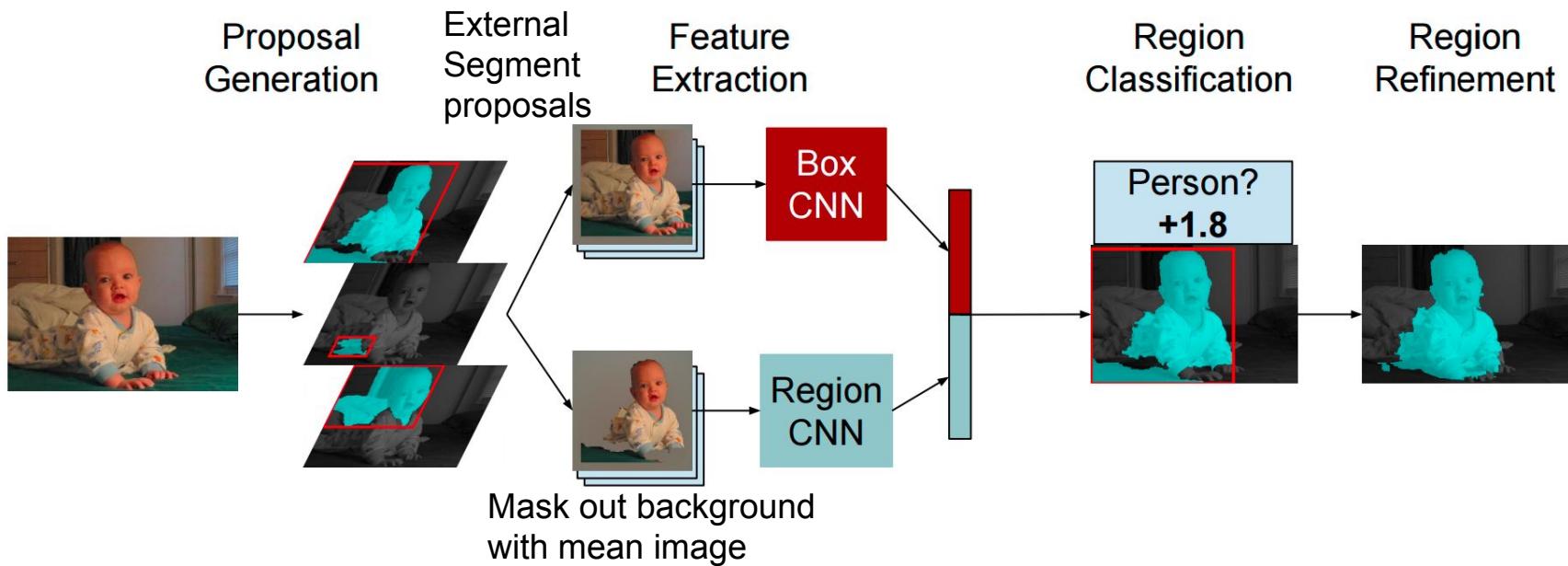
Similar to R-CNN, but with segments



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

# Instance Segmentation

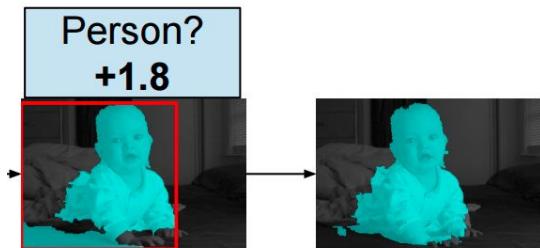
Similar to R-CNN, but with segments



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

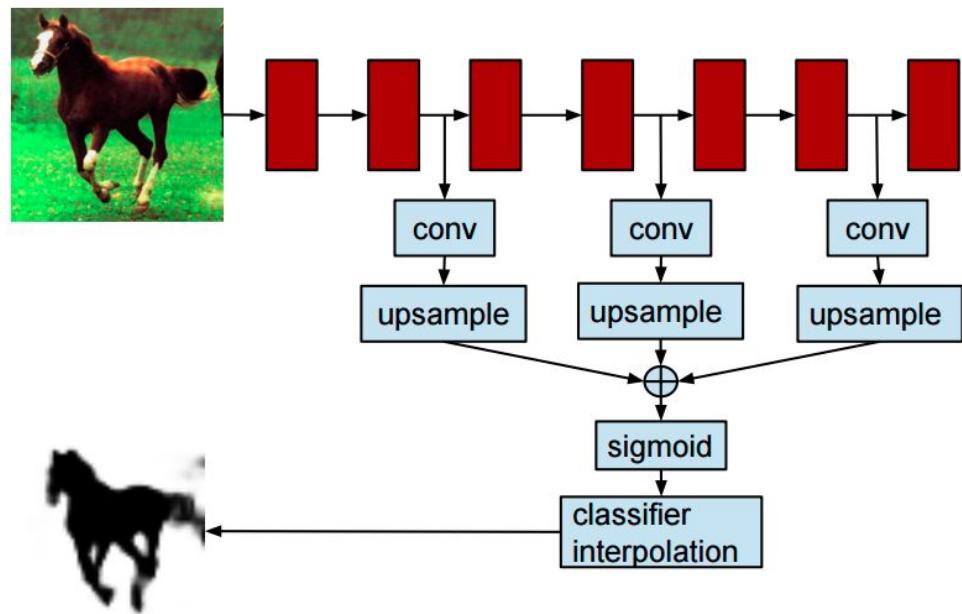
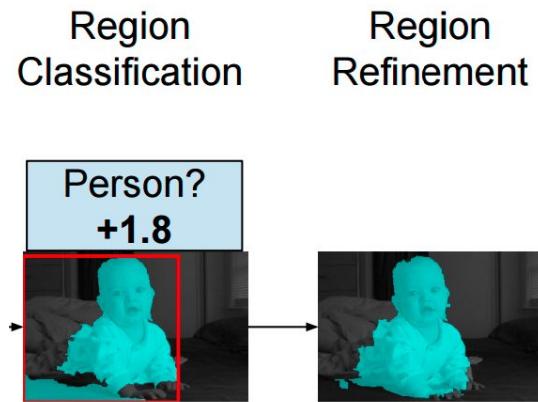
# Instance Segmentation: Hypercolumns

Region  
Classification      Region  
Refinement



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

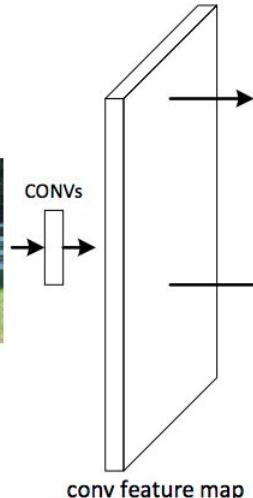


Won COCO 2015  
challenge  
(with ResNet)

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

# Instance Segmentation: Cascades

Similar to  
Faster R-CNN

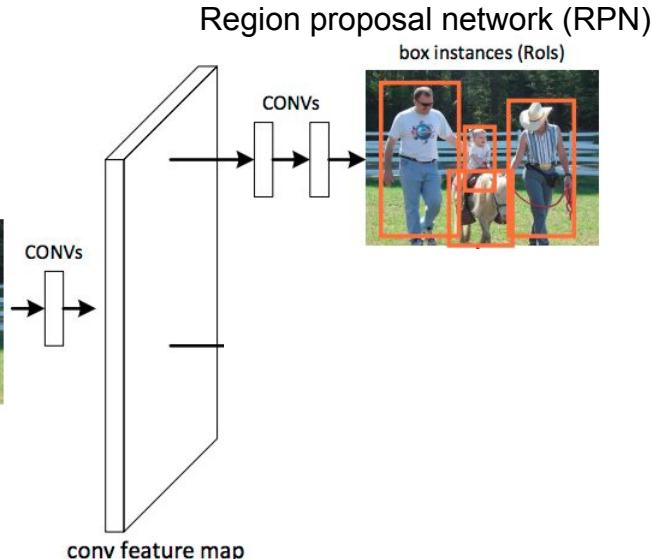


Won COCO 2015  
challenge  
(with ResNet)

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

# Instance Segmentation: Cascades

Similar to  
Faster R-CNN

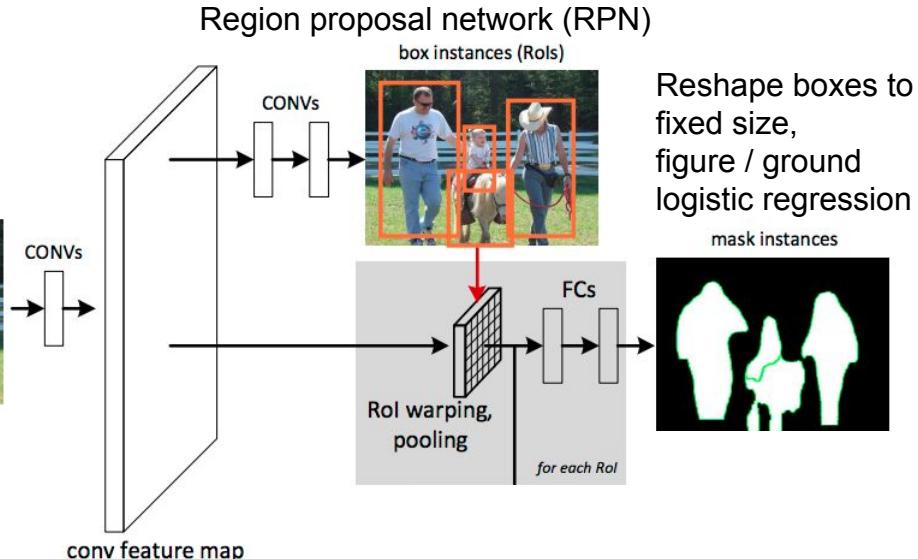


Won COCO 2015  
challenge  
(with ResNet)

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

# Instance Segmentation: Cascades

Similar to  
Faster R-CNN

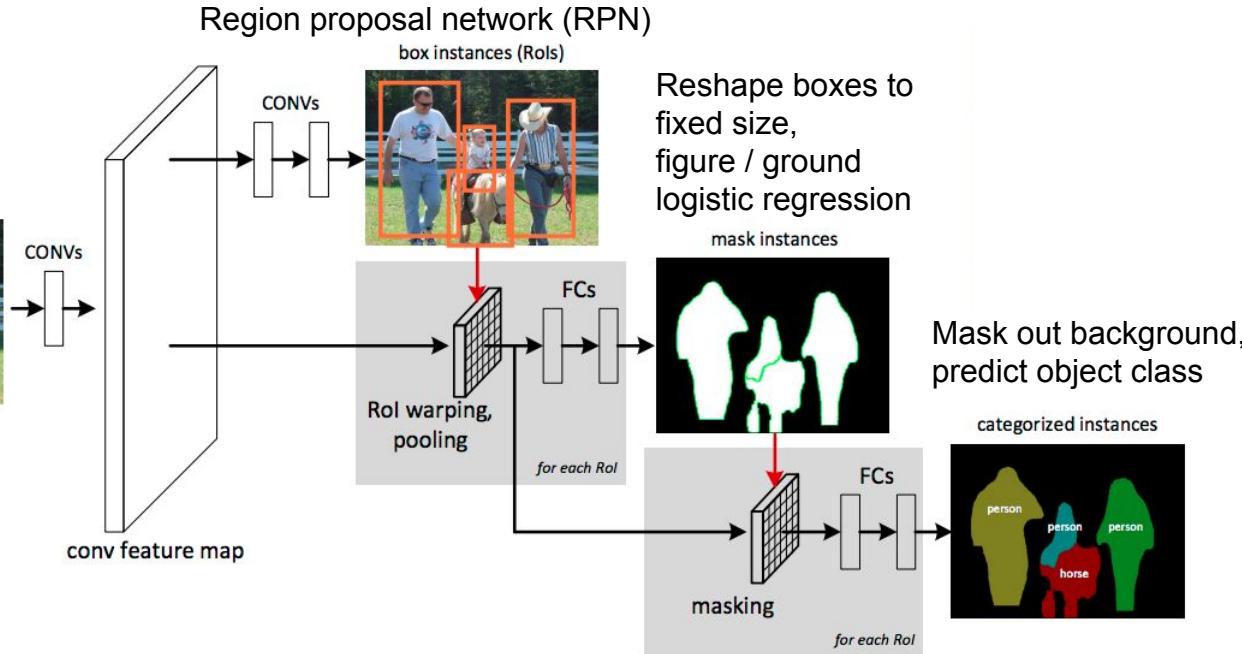


Won COCO 2015  
challenge  
(with ResNet)

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

# Instance Segmentation: Cascades

Similar to  
Faster R-CNN

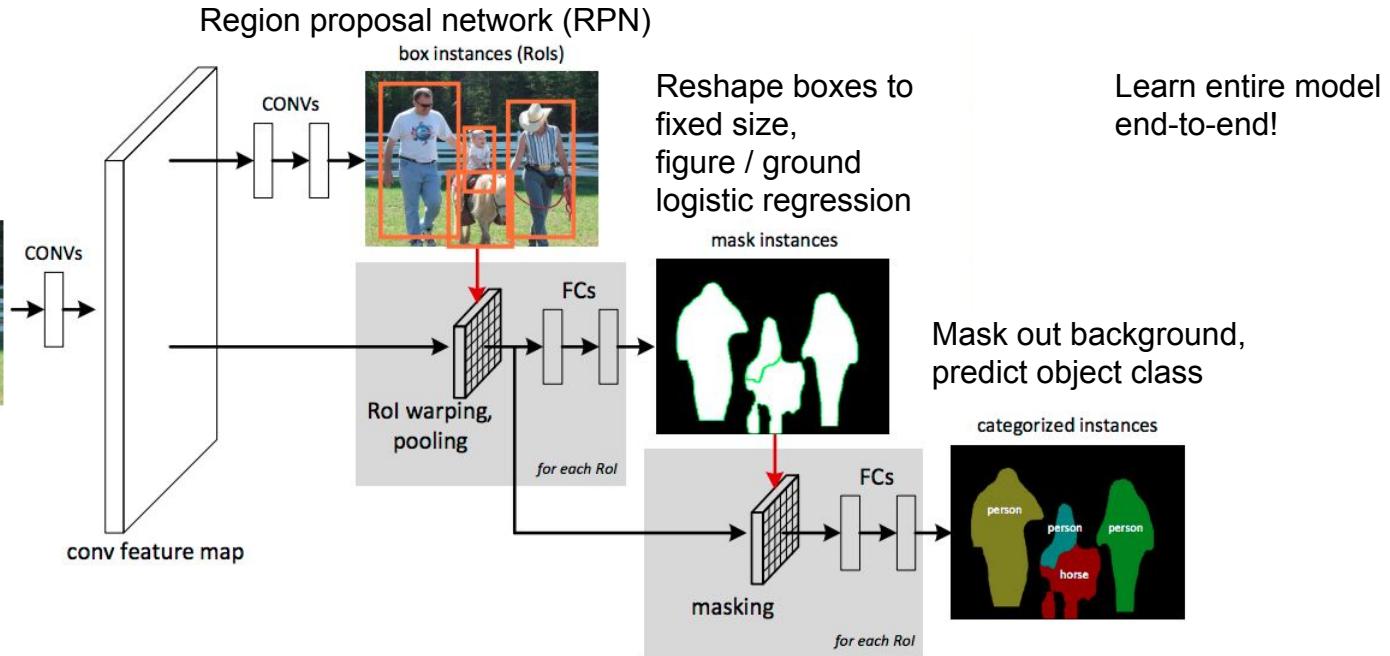


Won COCO 2015  
challenge  
(with ResNet)

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

# Instance Segmentation: Cascades

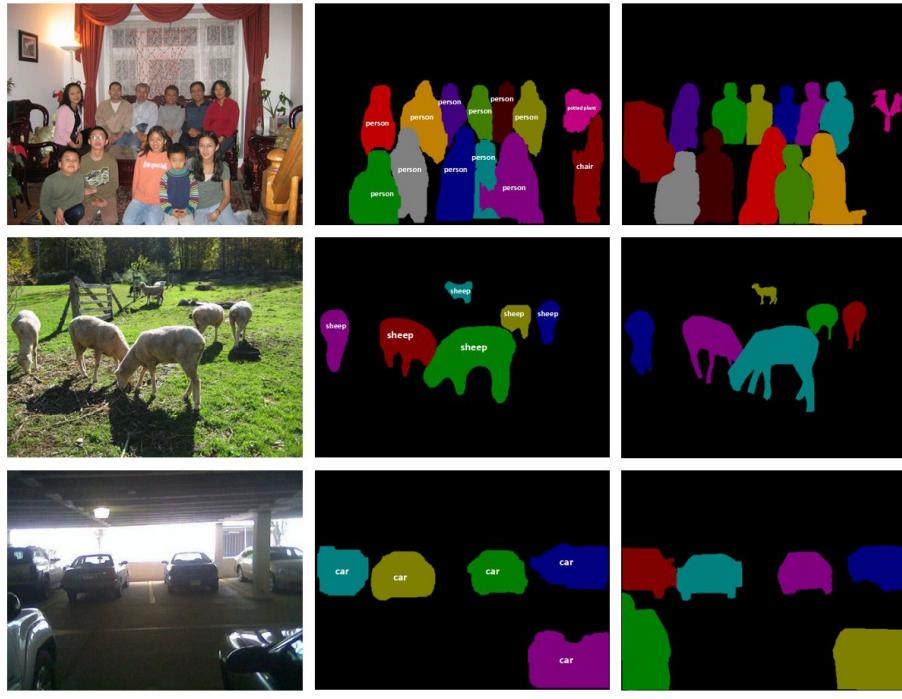
Similar to  
Faster R-CNN



Won COCO 2015 challenge  
(with ResNet)

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

# Instance Segmentation: Cascades



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

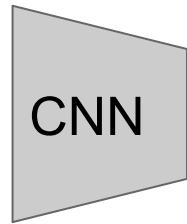


Image:  
 $H \times W \times 3$

# Recall: RNN for Captioning



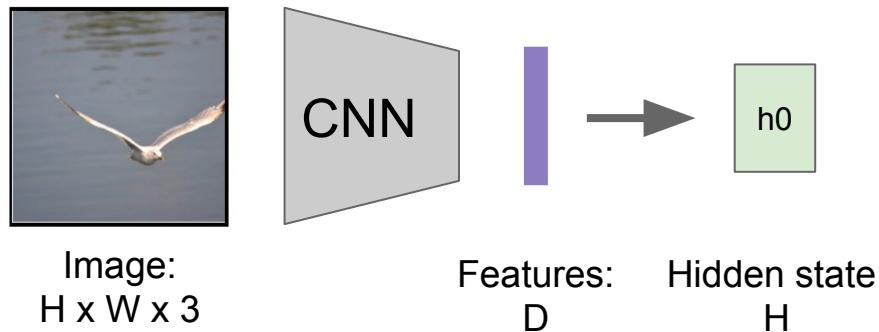
Image:  
 $H \times W \times 3$



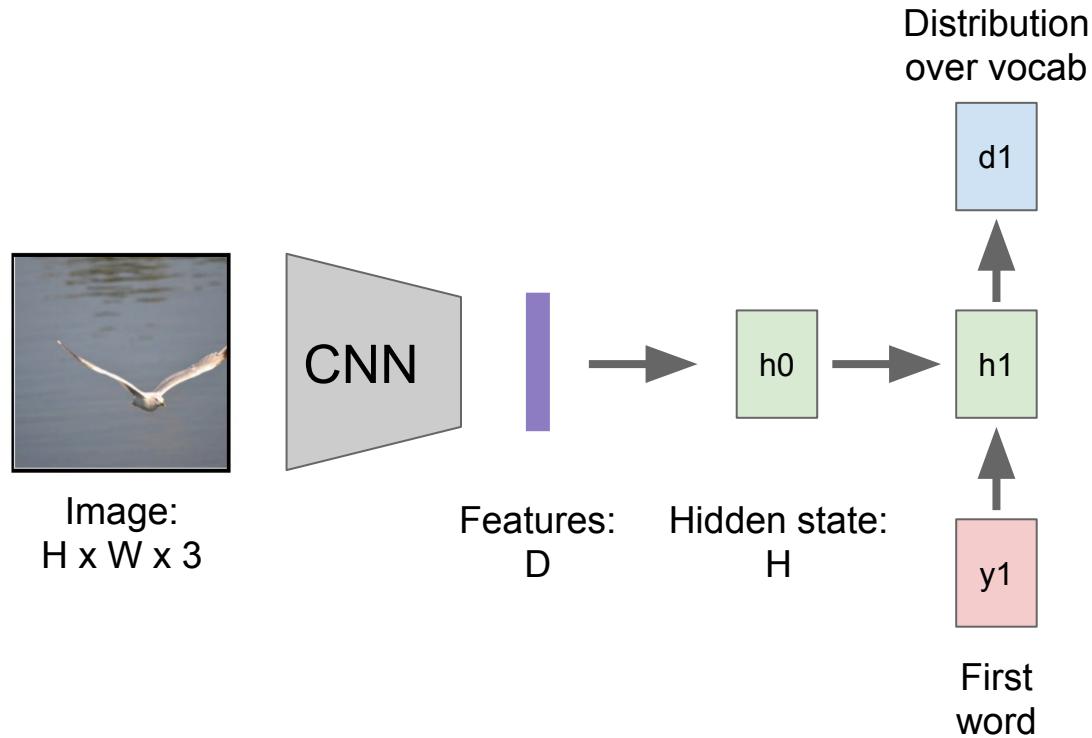
Features:  
 $D$



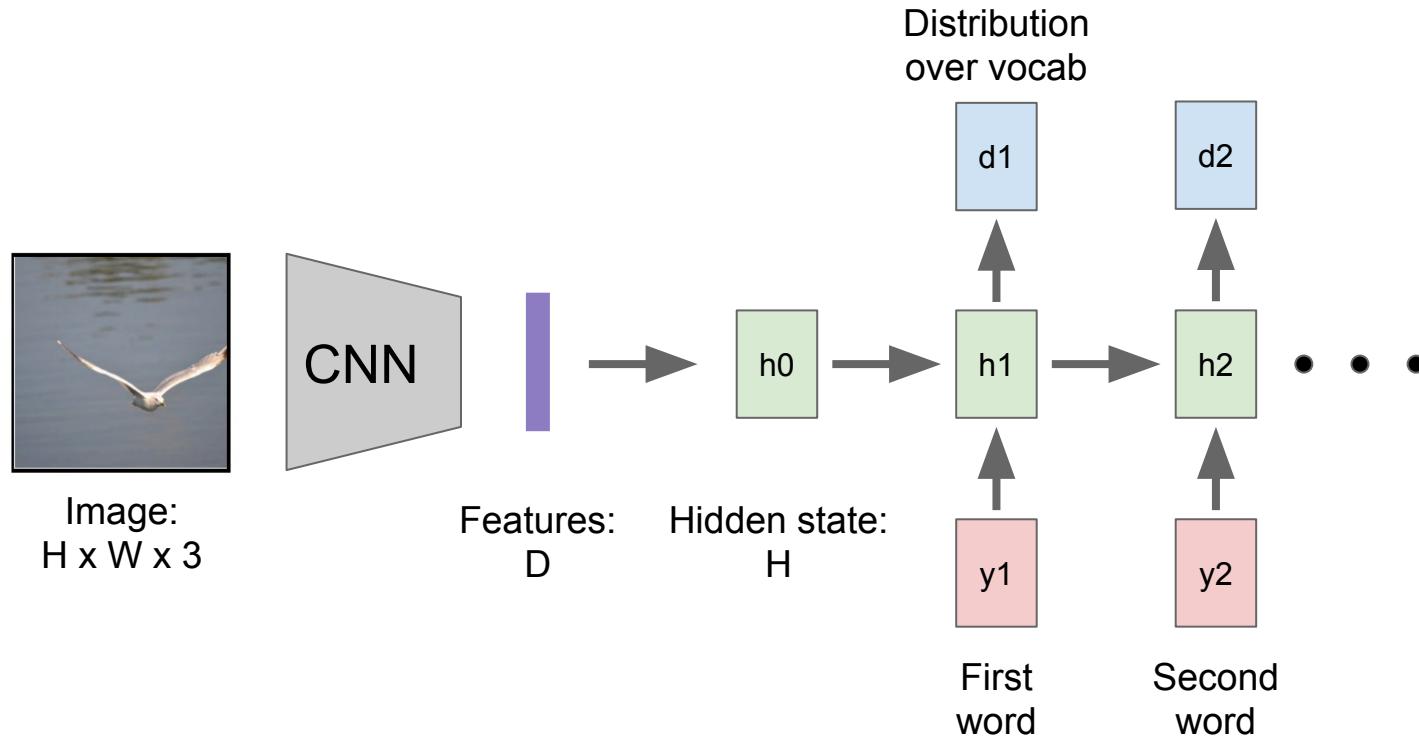
# Recall: RNN for Captioning



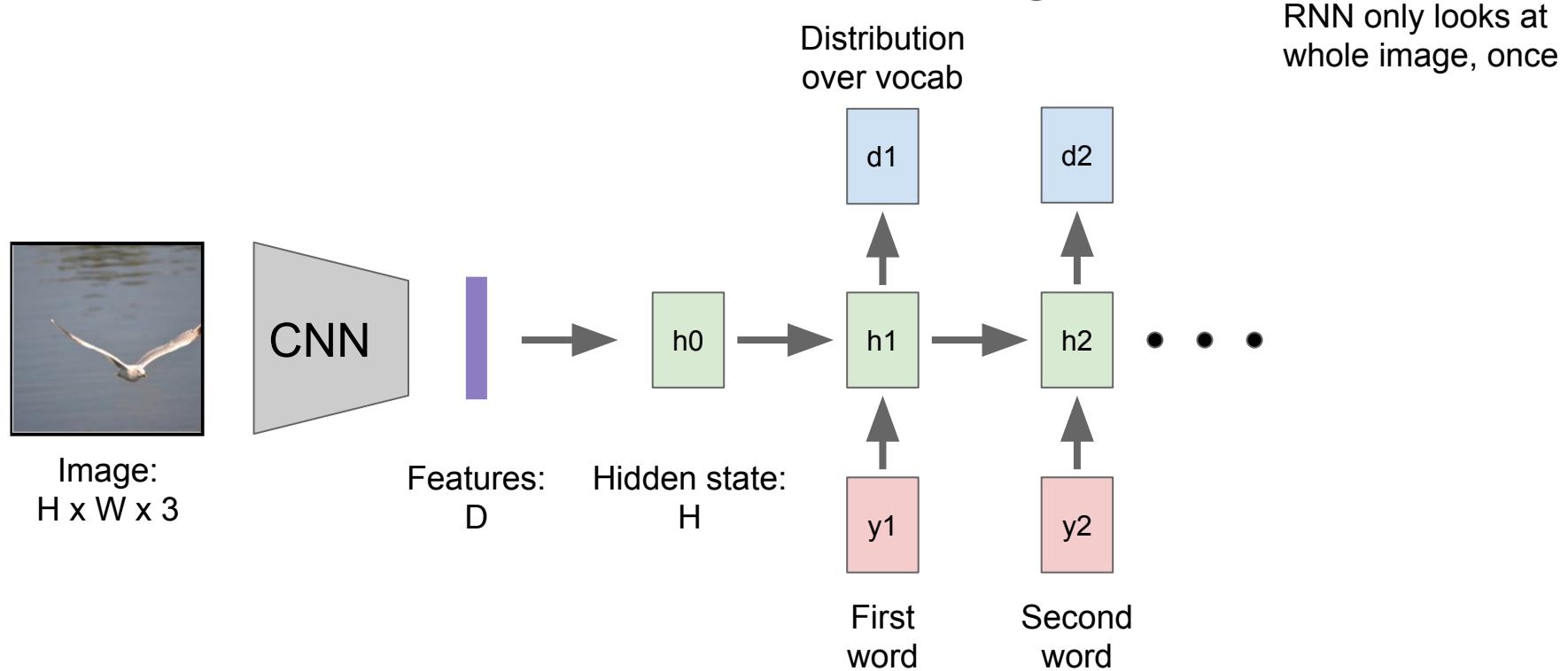
# Recall: RNN for Captioning



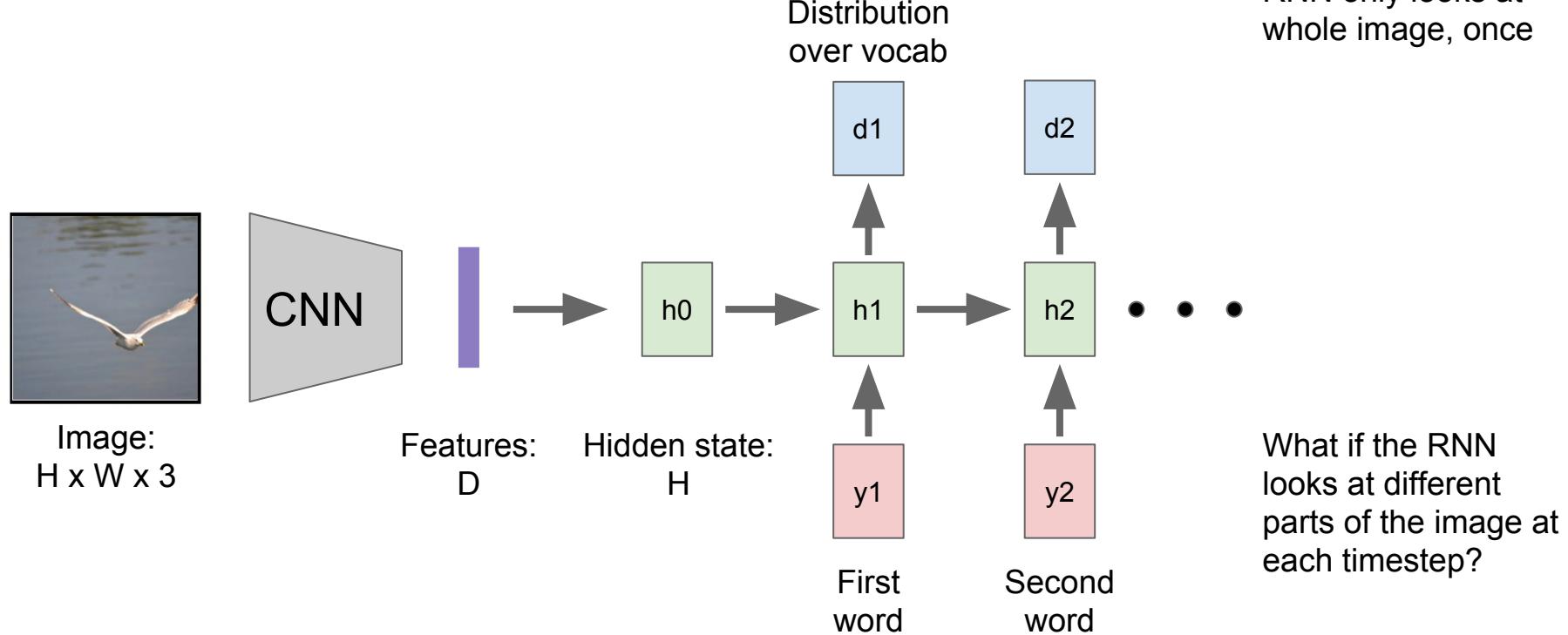
# Recall: RNN for Captioning



# Recall: RNN for Captioning



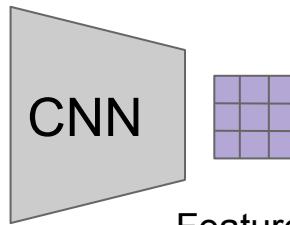
# Recall: RNN for Captioning



# Soft Attention for Captioning



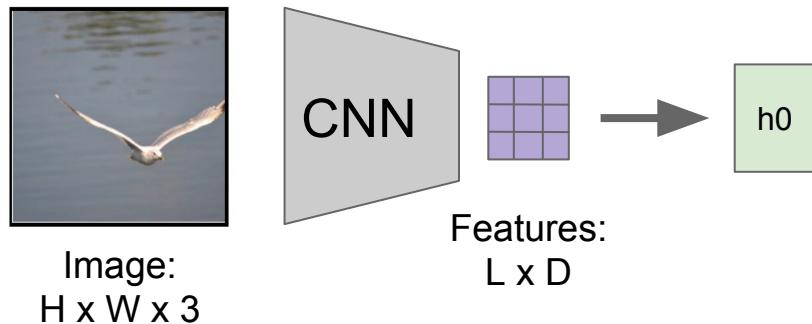
Image:  
 $H \times W \times 3$



Features:  
 $L \times D$

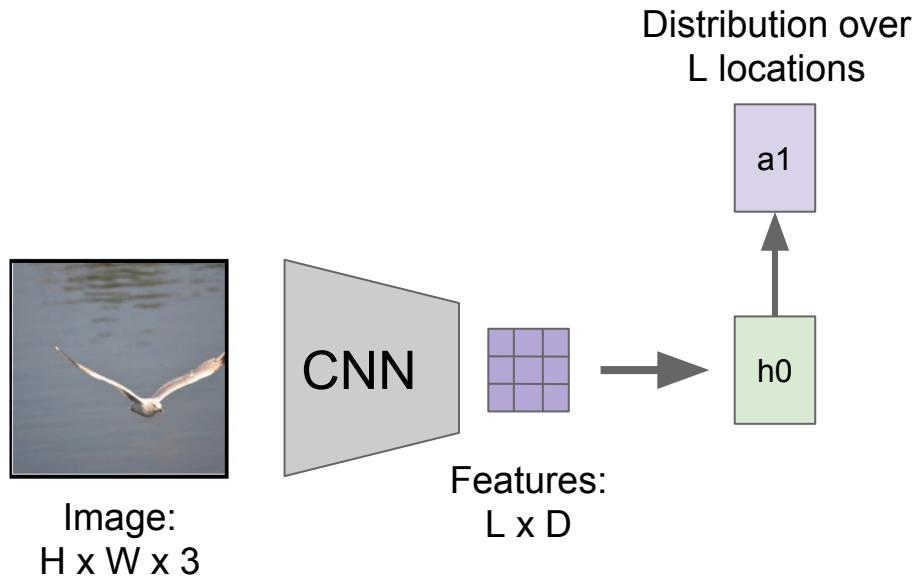
Xu et al, "Show, Attend and Tell: Neural  
Image Caption Generation with Visual  
Attention", ICML 2015

# Soft Attention for Captioning



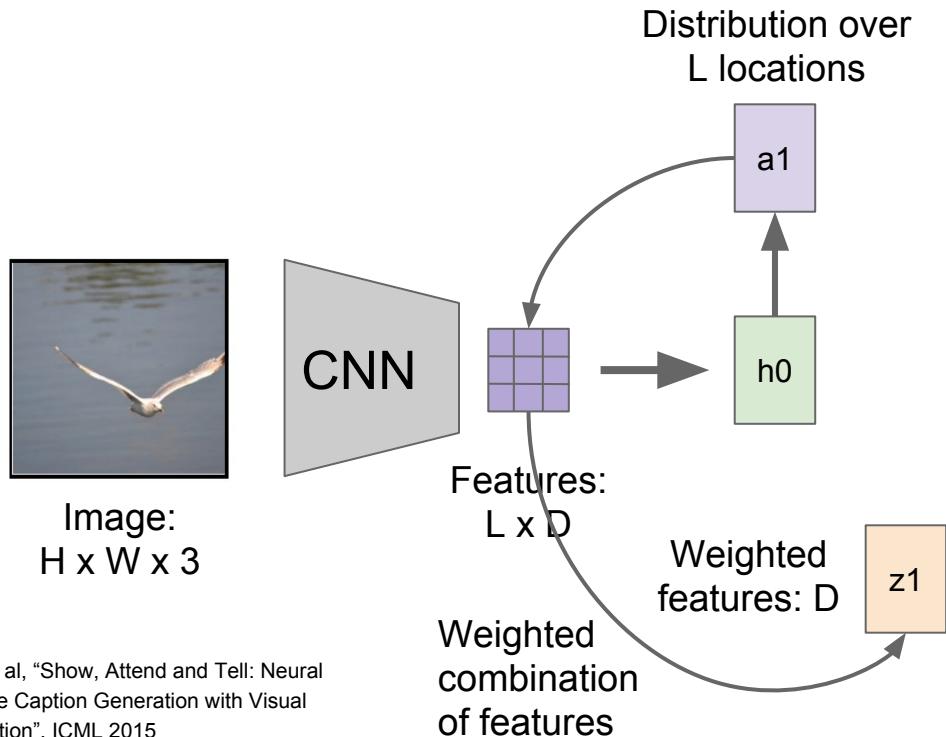
Xu et al, "Show, Attend and Tell: Neural  
Image Caption Generation with Visual  
Attention", ICML 2015

# Soft Attention for Captioning



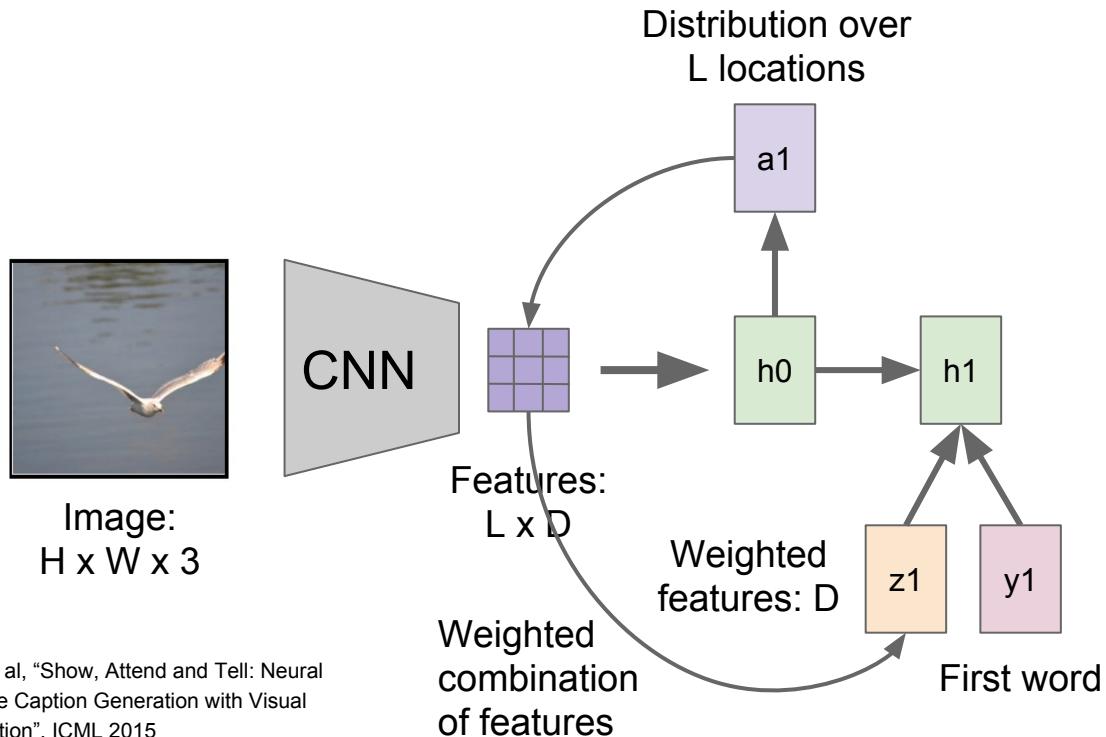
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Soft Attention for Captioning



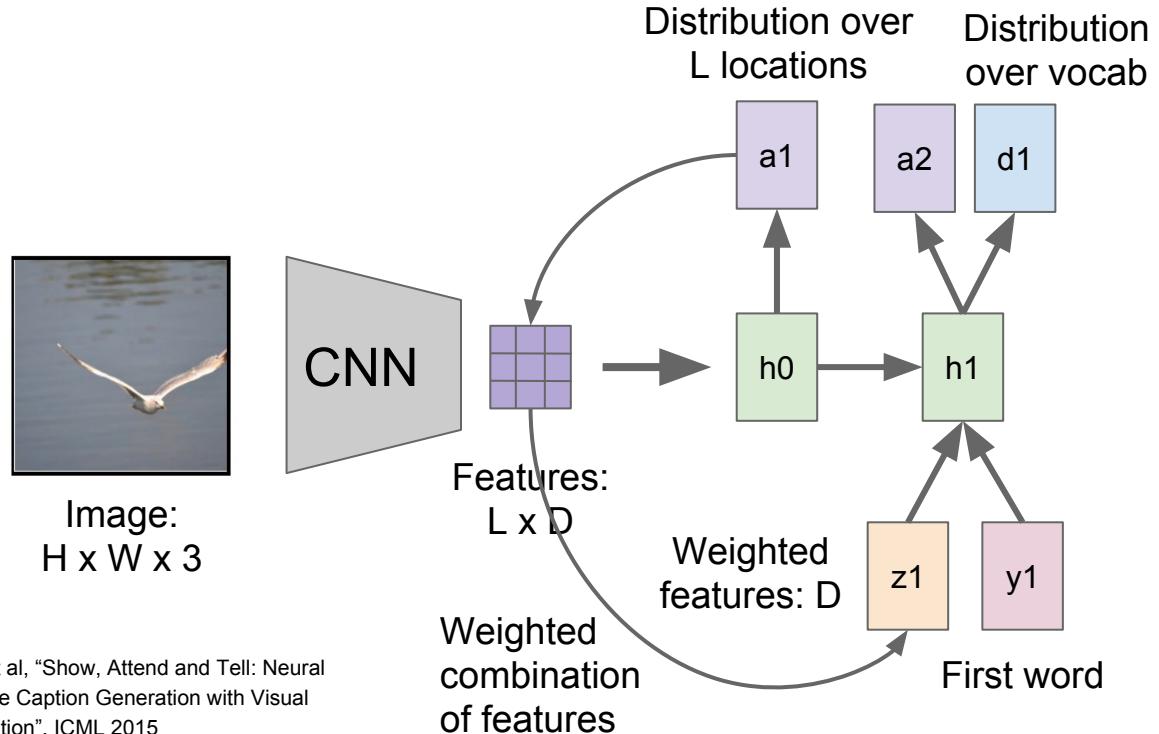
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Soft Attention for Captioning

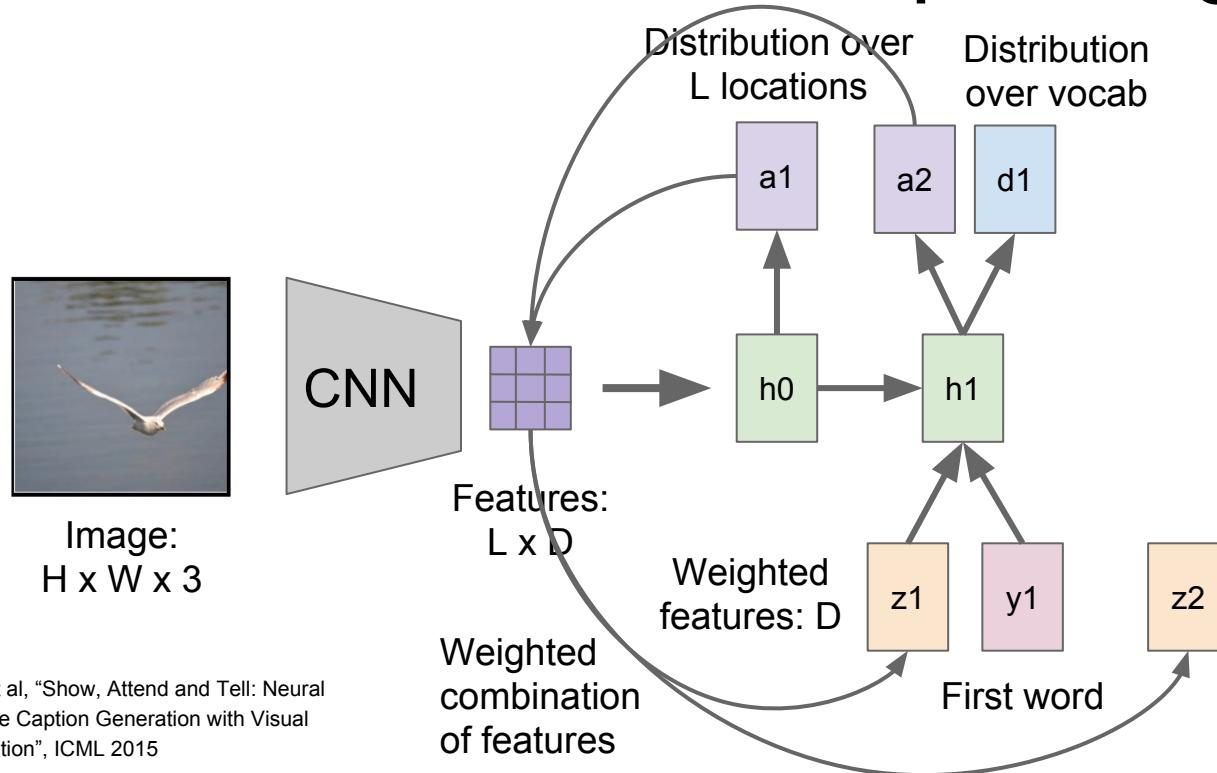


Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Soft Attention for Captioning

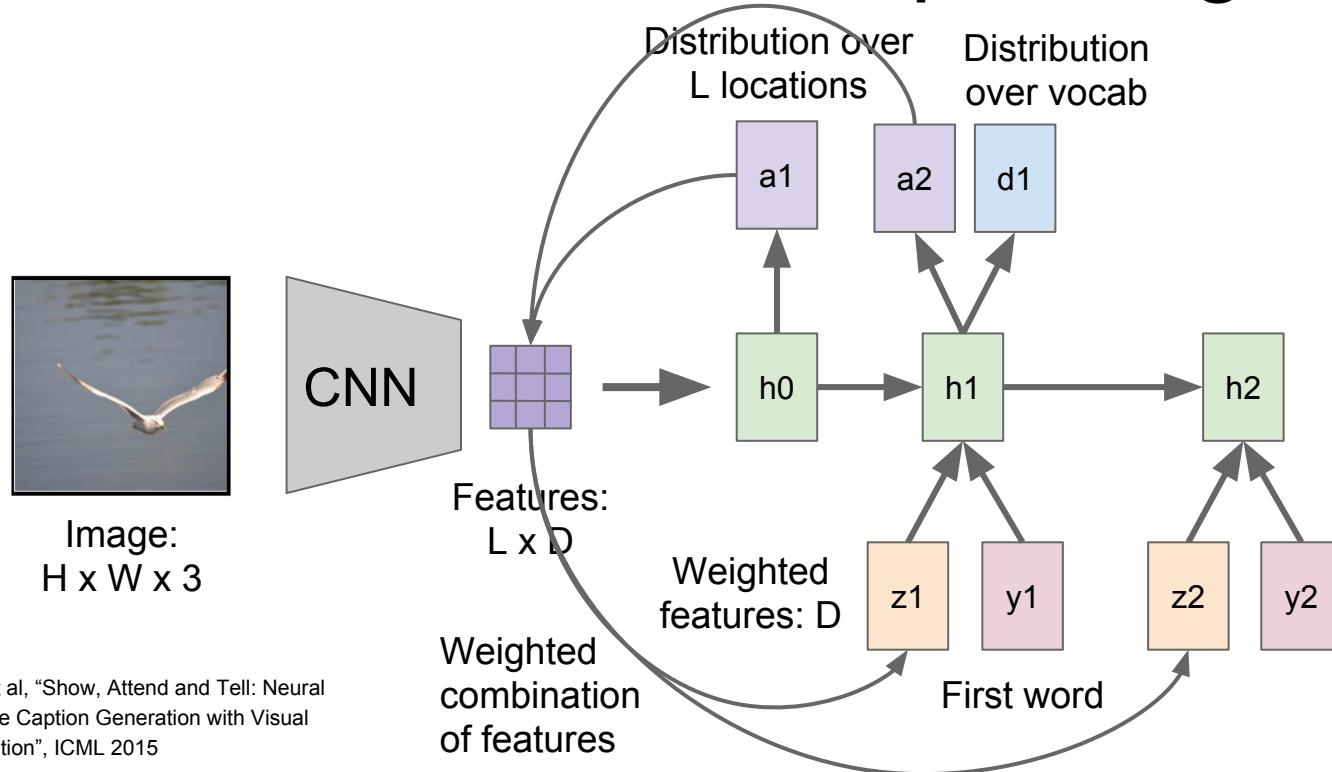


# Soft Attention for Captioning



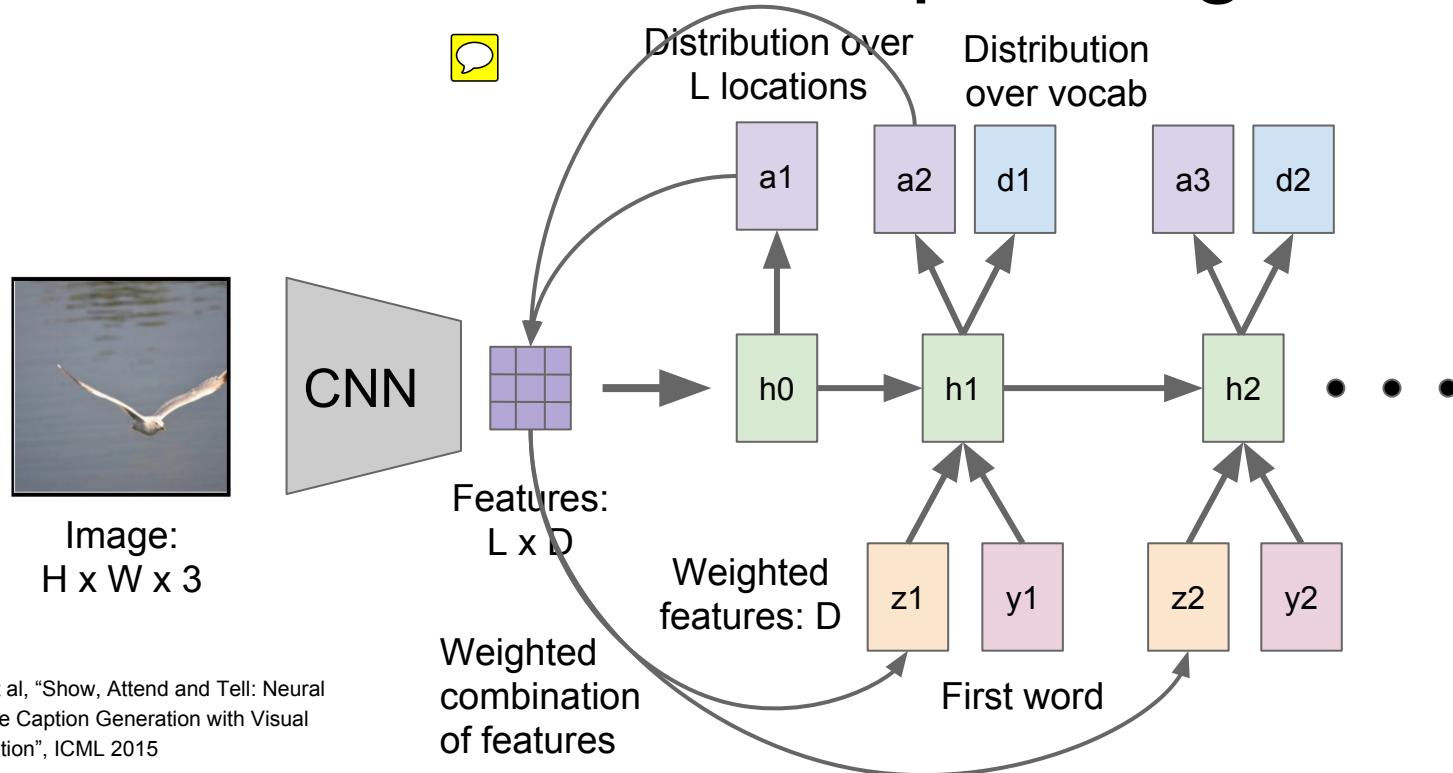
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Soft Attention for Captioning



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

# Soft Attention for Captioning



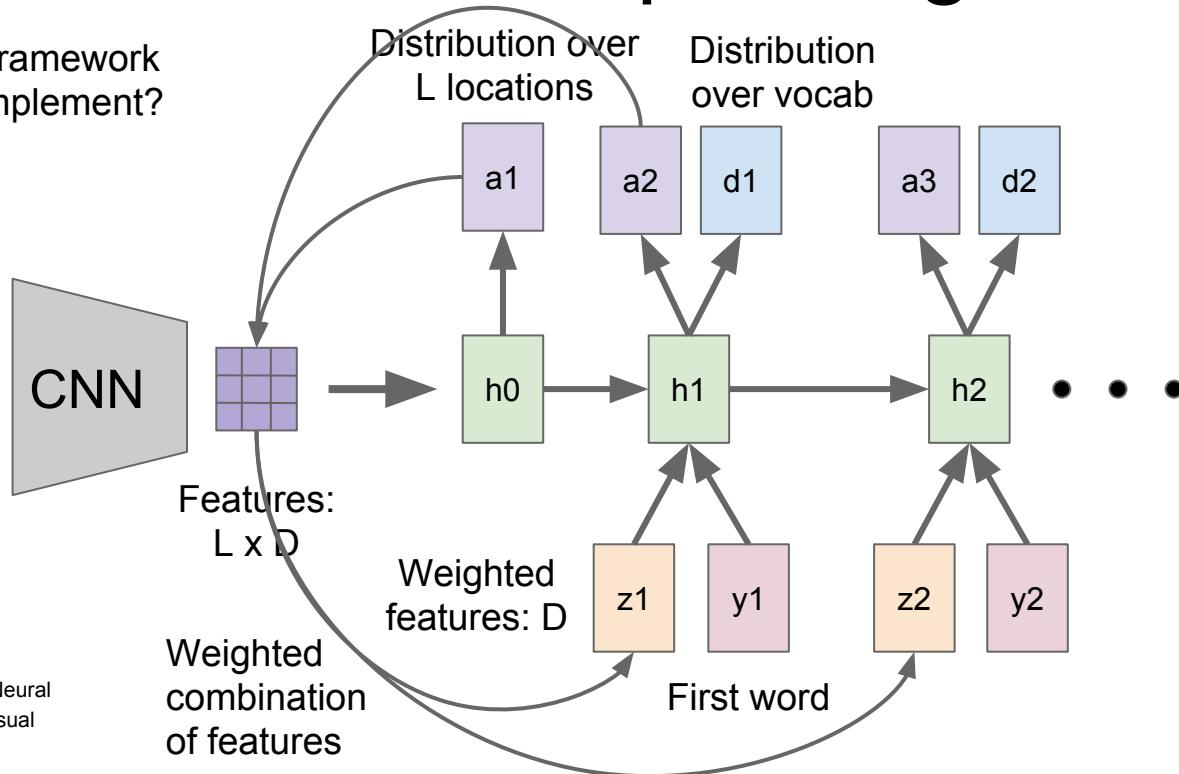
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Soft Attention for Captioning

Guess which framework was used to implement?



Image:  
 $H \times W \times 3$



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

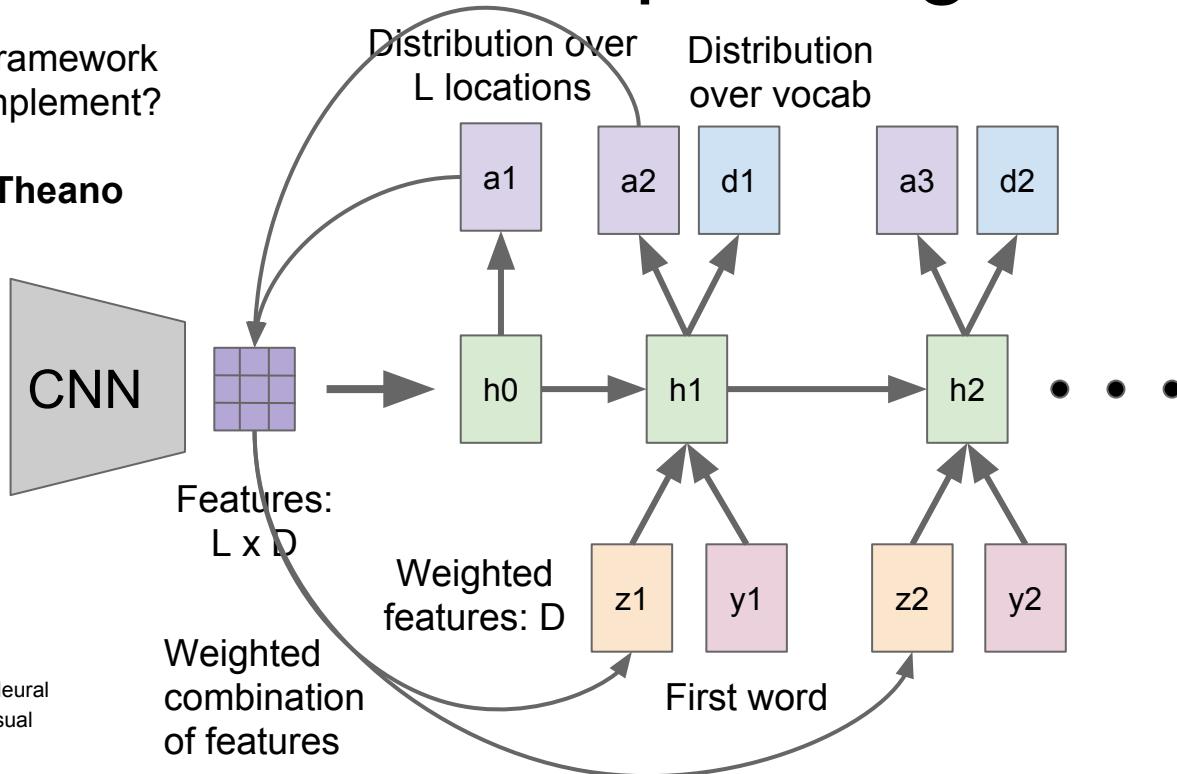
# Soft Attention for Captioning

Guess which framework  
was used to implement?

Crazy RNN = Theano



Image:  
 $H \times W \times 3$

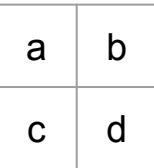
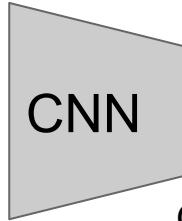


Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Soft vs Hard Attention

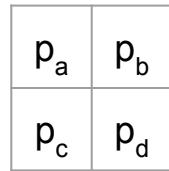


Image:  
 $H \times W \times 3$



Grid of features  
(Each D-dimensional)

From  
RNN:



Distribution over  
grid locations

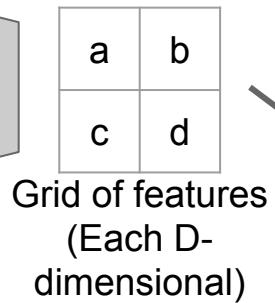
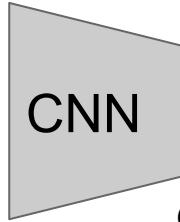
$$p_a + p_b + p_c + p_d = 1$$

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

# Soft vs Hard Attention

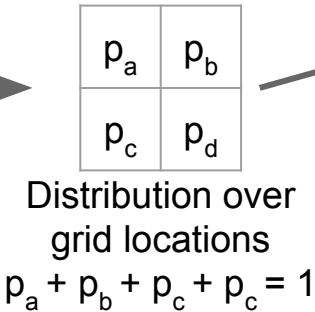


Image:  
 $H \times W \times 3$



Context vector  $z$   
(D-dimensional)

From  
RNN:

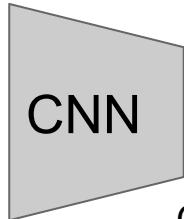


Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Soft vs Hard Attention



Image:  
 $H \times W \times 3$



Grid of features  
(Each D-dimensional)

$p_a$	$p_b$
$p_c$	$p_d$

Distribution over grid locations

$$p_a + p_b + p_c + p_d = 1$$

From RNN:

Context vector  $z$   
(D-dimensional)

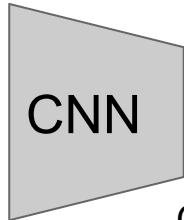
**Soft attention:**  
Summarize ALL locations  
$$z = p_a a + p_b b + p_c c + p_d d$$
  
Derivative  $dz/dp$  is nice!  
Train with gradient descent

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

# Soft vs Hard Attention



Image:  
 $H \times W \times 3$



Grid of features  
(Each D-dimensional)

From RNN:

$\xrightarrow{\hspace{1cm}}$

$p_a$	$p_b$
$p_c$	$p_d$

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

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

Soft attention



Hard attention



A

bird

flying

over

a

body

of

water

.

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.

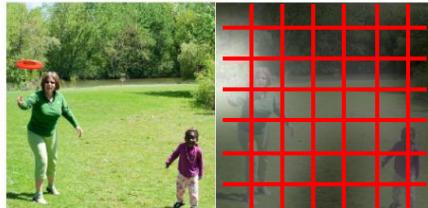


A giraffe standing in a forest with trees in the background.

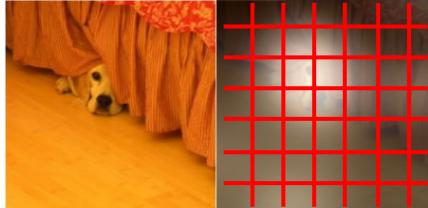
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Soft Attention for Captioning

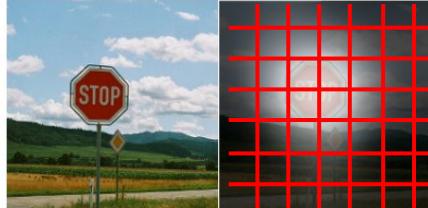
Attention constrained to fixed grid! We'll come back to this ....



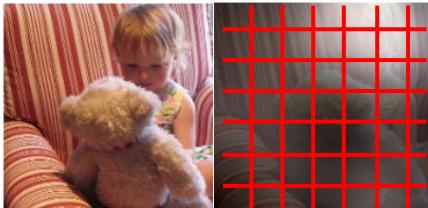
A woman is throwing a frisbee in a park.



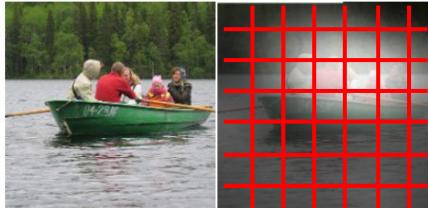
A dog is standing on a hardwood floor.



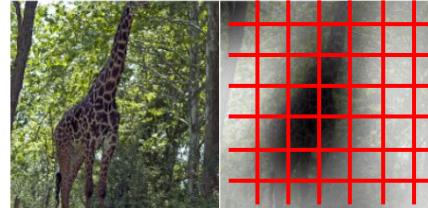
A stop sign is on a road with a mountain in the background.



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



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



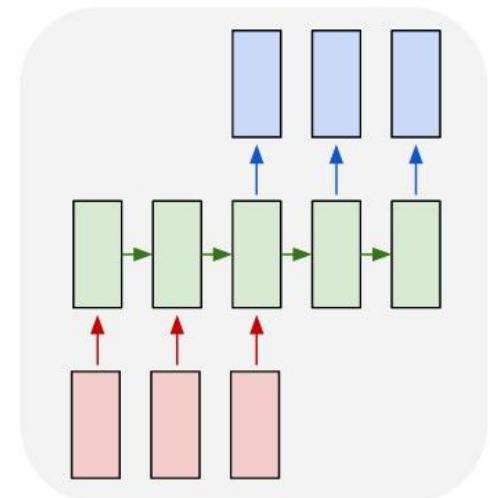
A giraffe standing in a forest with trees in the background.

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

# Soft Attention for Translation

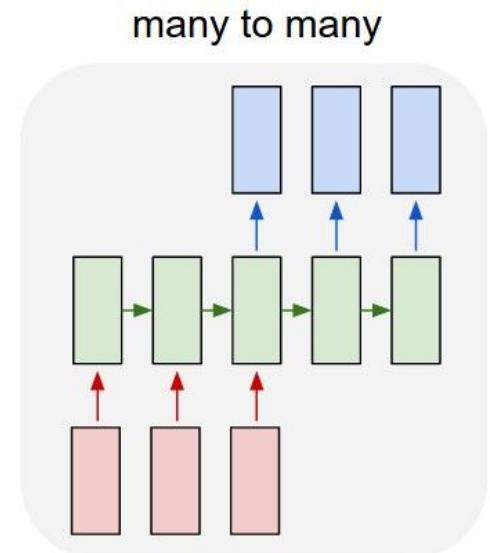
“Mi gato es el mejor” -> “My cat is the best”

many to many



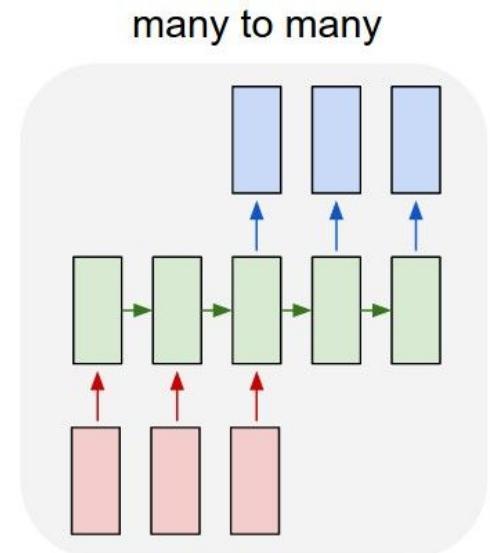
Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015

# Soft Attention for Translation



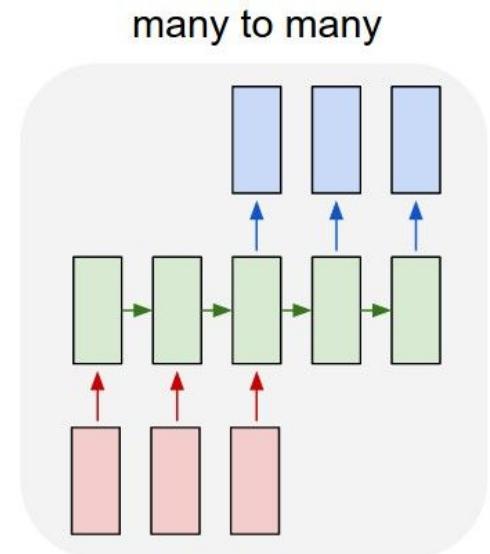
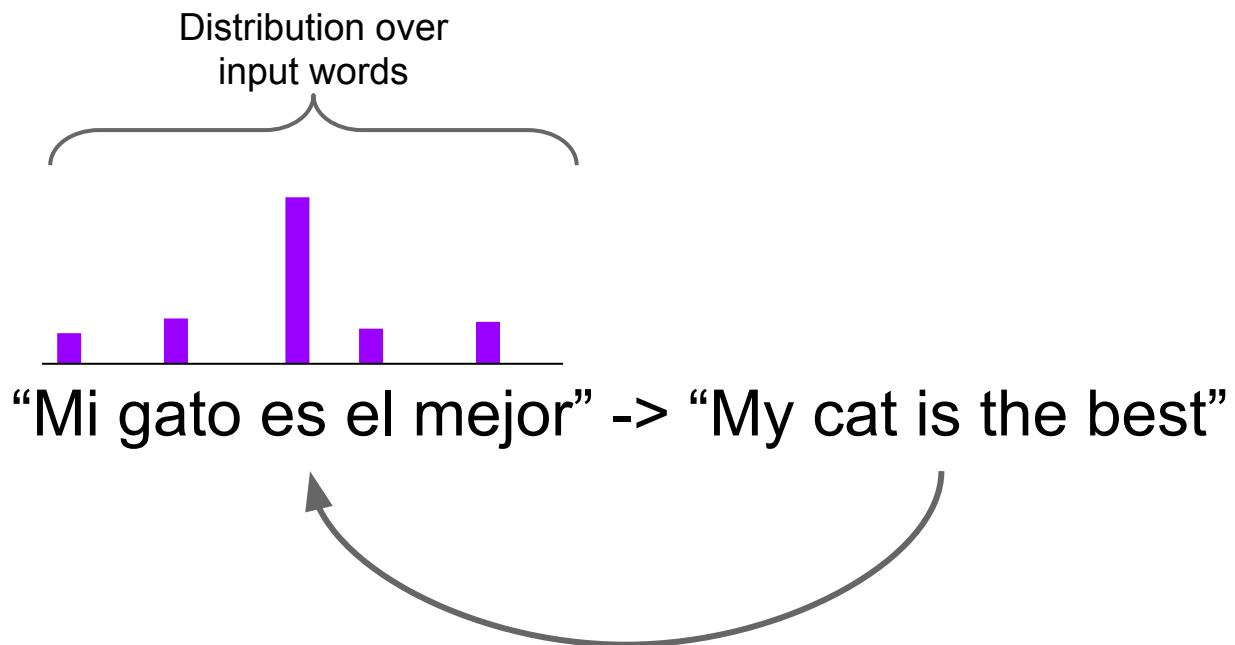
Bahdanau et al, "Neural Machine Translation by Jointly Learning to Align and Translate", ICLR 2015

# Soft Attention for Translation



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

What season does this appear to be?  
GT: fall Our Model: fall



What is soaring in the sky?  
GT: kite Our Model: kite



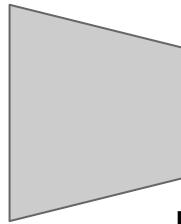
## Image, question to answer, attention over image:

- 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

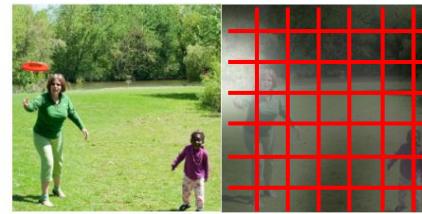
# Attending to arbitrary regions?



Image:  
 $H \times W \times 3$



Features:  
 $L \times D$

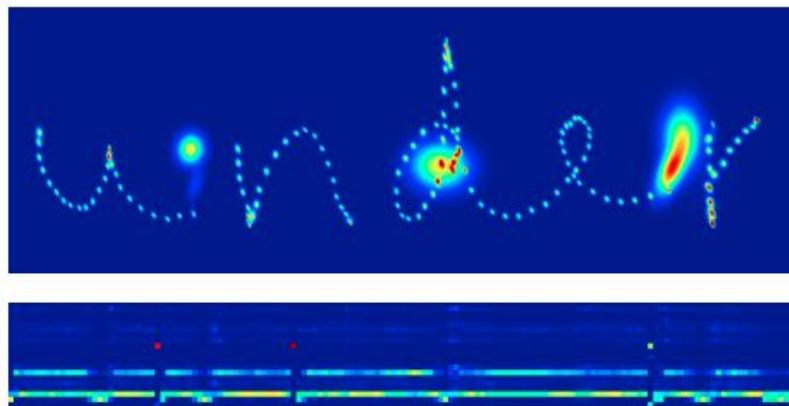


A woman is throwing a frisbee in a park.

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

# Attending to Arbitrary Regions

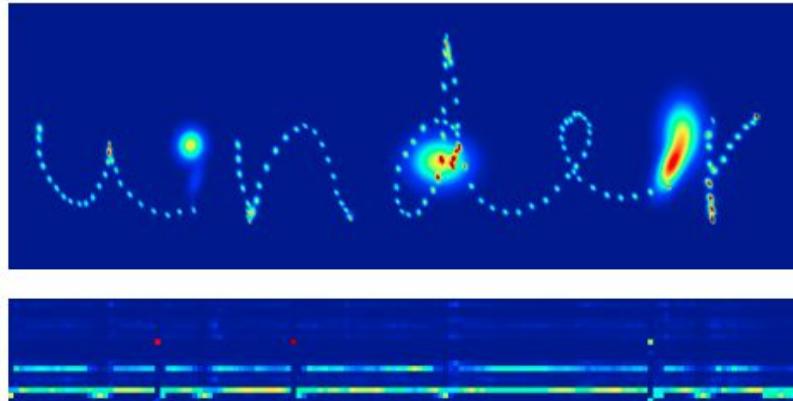
- Read text, generate handwriting using an RNN
- Attend to arbitrary regions of the **output** by predicting params of a mixture model



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



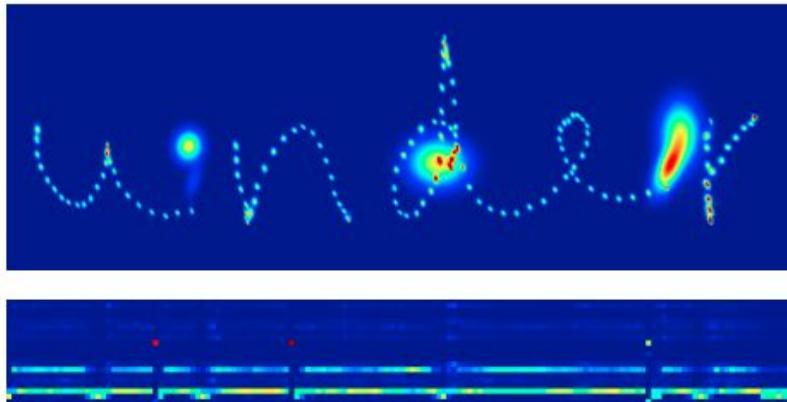
Which are real and which are generated?

more of national temperament  
more of national temperament

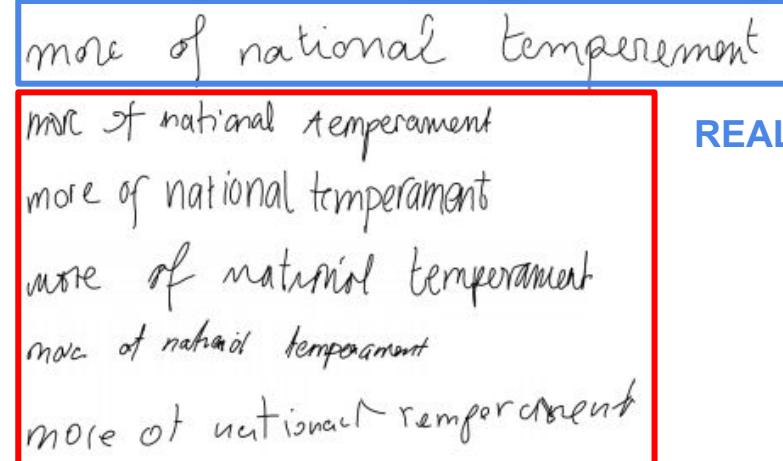
Graves, "Generating Sequences with Recurrent Neural Networks", arXiv 2013

# 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?



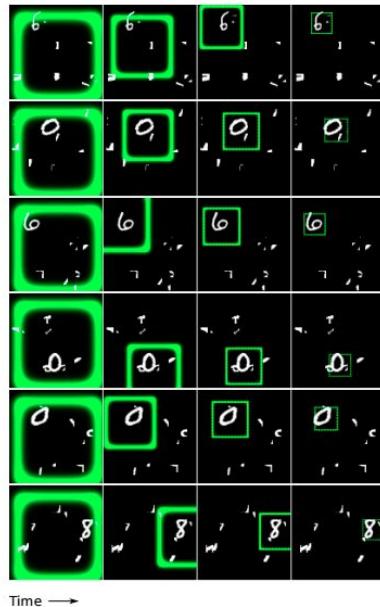
REAL

GENERATED

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

# Attending to Arbitrary Regions: DRAW

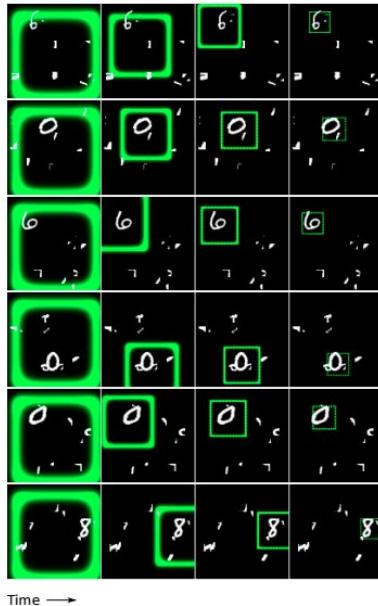
Classify images by attending to arbitrary regions of the *input*



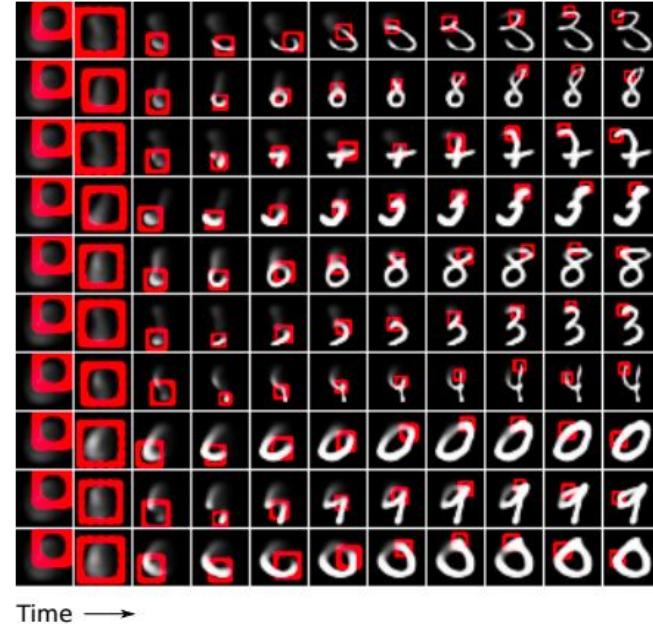
Gregor et al., "DRAW: A Recurrent Neural Network For Image Generation", ICML 2015

# Attending to Arbitrary Regions: DRAW

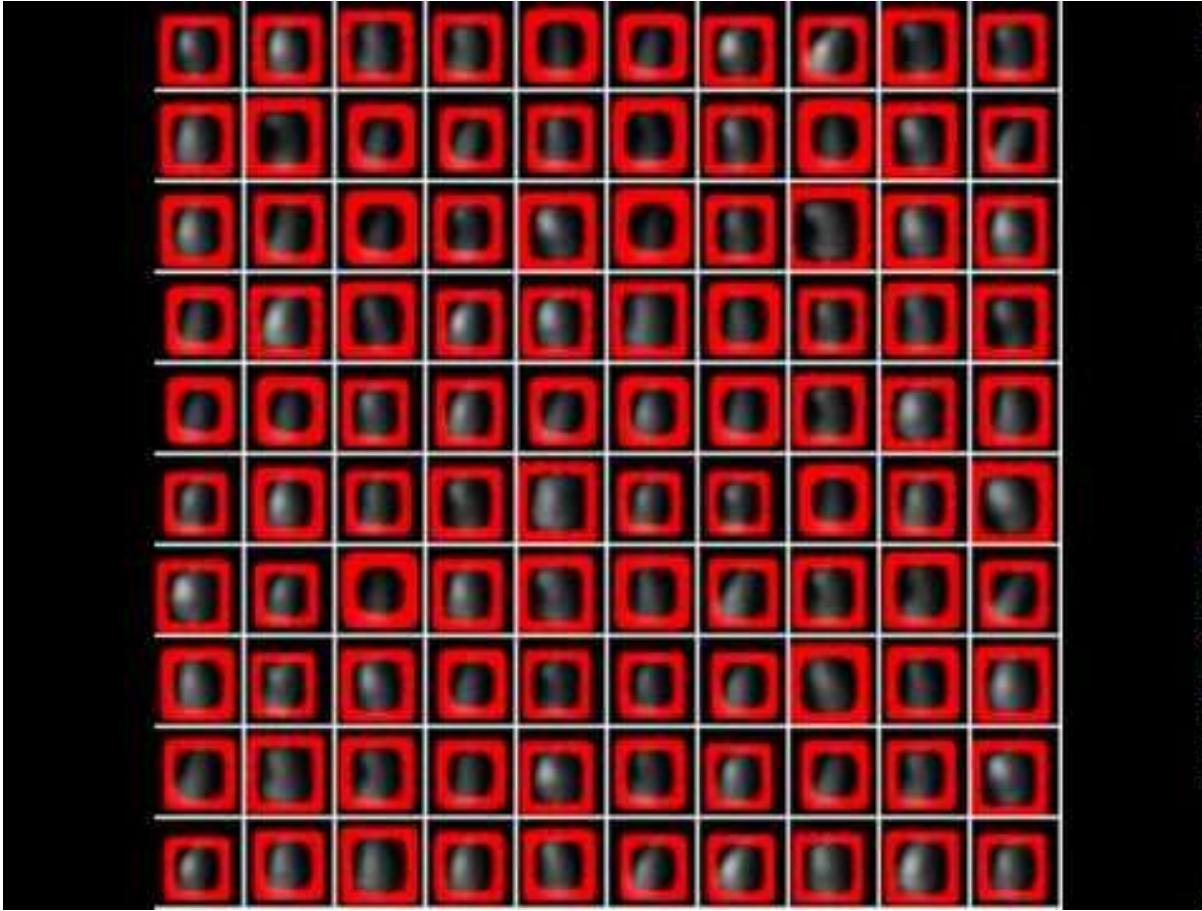
**Classify** images by attending to arbitrary regions of the *input*



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



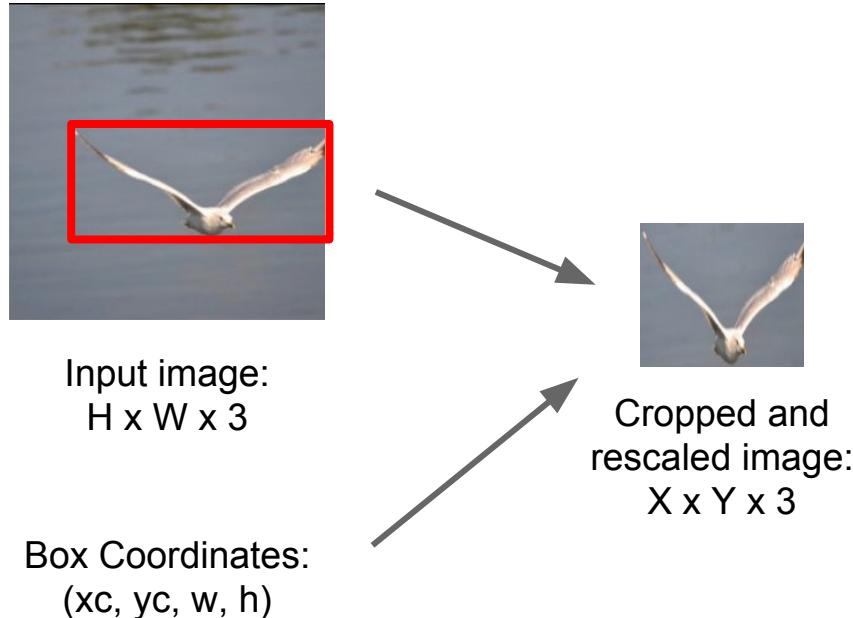
Gregor et al., "DRAW: A Recurrent Neural Network For Image Generation", ICML 2015



# Attending to Arbitrary Regions: Spatial Transformer Networks

Attention mechanism similar to DRAW, but easier to explain

# Spatial Transformer Networks



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

# Spatial Transformer Networks



Input image:  
 $H \times W \times 3$

Box Coordinates:  
 $(x_c, y_c, w, h)$

Can we make this  
function differentiable?



Cropped and  
rescaled image:  
 $X \times Y \times 3$

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

# Spatial Transformer Networks



Input image:  
 $H \times W \times 3$

Box Coordinates:  
 $(x_c, y_c, w, h)$

Can we make this  
function differentiable?

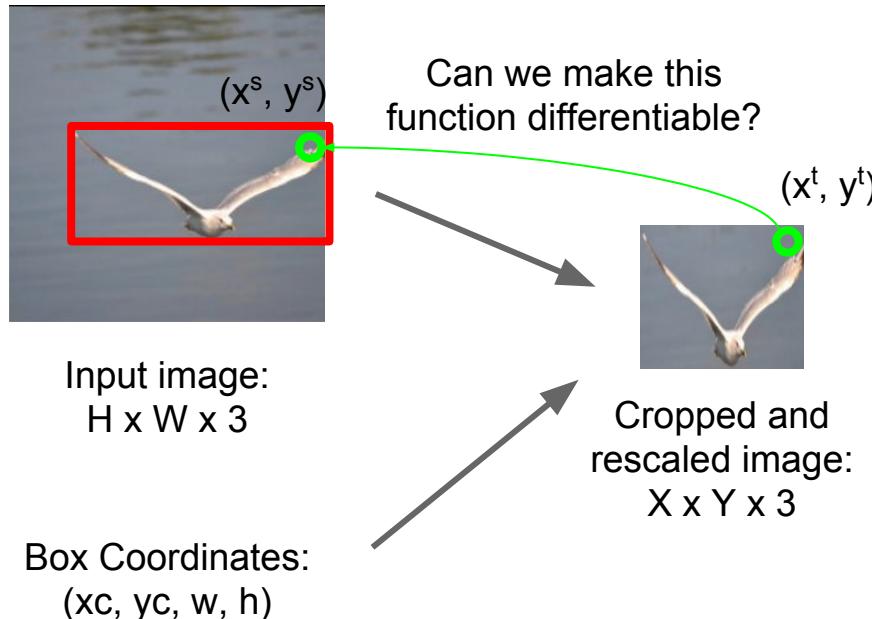


Cropped and  
rescaled image:  
 $X \times Y \times 3$

**Idea:** Function mapping  
*pixel coordinates* ( $x_t, y_t$ ) of  
output to *pixel coordinates*  
( $x_s, y_s$ ) of input

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

# Spatial Transformer Networks

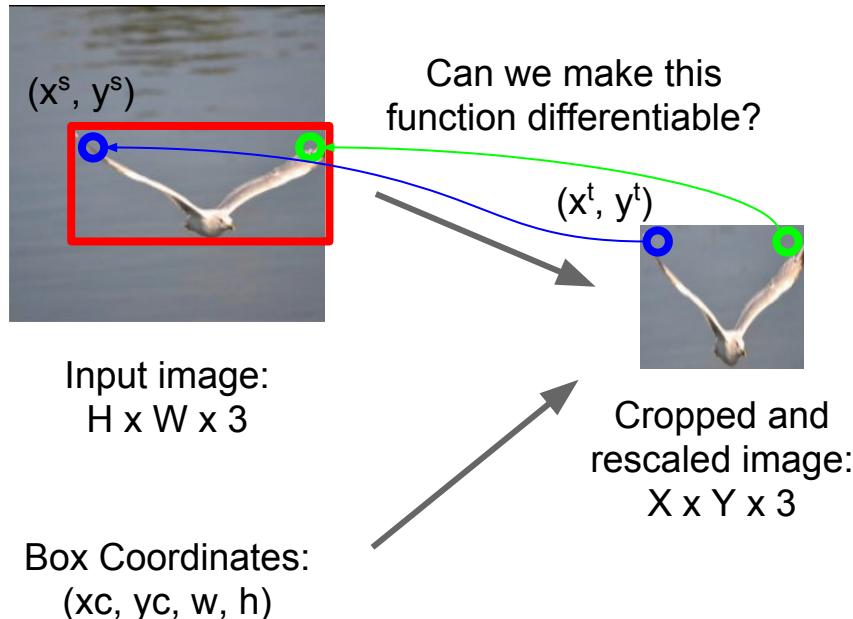


**Idea:** Function mapping *pixel coordinates*  $(x_t, y_t)$  of output to *pixel coordinates*  $(x_s, y_s)$  of input

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

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

# Spatial Transformer Networks

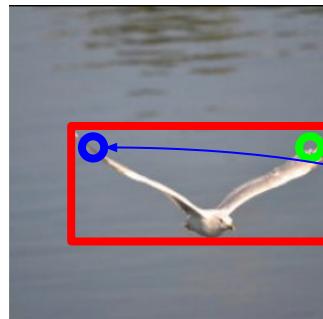


**Idea:** Function mapping *pixel coordinates*  $(x_t, y_t)$  of output to *pixel coordinates*  $(x_s, y_s)$  of input

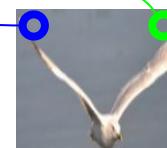
$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

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

# Spatial Transformer Networks



Can we make this function differentiable?



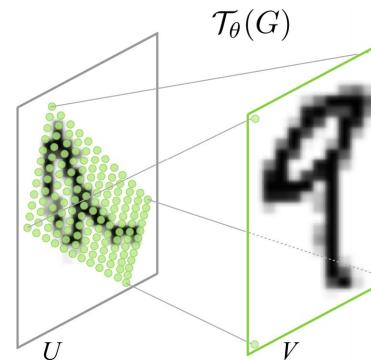
Input image:  
 $H \times W \times 3$

Box Coordinates:  
 $(x_c, y_c, w, h)$

Cropped and  
rescaled image:  
 $X \times Y \times 3$

**Idea:** Function mapping  
*pixel coordinates* ( $x_t, y_t$ ) of  
output to *pixel coordinates*  
( $x_s, y_s$ ) of input

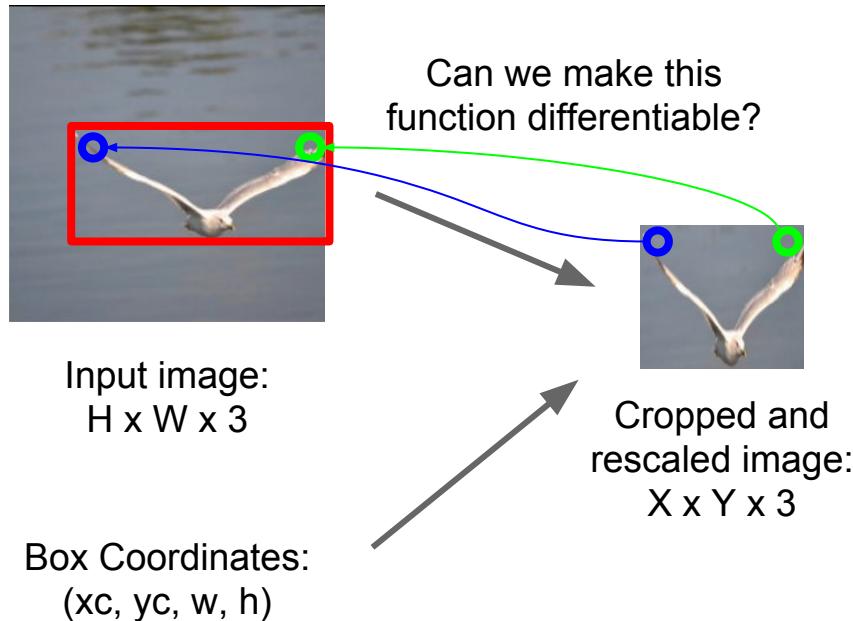
$$\begin{pmatrix} x_i^s \\ y_i^s \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$



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

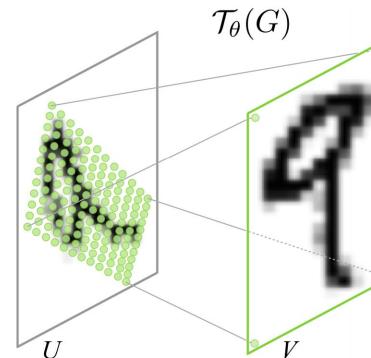
Jaderberg et al, "Spatial Transformer Networks", NIPS 2015

# Spatial Transformer Networks



**Idea:** Function mapping *pixel coordinates* ( $x_t, y_t$ ) of output to *pixel coordinates* ( $x_s, y_s$ ) of input

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

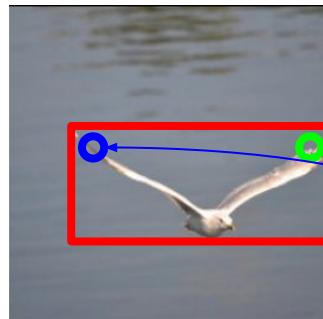


Repeat for all pixels in *output* to get a **sampling grid**

Then use **bilinear interpolation** to compute output

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

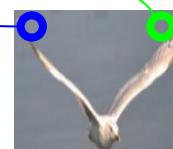
# Spatial Transformer Networks



Can we make this function differentiable?

Input image:  
 $H \times W \times 3$

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

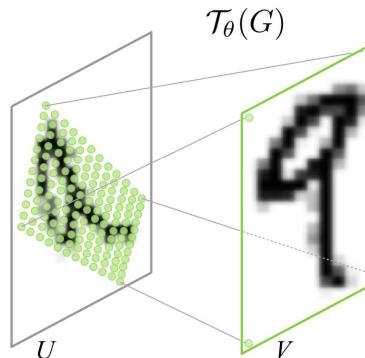


Cropped and  
rescaled image:  
 $X \times Y \times 3$

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

Network  
attends to  
input by  
predicting  $\theta$

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

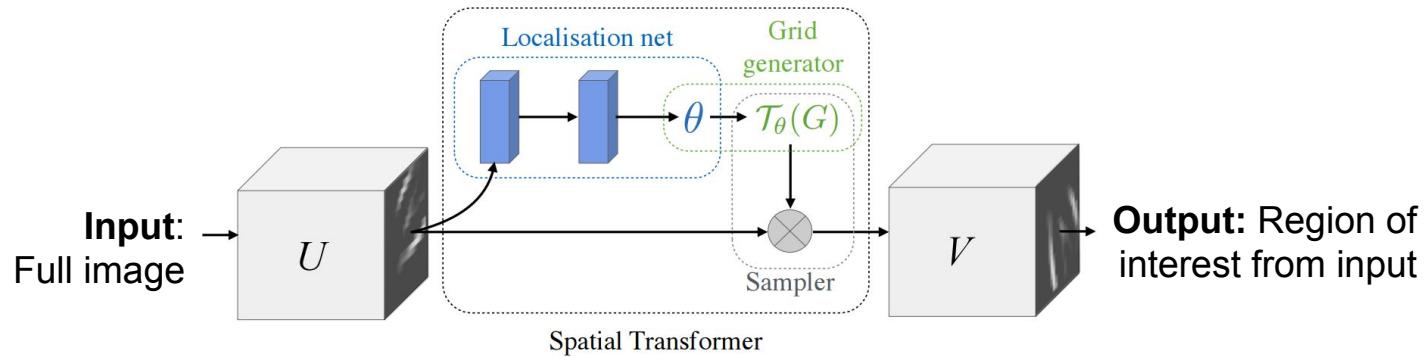


Repeat for all pixels  
in *output* to get a  
**sampling grid**

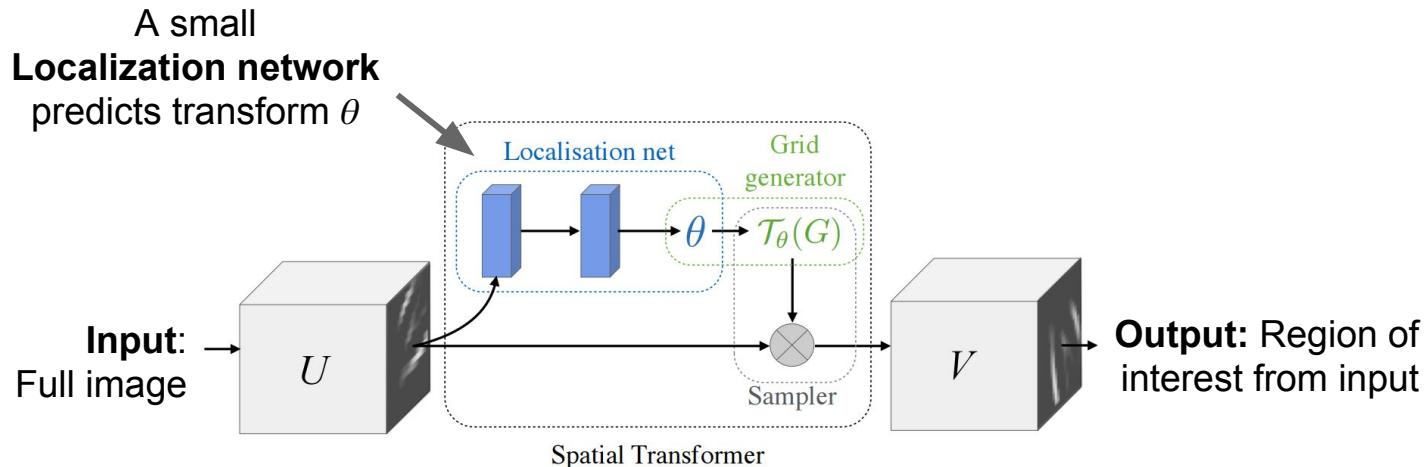
Then use **bilinear  
interpolation** to  
compute output

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

# Spatial Transformer Networks

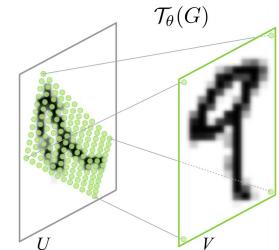
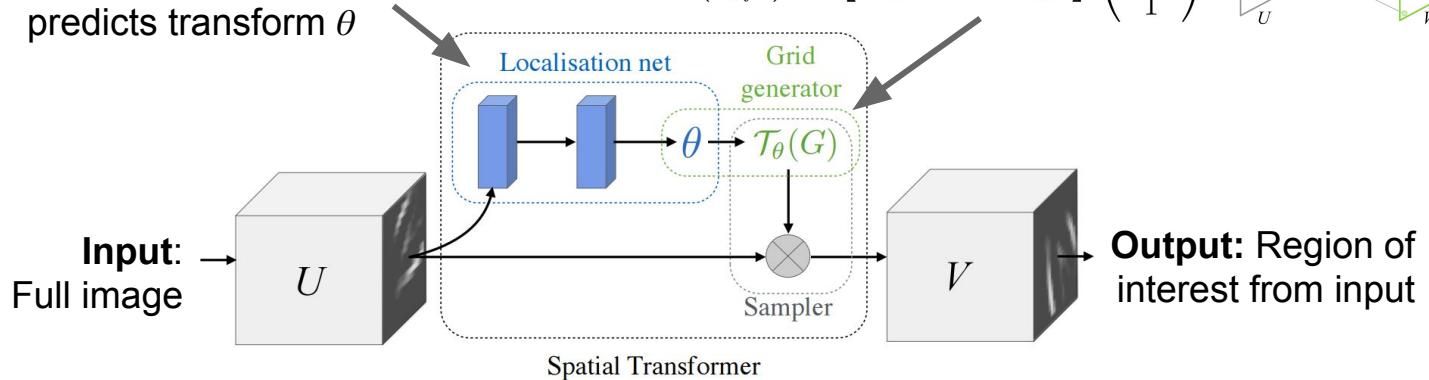


# Spatial Transformer Networks



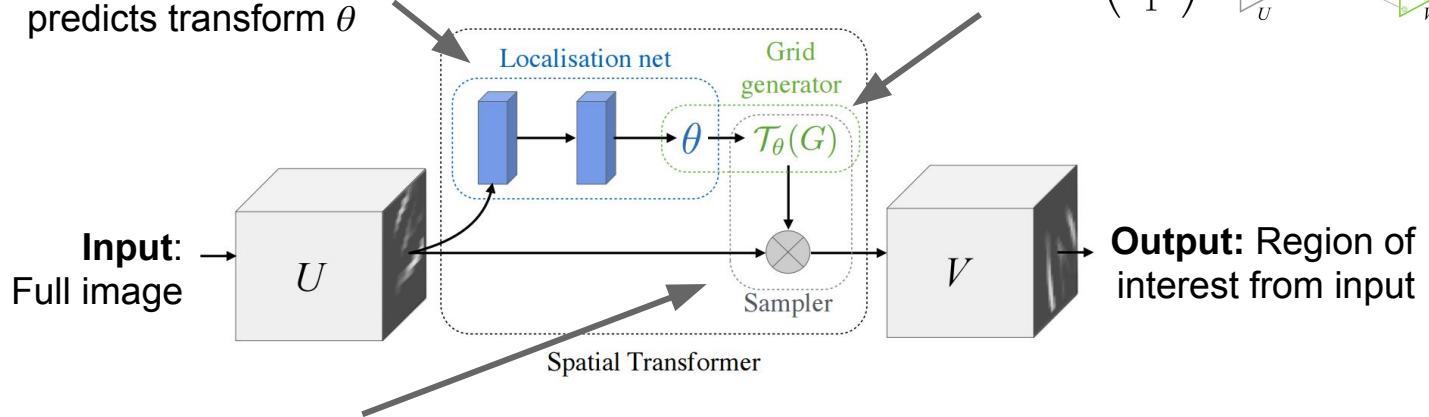
# Spatial Transformer Networks

A small  
**Localization network**  
predicts transform  $\theta$

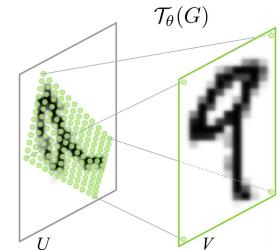


# Spatial Transformer Networks

A small  
**Localization network**  
predicts transform  $\theta$

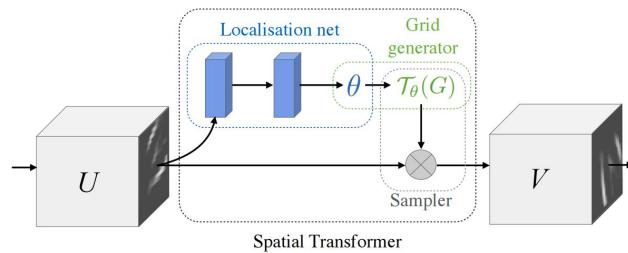


**Sampler** uses  
bilinear interpolation  
to produce output

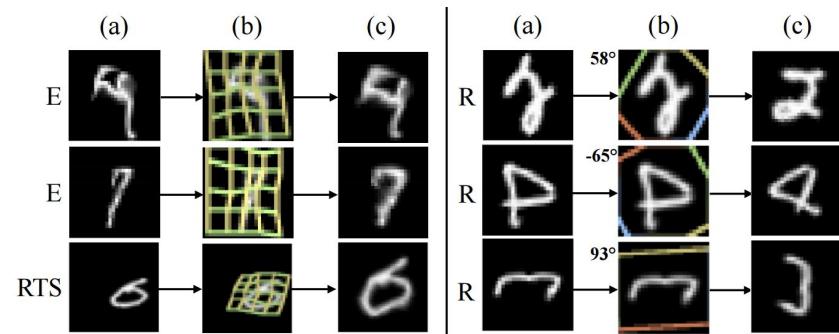


# Spatial Transformer Networks

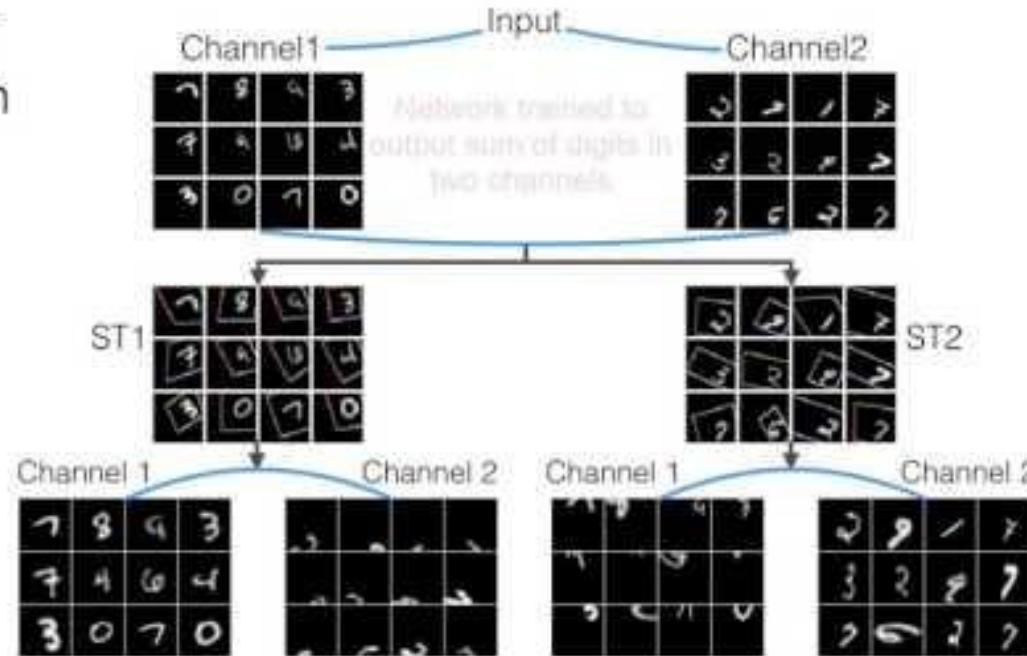
Differentiable “attention / transformation” module



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



## MNIST Addition



# 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!**