Deep Learning

Lecture 5 – Convolutional Neural Networks 2

UPPSALA
UNIVERSITET

Joakim Lindblad
Department of Information Technology
Uppsala University

joakim.lindblad@it.uu.se
cb.uu.se/~joakim

UPPSALA UNIVERSITET

Outline

Recap

Deep learning software frameworks

Going deeper

Semantic segmentation

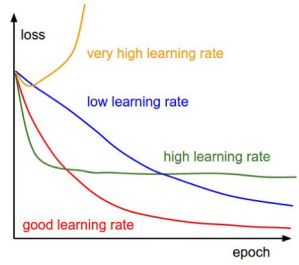
Localization, detection, instance segmentation

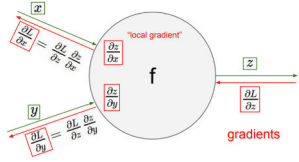
Concluding

Recap

Recap

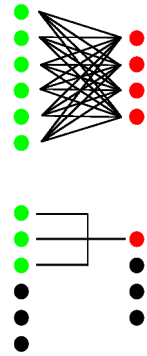
- Linear regression $y = Wx + b$
- Send the output through a nonlinearity (activation function) $y = f(Wx + b)$
- Send the output to another classifier, and another... $y = f(W^{(3)} f(W^{(2)} f(W^{(1)} x + b^{(1)}) + b^{(2)}) + b^{(3)})$
  – A *Feedforward Neural Network*
- Measure performance with a *loss function*
- Use Stochastic Gradient Descent (SGD) to minimize the loss
  – Take small steps in the direction of the negative gradient $\theta_{k+1} = \theta_k - \lambda \cdot \nabla_\theta L$, where stepsize (a.k.a. learning rate) λ typically in the range $[0.001, 0.01]$
- Need to compute partial derivatives of the Loss w.r.t. all model parameters. $\nabla_\theta L(\theta; \mathbf{x}) = (\frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_2}, \ldots)$
  – Back-propagation – recursive application of the chain rule to propagate the gradient backwards up the network.



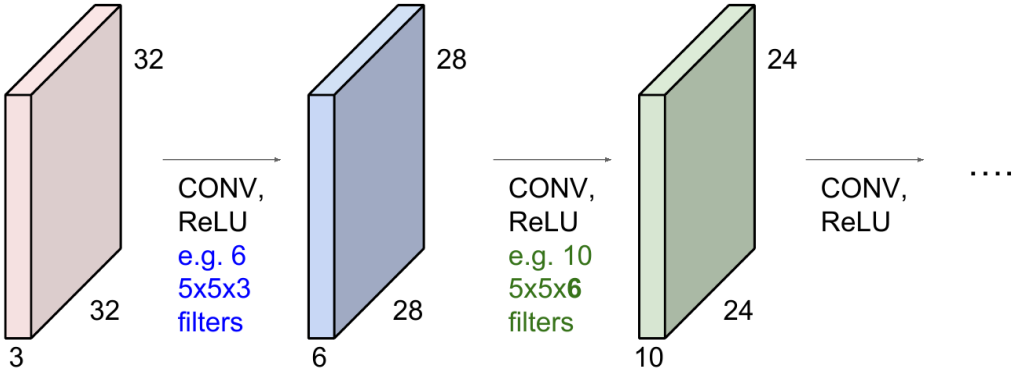


Recap – Convolutional neural networks

- Fully connected layer (fully Affine layer)
  - Connect every input with every output
  - Very many parameters, slow and difficult to train
- Convolutional layer
  - Learn a local filter kernel
  - Preserves spatial relationships
  - Huge reduction in number of parameters: fewer connections and parameter sharing
- Max pooling (downsampling) to shrink the spatial dimensions
  - Typically increase the depth (number of filters) instead



Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

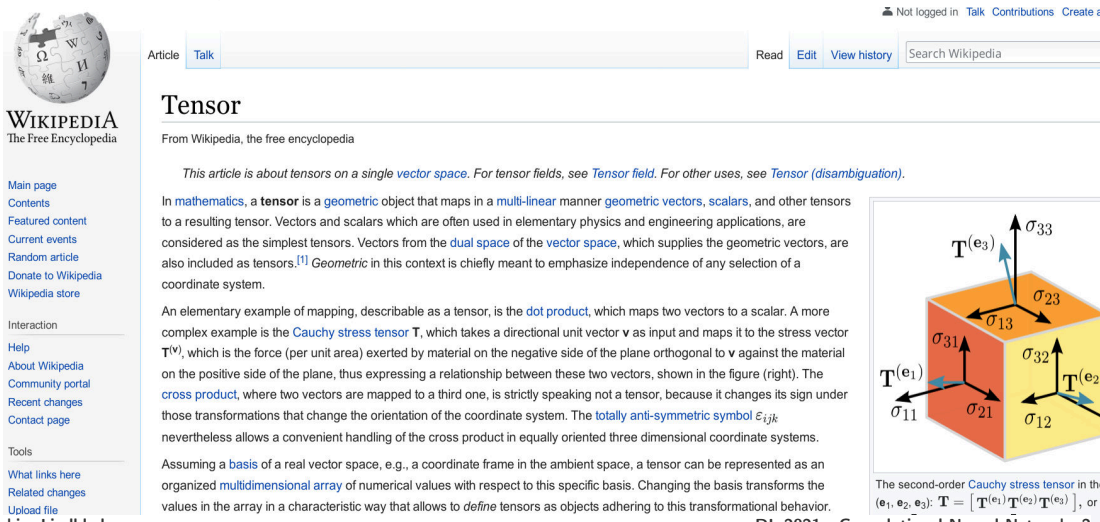


- Number of channels of the filter, almost always equal to the channels of the input
- Number of filters equals the number of channels of the output

Tensor

A (mathematical) tensor can be defined as a multidimensional array obeying certain transformation laws under the change of coordinates.

In machine learning / data processing a tensor is just a multidimensional numerical array



A few concepts to summarize lecture 4

Fully connected layer: Each output is connected to every input. (Input size plus one) $\times$ output size number of parameters.

Convolutional layer: Only local connections and weight sharing; sliding filter over the image. (Filter size plus one) $\times$ number of filters (depth) number of parameters.

Pooling layer: Aggregation of local information plus downsampling (striding). Most often max-pooling, but sometimes average pooling. No parameters.

Striding: When only computing output for every n-th pixel/voxel, reducing the spatial size of the activation. For example strided convolution.

Padding: Artificially extending the input data around the borders; typically to fit a convolution kernel (if odd sized kernel, then padding=(size-1)/2).

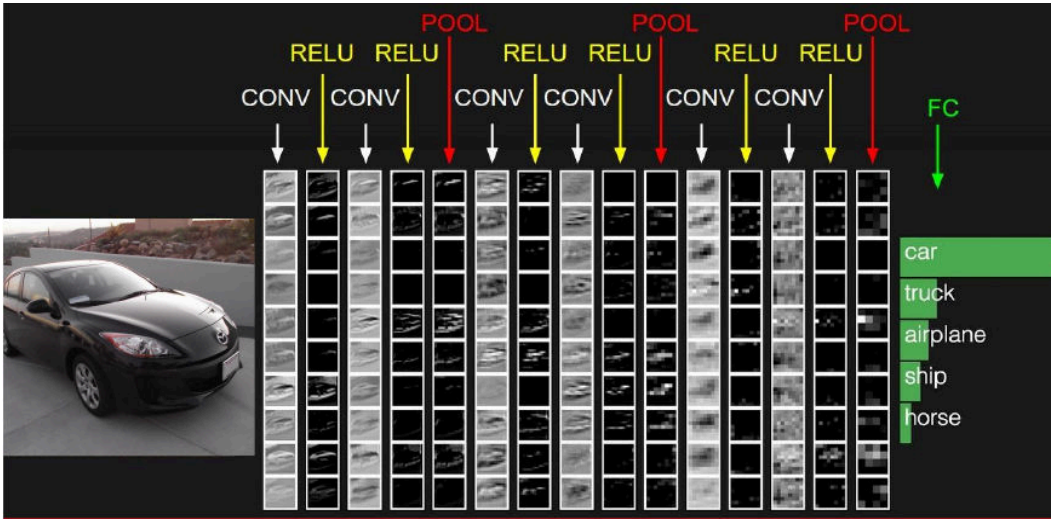
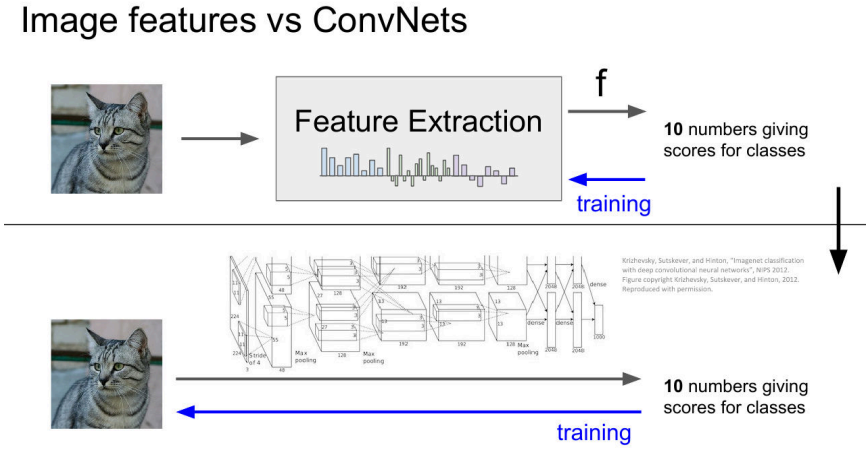


Figure: A typical "classic" convolutional neural network, ConvNet, CNN – Src: CS231n

Deep vs. shallow learning

Deep may mean "many layers" (as in a deep network) but may also mean hierarchical "end-to-end learning" (avoiding engineered features) – that is "Deep learning".



Deep learning software frameworks

A few popular DL frameworks

- Matlab, easy to get going, easy to install (on Windows), good examples, propriety, limiting flexibility after a while. Make sure to use the latest version!
- PyTorch is good for research, easy to modify – most popular at the Centre for Image Analysis!
- TensorFlow is a safe bet for most projects. Not perfect but has huge community, wide usage. Maybe pair with high-level wrapper (**Keras**, Sonnet, etc), good for one graph over many machines
- Consider Caffe2, or TensorFlow for production deployment and mobile

The point of deep learning frameworks

(1) Quick to develop and test new ideas
(2) Automatically compute gradients
(3) Run it all efficiently on GPU (wrap cuDNN, cuBLAS, etc)

Computational Graphs

Numpy

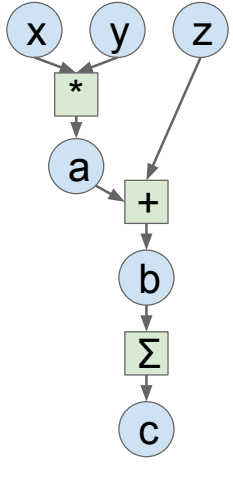
```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



Good:

- Clean API, easy to write numeric code

Bad:

- Have to compute our own gradients
- Can't run on GPU

PyTorch
(More detail)

PyTorch: Fundamental Concepts

Tensor: Like a numpy array, but can run on GPU

Autograd: Package for building computational graphs out of Tensors, and automatically computing gradients

Module: A neural network layer; may store state or learnable weights

PyTorch: Tensors

PyTorch Tensors are just like numpy arrays, but they can run on GPU.

PyTorch Tensor API looks almost exactly like numpy!

Here we fit a two-layer net using PyTorch Tensors:

```
import torch

device = torch.device('cpu')

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in, device=device)
y = torch.randn(N, D_out, device=device)
w1 = torch.randn(D_in, H, device=device)
w2 = torch.randn(H, D_out, device=device)

learning_rate = 1e-6
for t in range(500):
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)
    loss = (y_pred - y).pow(2).sum()

    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = x.t().mm(grad_h)

    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```

PyTorch: Tensors

To run on GPU, just use a different device!

```
import torch

device = torch.device('cuda:0')

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in, device=device)
y = torch.randn(N, D_out, device=device)
w1 = torch.randn(D_in, H, device=device)
w2 = torch.randn(H, D_out, device=device)

learning_rate = 1e-6
for t in range(500):
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)
    loss = (y_pred - y).pow(2).sum()

    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = x.t().mm(grad_h)

    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```

PyTorch: Autograd

Creating Tensors with requires_grad=True enables autograd

Operations on Tensors with requires_grad=True cause PyTorch to build a computational graph

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()

    with torch.no_grad():
        w1 -= learning_rate * w1.grad
        w2 -= learning_rate * w2.grad
        w1.grad.zero_()
        w2.grad.zero_()
```

PyTorch: nn

Higher-level wrapper for working with neural nets

Use this! It will make your life easier

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out))

learning_rate = 1e-2
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()

    with torch.no_grad():
        for param in model.parameters():
            param -= learning_rate * param.grad
    model.zero_grad()
```

PyTorch: optim

Use an **optimizer** for different update rules

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out))

learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(),
                             lr=learning_rate)
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()

    optimizer.step()
    optimizer.zero_grad()
```

TensorFlow

TensorFlow: Neural Net

First **define** computational graph

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])
```

Then **run** the graph many times

```
with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

Keras: High-Level Wrapper

```
N, D, H = 64, 1000, 100
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(H, input_shape=(D,),
                                activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(D))

model.compile(loss=tf.keras.losses.mean_squared_error,
              optimizer=tf.keras.optimizers.SGD(lr=1e0))

x = np.random.randn(N, D)
y = np.random.randn(N, D)

history = model.fit(x, y, epochs=50, batch_size=N)
```

Keras can handle the training loop for you! No sessions or feed_dict

UPPSALA UNIVERSITET

MATLAB

Quite similar to Keras – Easy to get started, good examples

```
%% Specify the size of the images in the input layer of the network.
imsize=size(readimage(imds,1));
outsize=size(labelCount,1);

%% Divide the data into training and validation data sets
trainFraction = 0.75;
[imdsTrain,imdsValidation] = splitEachLabel(imds,trainFraction,'randomize');

%% Define the convolutional neural network architecture.
layers = [
    imageInputLayer(imsize)
    convolution2dLayer(3,8,'Padding',1)
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,16,'Padding',1)
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,32,'Padding',1)
    reluLayer
    fullyConnectedLayer(outsize)
    softmaxLayer
    classificationLayer];

%% Set training options
options = trainingOptions('adam', ...
    'InitialLearnRate',0.001, ...
    'MaxEpochs',3, ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',30, ...
    'ValidationPatience',inf, ...
    'Verbose',false, ...
    'Plots','training-progress', ...
    'L2Regularization',1e-4, ...
    'Shuffle','every-epoch', ... %Strange that this is not the default
    'MiniBatchSize',128);

%% Train Network Using Training Data
rng('default');
net = trainNetwork(imdsTrain,layers,options);

%% Classify Validation Images and Compute Accuracy
imdsTest = imageDatastore('MNIST/Test', 'IncludeSubfolders',true,'LabelSource','foldernames');
YPred = classify(net,imdsTest); YTest = imdsTest.Labels;

accuracy = sum(YPred == YTest)/numel(YTest);
fprintf('Test accuracy: %f\n',accuracy);
```

A few popular DL frameworks

- Matlab, easy to get going, easy to install (on Windows), good examples, propriety, limiting flexibility after a while. Make sure to use the latest version!
- **PyTorch** is good for research, easy to modify – most popular at the Centre for Image Analysis!
- TensorFlow is a safe bet for most projects. Not perfect but has huge community, wide usage. Maybe pair with high-level wrapper (**Keras**, Sonnet, etc), good for one graph over many machines
- Consider Caffe2, or TensorFlow for production deployment and mobile

A few popular DL frameworks

- Matlab, easy to get going, easy to install (on Windows), good examples, propriety, limiting flexibility after a while. Make sure to use the latest version!
- **PyTorch** is good for research, easy to modify – most popular at the Centre for Image Analysis!
- TensorFlow is a safe bet for most projects. Not perfect but has huge community, wide usage. Maybe pair with high-level wrapper (**Keras**, Sonnet, etc), good for one graph over many machines
- Consider Caffe2, or TensorFlow for production deployment and mobile
- **Use the tons of onlince resources out there!**

Going deeper

Lenet

1998

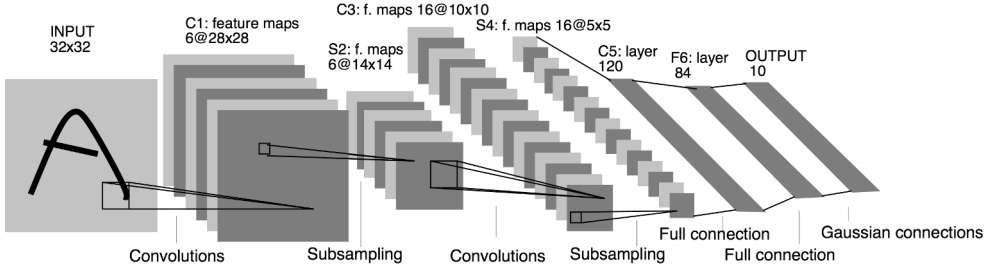


Figure: Src. Yann LeCun, et al, Gradient-based learning applied to document recognition, 1998

Alexnet
2012

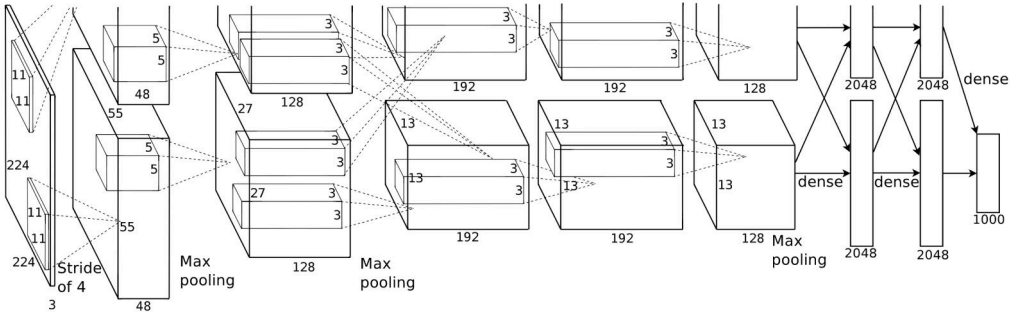


Figure: Src. Alex Krishevsky et al, ImageNet Classification with Deep Convolutional Neural Networks, 2012

VGG16

2014 – *Visual Geometry Group at University of Oxford*

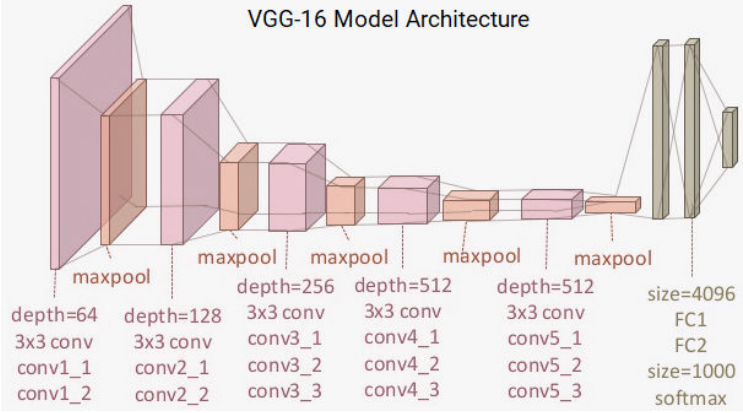


Figure: Simonyan, K., & Zisserman, A. Very deep convolutional networks for large-scale image recognition, 2014

Fully convolutional network

The fully connected (dense) layers at the end of the network require fixed image sizes. Re-scaling implies interpolation and information loss.

Better to use a **Fully convolutional network** = a network without any fully connected layers (i.e., only convolution and pooling).

The fully connected layers (FC layers) perform the classification tasks for us. Observe that there are two ways in which we can build FC layers:

- Dense layers
- 1x1 convolutions (of a 1-pixel sized feature map)

How to reach the 1-pixel feature map? By pooling; not by fixed size pooling but **Global pooling** (average (GAP) or max).

VGG16

2014

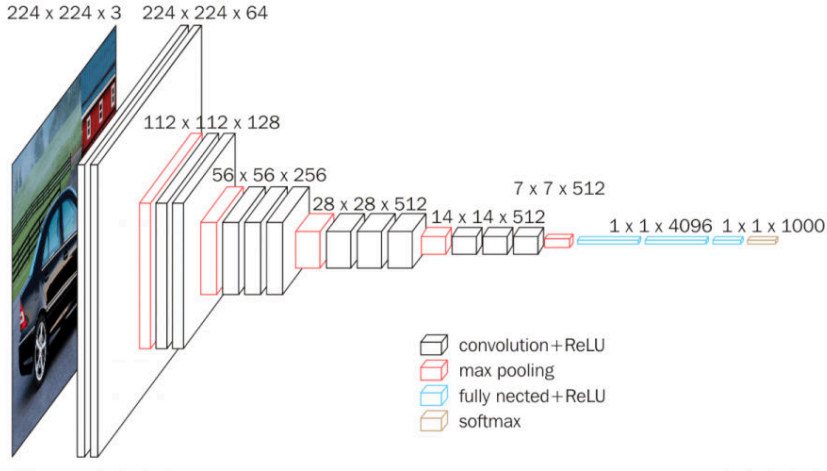


Figure: Save your GPU by skipping one of the FC 4096 layers! It's overkill; generally does not lead to better performance.

Designing CNNs in a nutshell

2012-2014

Designing CNNs in a nutshell

2012-2014

A struggle to avoid *vanishing gradients*

Googlenet

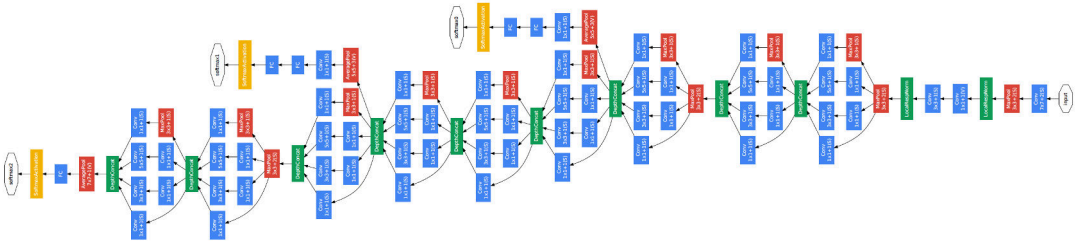


Figure: Src. Going deeper with convolutions, Szegedy et al., 2015

Googlenet - inception layer

Going wider instead, with different scales in parallel

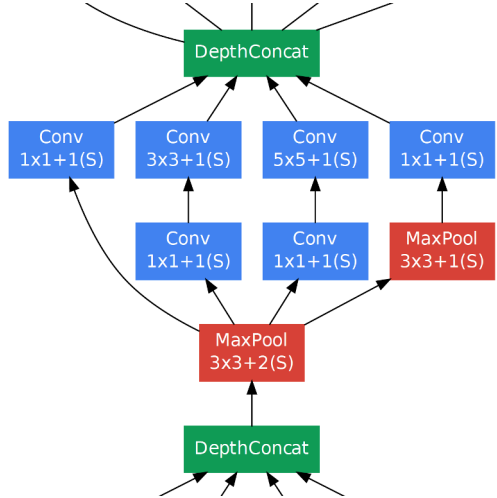


Figure: Src. Going deeper with convolutions, Szegedy et al., 2015

Residual encoding

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

Abstract

Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we

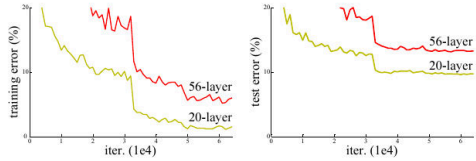


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

An approach which has stood the test of time

Remember how gradients flow through a plus operation

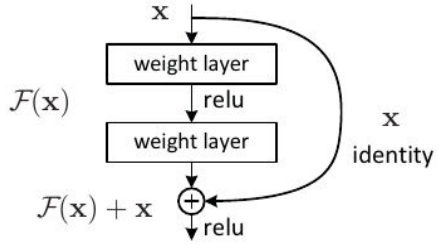


Figure 2. Residual learning: a building block.

Figure: Src. Deep Residual Learning for Image Recognition, He et al. 2016

Resnet

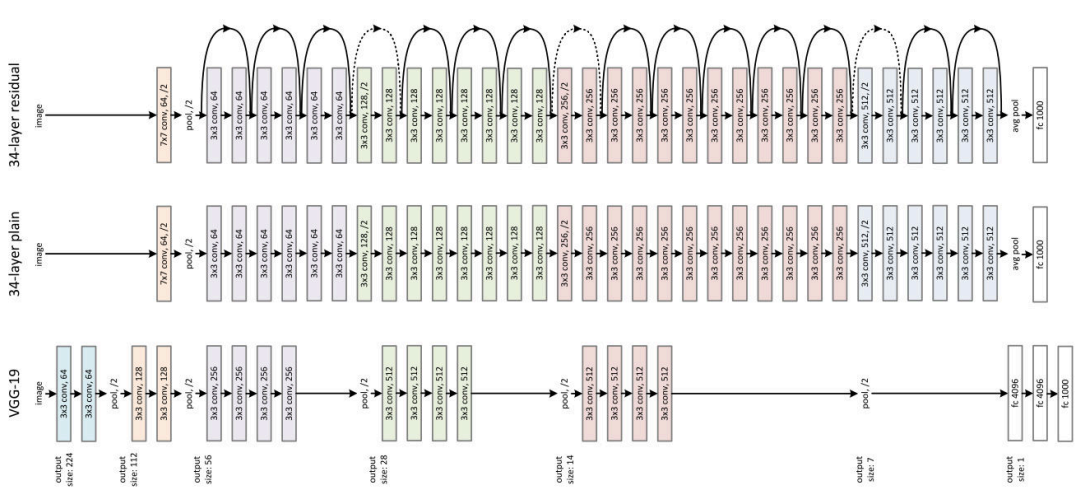


Figure: Src. Deep Residual Learning for Image Recognition, He et al. 2016

Resnet

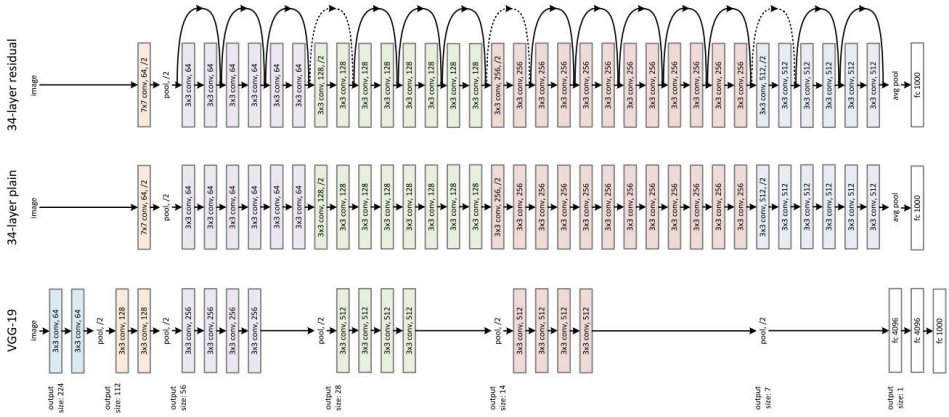
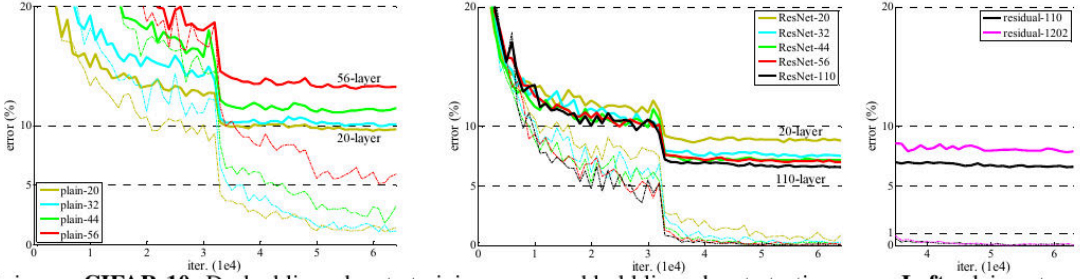


Figure: Src. Deep Residual Learning for Image Recognition, He et al. 2016

Notice: Fully Convolutional Network (FCN) – no Fully Connected layers, only Convolutions and its sidekicks (maxpool/batchnorm/relu)

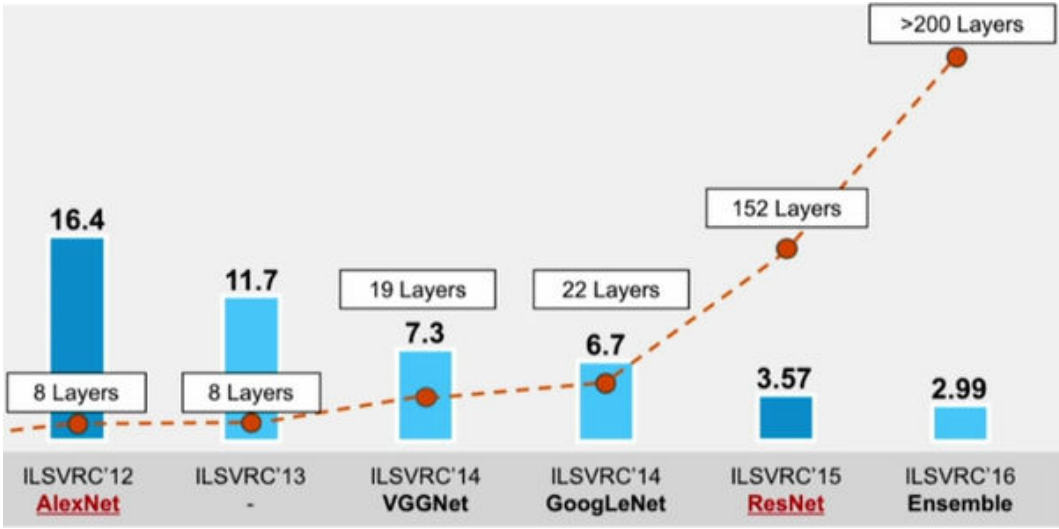
Resnet



ure 6. Training on **CIFAR-10**. Dashed lines denote training error, and bold lines denote testing error. **Left**: plain networks. plain-110 is higher than 60% and not displayed. **Middle**: ResNets. **Right**: ResNets with 110 and 1202 layers.

Figure: Src. Deep Residual Learning for Image Recognition, He et al. 2016

Deeeeeep


>200 Layers
152 Layers
16.4
11.7
19 Layers
22 Layers
7.3
6.7
3.57
2.99
8 Layers
8 Layers
ILSVRC'12
AlexNet
ILSVRC'13
-
ILSVRC'14
VGGNet
ILSVRC'14
GoogLeNet
ILSVRC'15
ResNet
ILSVRC'16
Ensemble

Semantic segmentation

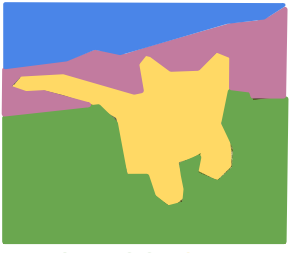
Computer Vision Tasks

Classification

CAT

No spatial extent

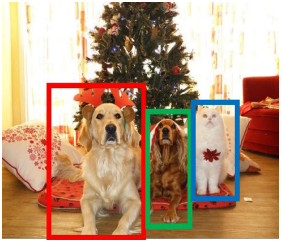
Semantic Segmentation



GRASS, **CAT**, **TREE**, **SKY**

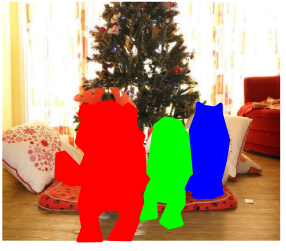
No objects, just pixels

Object Detection



DOG, **DOG**, **CAT**

Instance Segmentation



DOG, **DOG**, **CAT**

Multiple Object

Semantic Segmentation

Classification

CAT

No spatial extent

Semantic Segmentation

GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection

DOG, DOG, CAT

Instance Segmentation

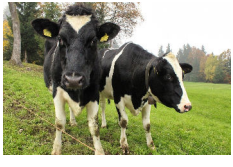
DOG, DOG, CAT

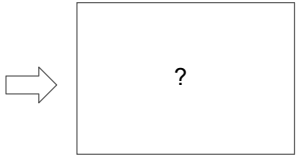
Multiple Object

Semantic Segmentation: The Problem

GRASS, **CAT**,
TREE, **SKY**, **...**

Paired training data: for each training image, each pixel is labeled with a semantic category.





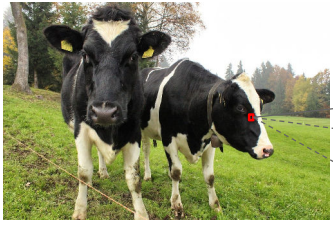
At test time, classify each pixel of a new image.

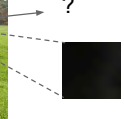
Semantic Segmentation Idea: Sliding Window

Full image

Semantic Segmentation Idea: Sliding Window

Full image

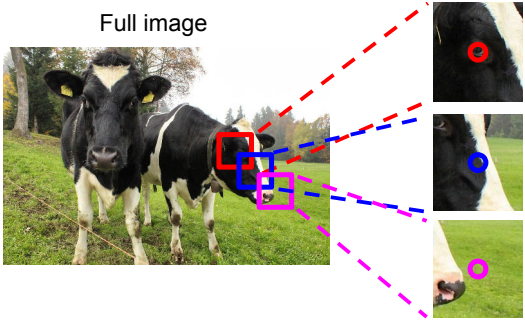




Impossible to classify without context

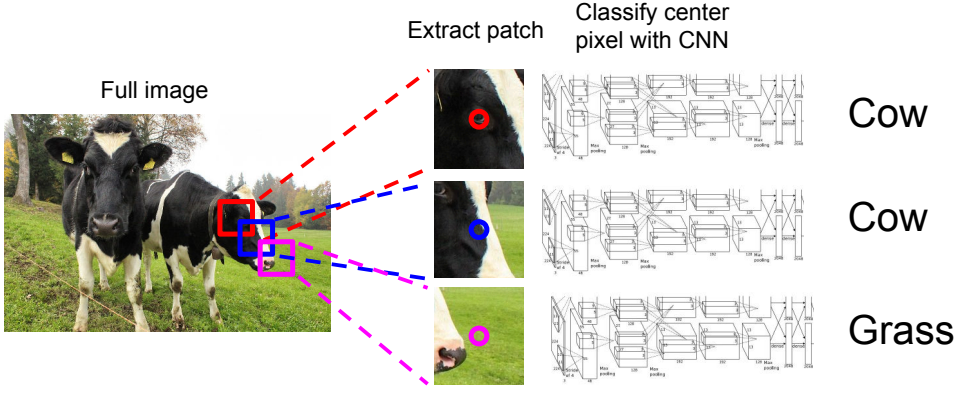
Q: how do we include context?

Semantic Segmentation Idea: Sliding Window



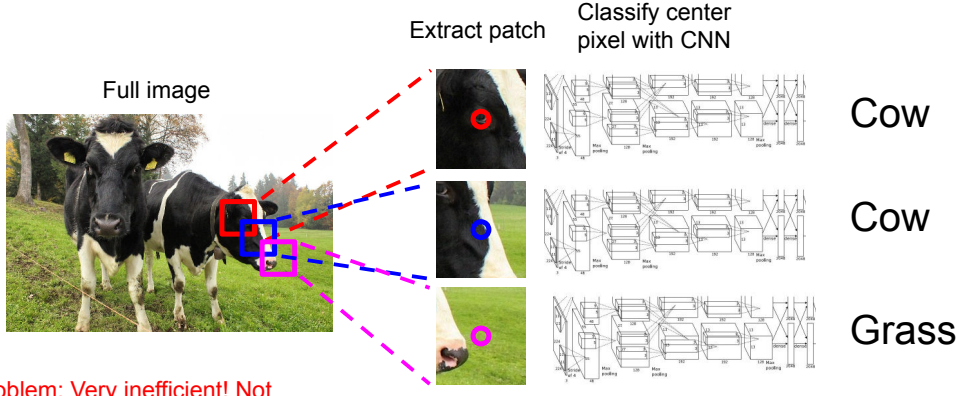
Q: how do we model this?

Semantic Segmentation Idea: Sliding Window



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

Semantic Segmentation Idea: Sliding Window

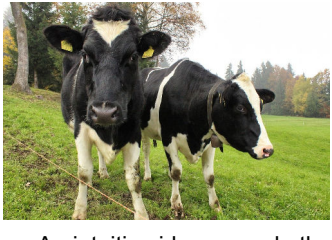


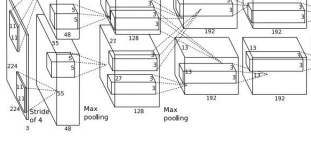
Problem: Very inefficient! Not reusing shared features between overlapping patches

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

Semantic Segmentation Idea: Convolution

Full image



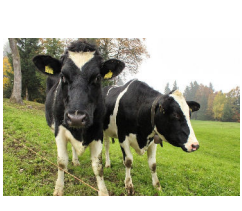


An intuitive idea: encode the entire image with conv net, and do semantic segmentation on top.

Problem: classification architectures often reduce feature spatial sizes to go deeper, but semantic segmentation requires the output size to be the same as input size.

Semantic Segmentation Idea: Fully Convolutional

Design a network with only convolutional layers without downsampling operators to make predictions for pixels all at once!



Input:
3 x H x W

Conv

Conv

Conv

Conv

Convolutions:
D x H x W

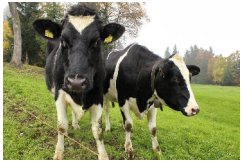
Scores:
C x H x W

argmax

Predictions:
H x W

Semantic Segmentation Idea: Fully Convolutional

Design a network with only convolutional layers without downsampling operators to make predictions for pixels all at once!



Input:
3 x H x W

Conv

Conv

Conv

Conv

Convolutions:
D x H x W

Scores:
C x H x W

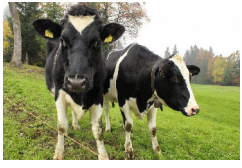
argmax

Predictions:
H x W

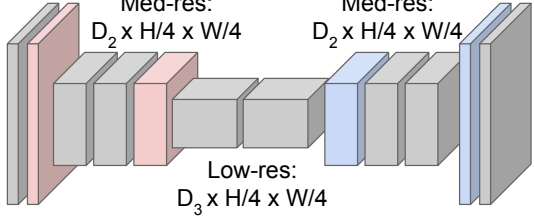
Problem: convolutions at original image resolution will be very expensive ...

Semantic Segmentation Idea: Fully Convolutional

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!



Input:
3 x H x W



High-res:
D_1 x H/2 x W/2

High-res:
D_1 x H/2 x W/2

Predictions:
H x W

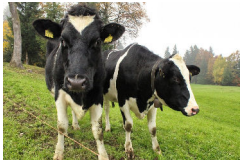
Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Semantic Segmentation Idea: Fully Convolutional

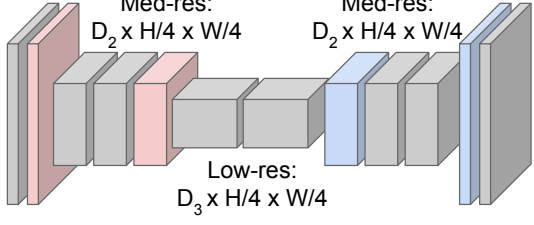
Downsampling: Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling: ???



Input:
3 x H x W



High-res:
D1 x H/2 x W/2

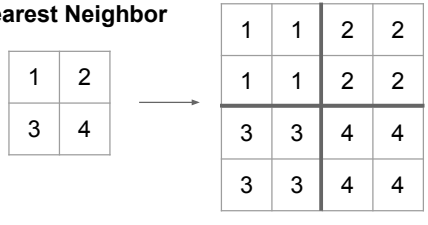
High-res:
D1 x H/2 x W/2

Predictions:
H x W

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

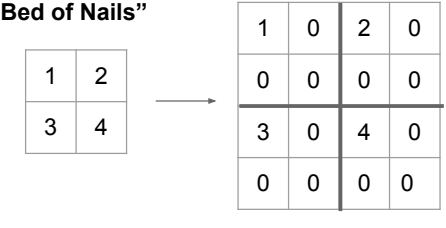
In-Network upsampling: “Unpooling”

Nearest Neighbor



Input: 2 x 2 Output: 4 x 4

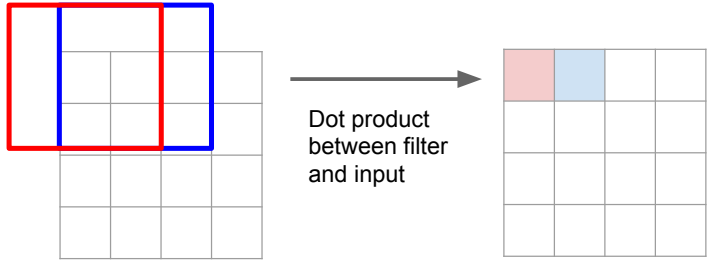
“Bed of Nails”



Input: 2 x 2 Output: 4 x 4

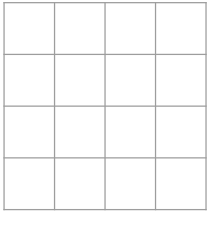
Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 1 pad 1

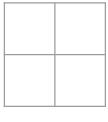


Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



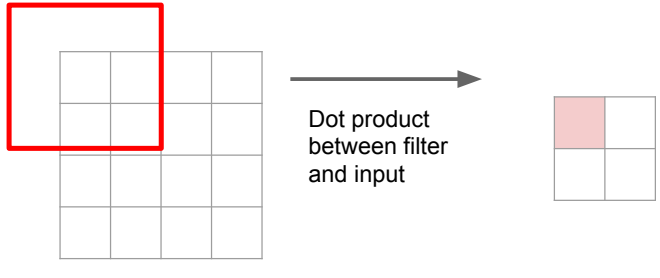
Input: 4 x 4



Output: 2 x 2

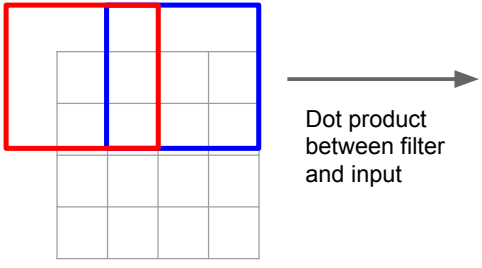
Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



Input: 4 x 4

Output: 2 x 2

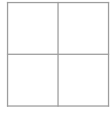
Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output

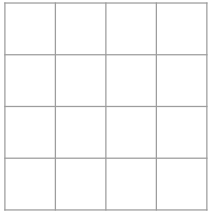
We can interpret strided convolution as “learnable downsampling”.

Learnable Upsampling: Transpose Convolution

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



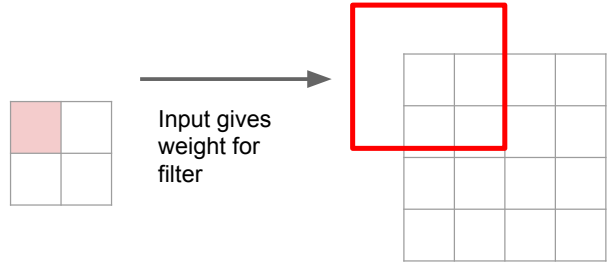
Input: 2 x 2



Output: 4 x 4

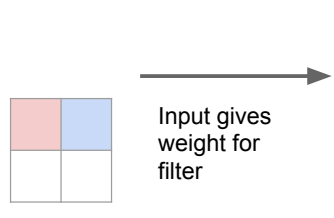
Learnable Upsampling: Transpose Convolution

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



Learnable Upsampling: Transpose Convolution

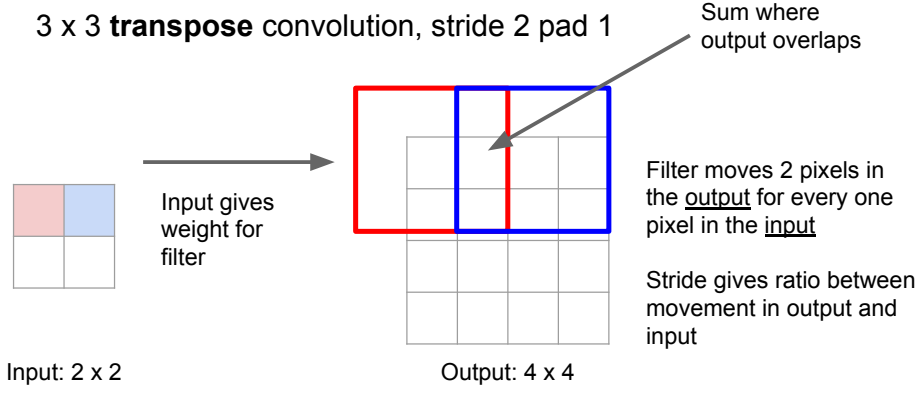
3 x 3 **transpose** convolution, stride 2 pad 1



Filter moves 2 pixels in the output for every one pixel in the input

Stride gives ratio between movement in output and input

Learnable Upsampling: Transpose Convolution

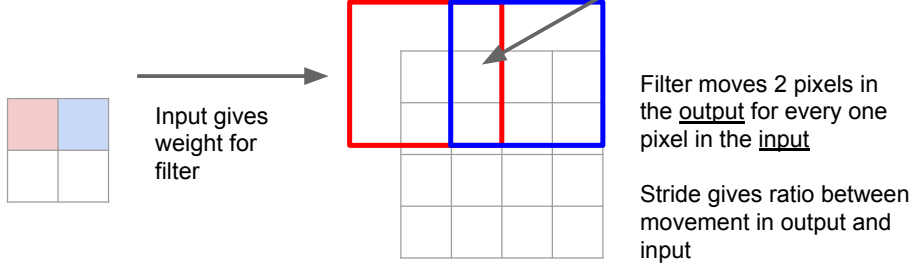


Learnable Upsampling: Transpose Convolution

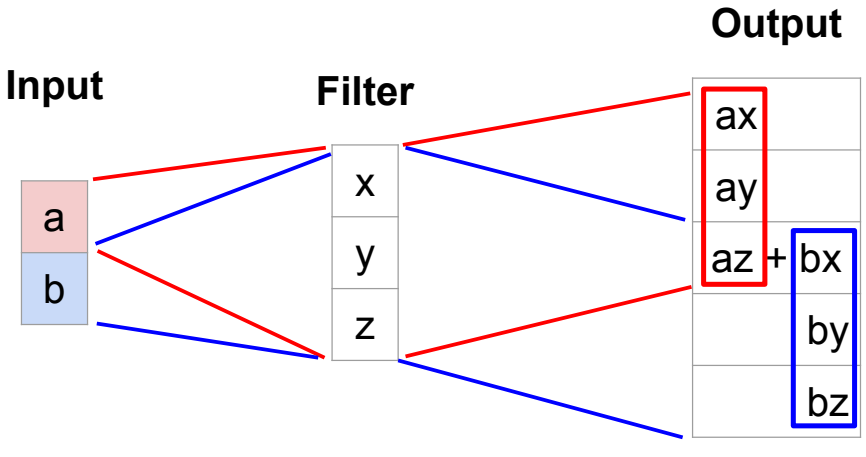
Other names:
-Deconvolution (bad)
-Upconvolution
-Fractionally strided convolution
-Backward strided convolution

Q: Why is it called transpose convolution?

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



Learnable Upsampling: 1D Example



Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & 0 & x & y & x & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=2, padding=1

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & 0 & x & y & x & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=2, padding=1

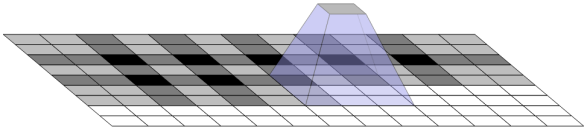
Convolution transpose multiplies by the transpose of the same matrix:

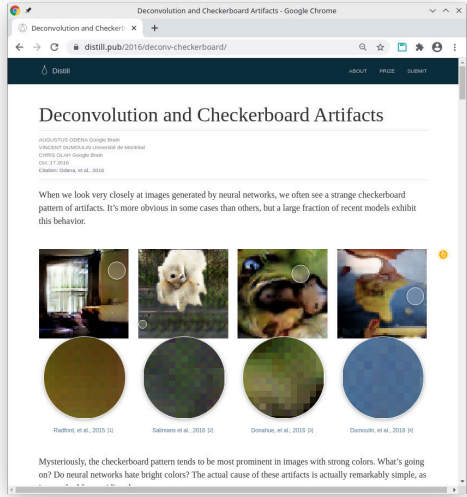
$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

Example: 1D transpose conv, kernel size=3, stride=2, padding=0

Checkerboard artifacts





- Uneven overlap easily leads to checkerboard patterns
- Make sure to match stride and kernel size!
- Recommended to use stride=2 and 4×4 kernel.

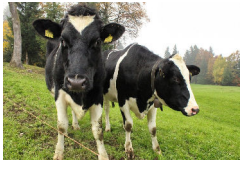
See also
https://distill.pub/2016/deconv-checkerboard/
for more on this topic.

Semantic Segmentation Idea: Fully Convolutional

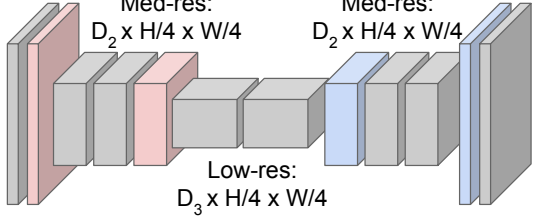
Downsampling: Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling: Unpooling or strided transpose convolution



Input:
3 x H x W



High-res:
D1 x H/2 x W/2

High-res:
D1 x H/2 x W/2

Predictions:
H x W

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

U-Net

Skip connections between downsampling and upsampling path

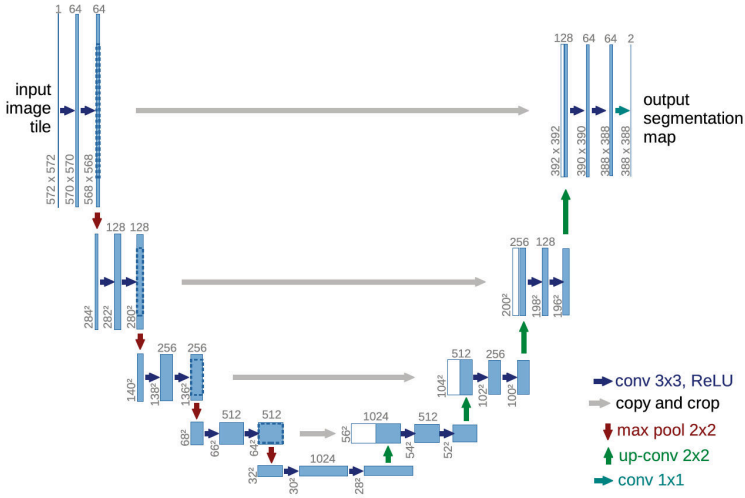


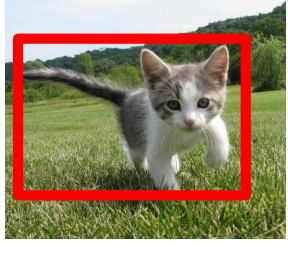
Figure: Src. Olaf Ronneberger, et al., U-Net: Convolutional Networks for Biomedical Image Segmentation, MICCAI 2015

Localization, detection, instance segmentation

Classification + Localization

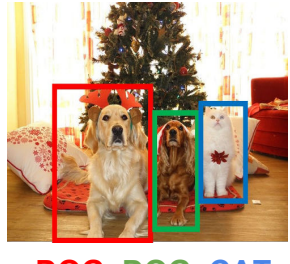
GRASS, CAT, TREE, SKY

No objects, just pixels



CAT

Single Object



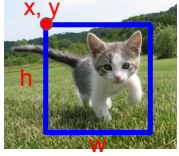
DOG, DOG, CAT

DOG, DOG, CAT

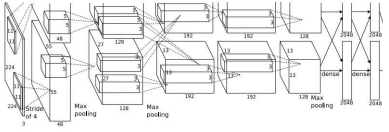
Multiple Object

Object Detection: Single Object

(Classification + Localization)



This image is CC0 public domain



Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

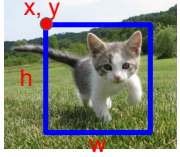
Vector: 4096

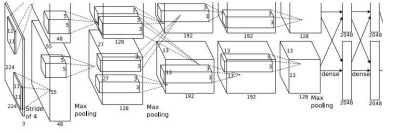
Fully Connected: 4096 to 4

Box Coordinates
(x, y, w, h)

Object Detection: Single Object

(Classification + Localization)





Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Correct label: Cat

Vector: 4096

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

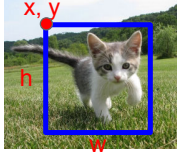
L2 Loss

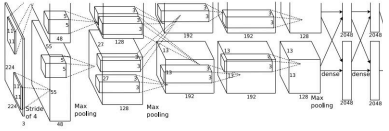
Correct box: (x', y', w', h')

Treat localization as a regression problem!

Object Detection: Single Object

(Classification + Localization)





Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Correct label: Cat

Vector: 4096

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

L2 Loss

Correct box: (x', y', w', h')

Multitask Loss

+

Loss

Treat localization as a regression problem!

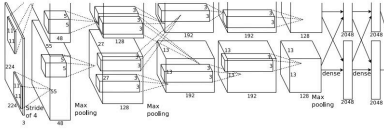
Aside: Human Pose Estimation

Represent pose as a set of 14 joint positions:

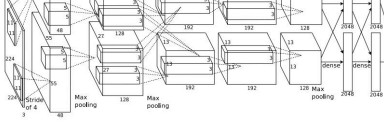
Left / right foot
Left / right knee
Left / right hip
Left / right shoulder
Left / right elbow
Left / right hand
Neck
Head top

Johnson and Everingham, "Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation", BMVC 2010

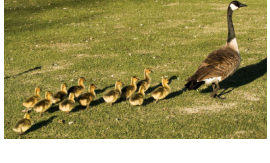
Object Detection: Multiple Objects

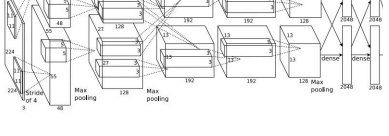


CAT: (x, y, w, h)



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

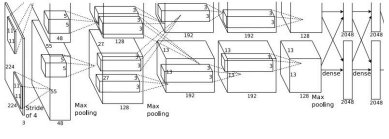




DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
….

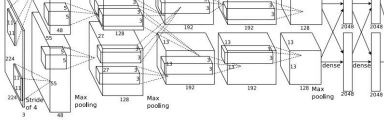
Object Detection: Multiple Objects

Each image needs a different number of outputs!



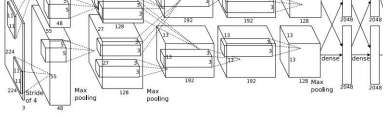
CAT: (x, y, w, h)

4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

12 numbers

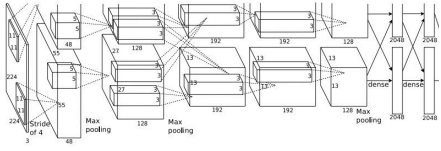


DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
….

Many numbers!

Object Detection: Multiple Objects

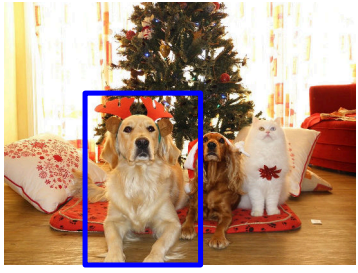
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

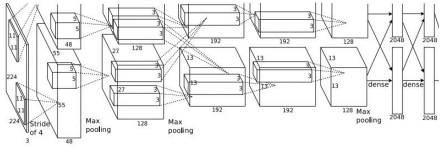


Dog? NO
Cat? NO
Background? YES

Object Detection: Multiple Objects

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

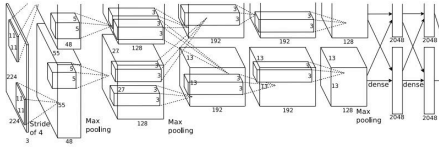




Dog? YES
Cat? NO
Background? NO

Object Detection: Multiple Objects

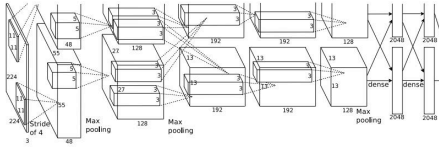
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? YES
Cat? NO
Background? NO

Object Detection: Multiple Objects

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

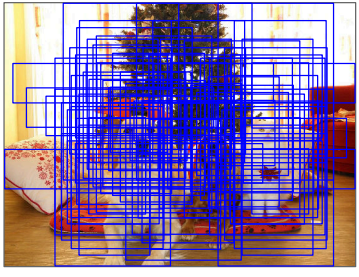


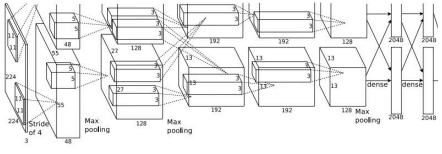
Dog? NO
Cat? YES
Background? NO

Q: What's the problem with this approach?

Object Detection: Multiple Objects

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



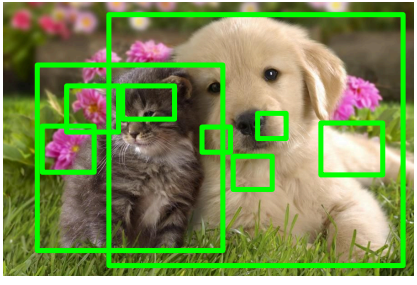


Dog? NO
Cat? YES
Background? NO

Problem: Need to apply CNN to huge number of locations, scales, and aspect ratios, very computationally expensive!

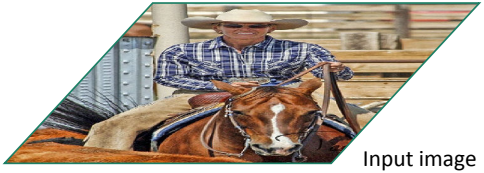
Region Proposals: Selective Search

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU



Alexe et al, “Measuring the objectness of image windows”, TPAMI 2012
Uijlings et al, “Selective Search for Object Recognition”, IJCV 2013
Cheng et al, “BING: Binarized normed gradients for objectness estimation at 300fps”, CVPR 2014
Zitnick and Dollar, “Edge boxes: Locating object proposals from edges”, ECCV 2014

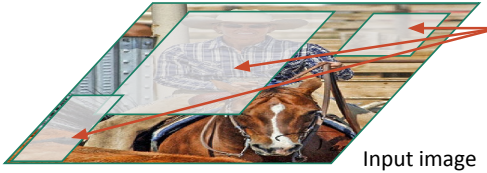
R-CNN



Input image

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN

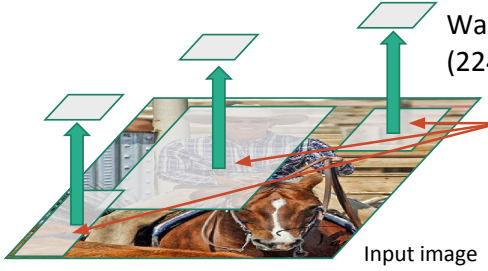


Input image

Regions of Interest (RoI) from a proposal method (~2k)

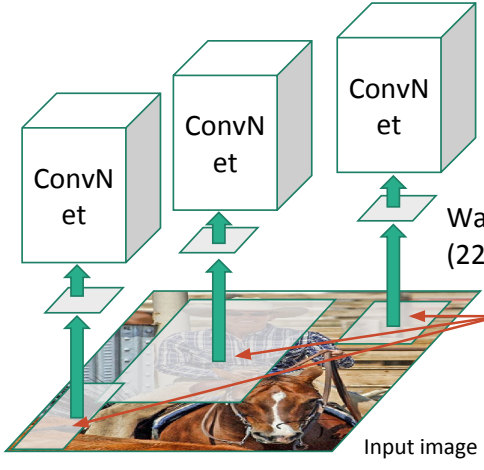
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN



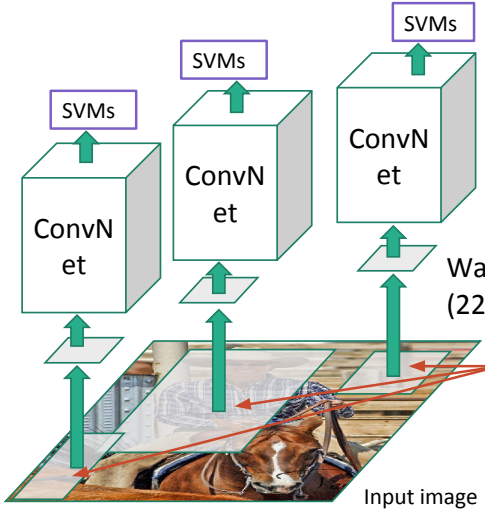
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

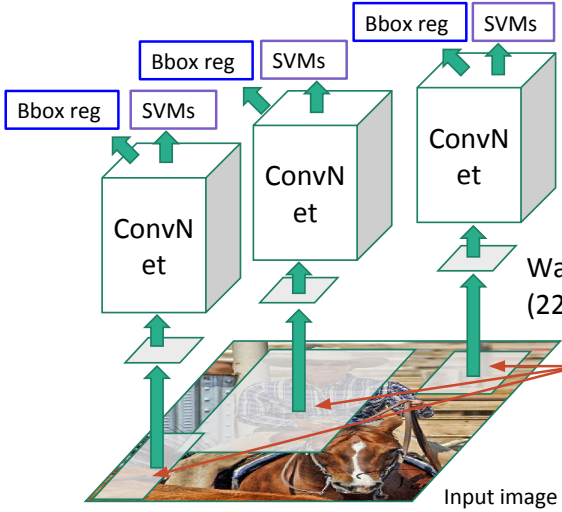
R-CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN

Predict “corrections” to the RoI: 4 numbers: (dx, dy, dw, dh)



Classify regions with SVMs

Forward each region through ConvNet (ImageNet-pretranied)

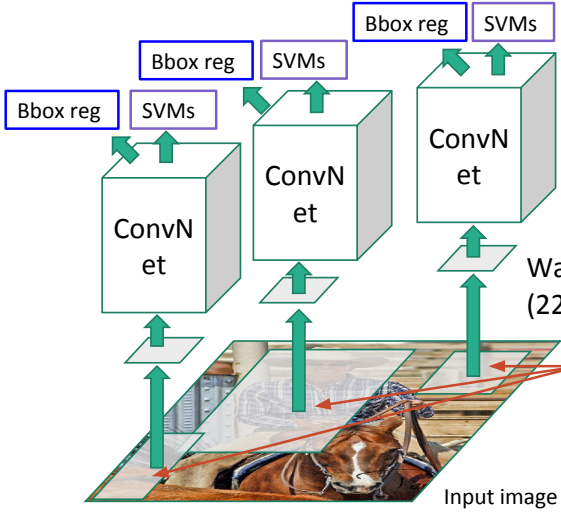
Warped image regions (224x224 pixels)

Regions of Interest (RoI) from a proposal method (~2k)

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN

Predict “corrections” to the RoI: 4 numbers: (dx, dy, dw, dh)



Classify regions with SVMs

Forward each region through ConvNet

Warped image regions (224x224 pixels)

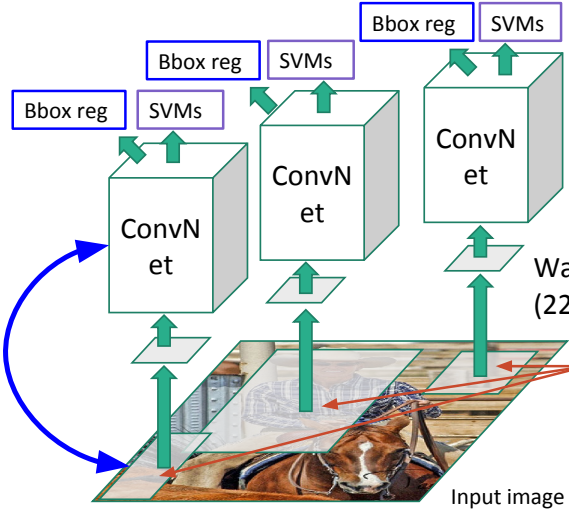
Regions of Interest (RoI) from a proposal method (~2k)

Problem: Very slow! Need to do ~2k independent forward passes for each image!

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

"Slow" R-CNN

Predict "corrections" to the RoI: 4 numbers: (dx, dy, dw, dh)



Classify regions with SVMs

Forward each region through ConvNet

Warped image regions (224x224 pixels)

Regions of Interest (RoI) from a proposal method (~2k)

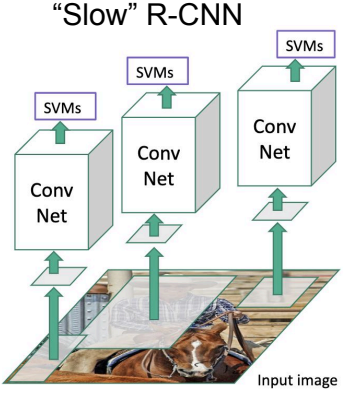
Problem: Very slow! Need to do ~2k independent forward passes for each image!

Idea: Pass the image through convnet before cropping! Crop the conv feature instead!

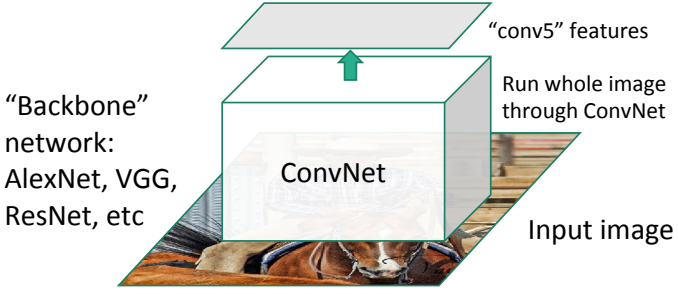
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

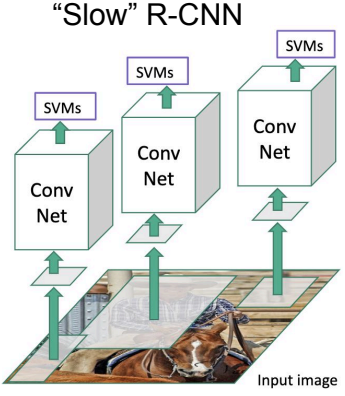
Fast R-CNN

Input image



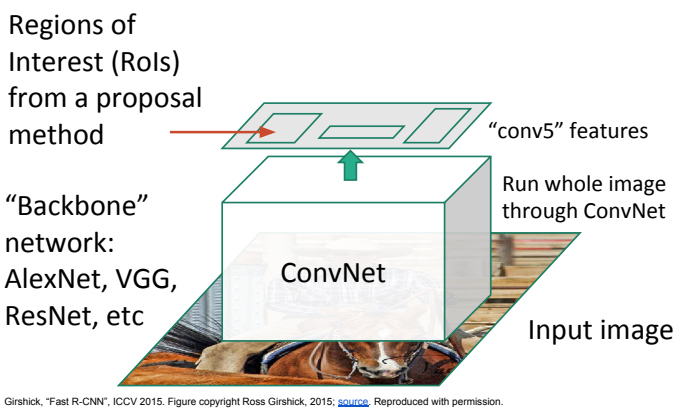
Fast R-CNN



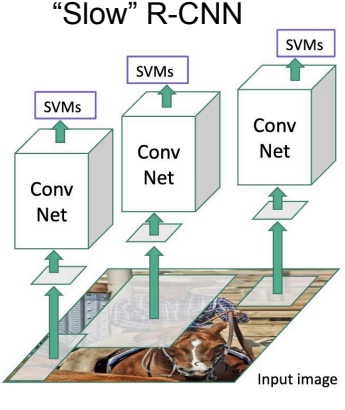


Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

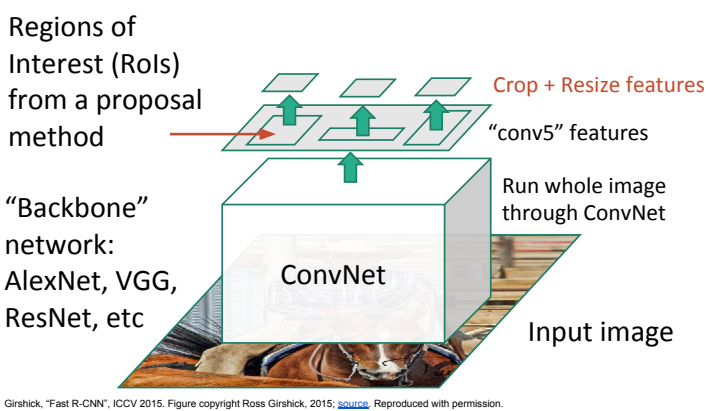
Fast R-CNN

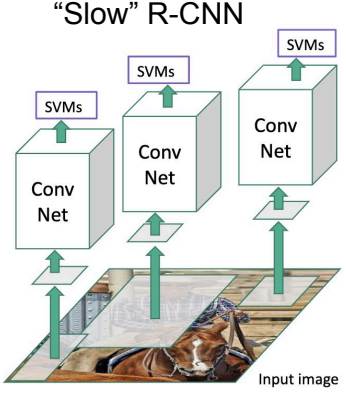


Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.



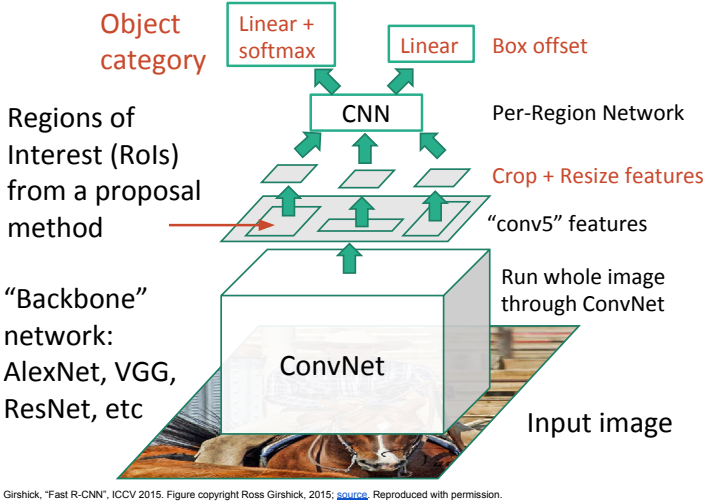
Fast R-CNN

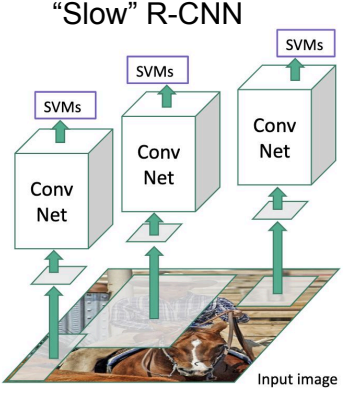




Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

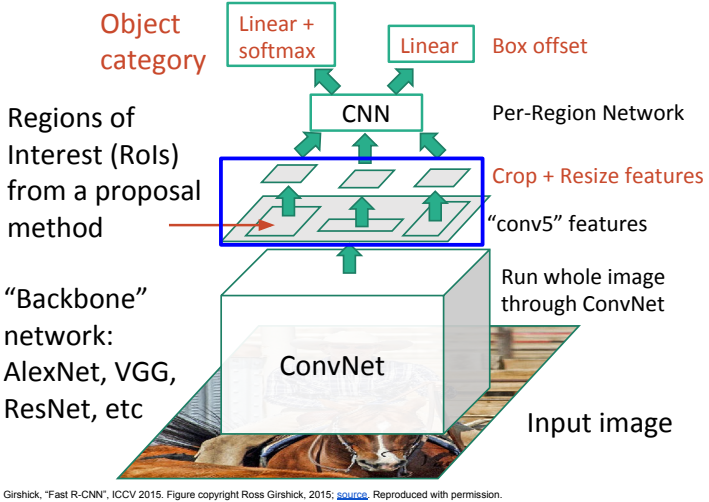
Fast R-CNN

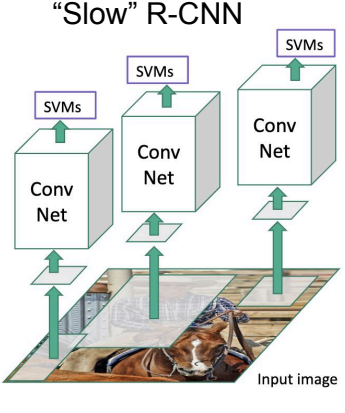




Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

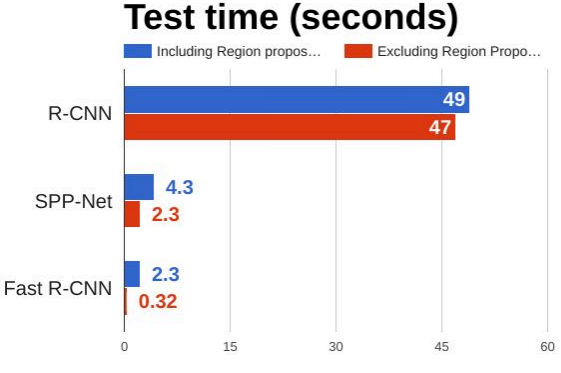
Fast R-CNN





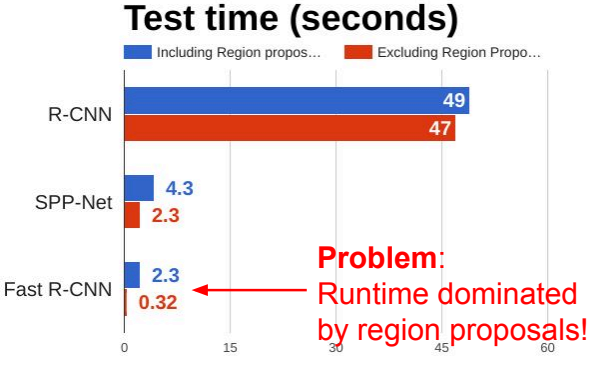
Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN vs Fast R-CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
He et al, "Spatial pyramid pooling in deep convolutional networks for visual recognition", ECCV 2014
Girshick, "Fast R-CNN", ICCV 2015

R-CNN vs Fast R-CNN



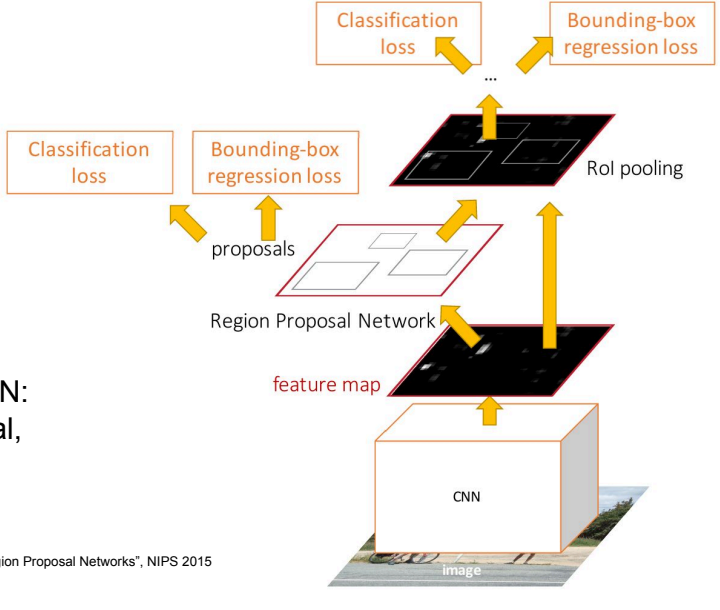
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
He et al, "Spatial pyramid pooling in deep convolutional networks for visual recognition", ECCV 2014
Girshick, "Fast R-CNN", ICCV 2015

Faster R-CNN:

Make CNN do proposals!

Insert **Region Proposal Network (RPN)** to predict proposals from features

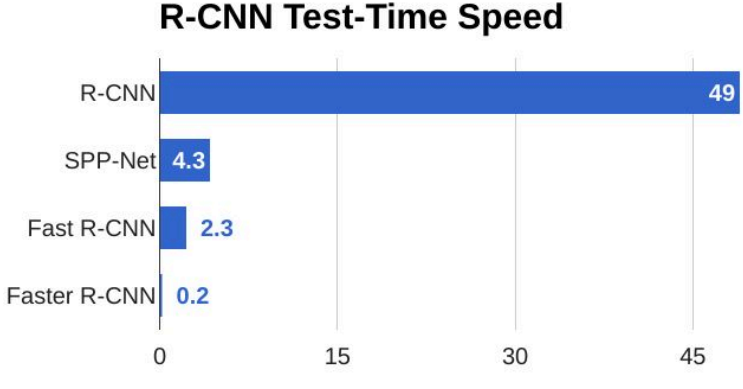
Otherwise same as Fast R-CNN: Crop features for each proposal, classify each one



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

Faster R-CNN:

Make CNN do proposals!



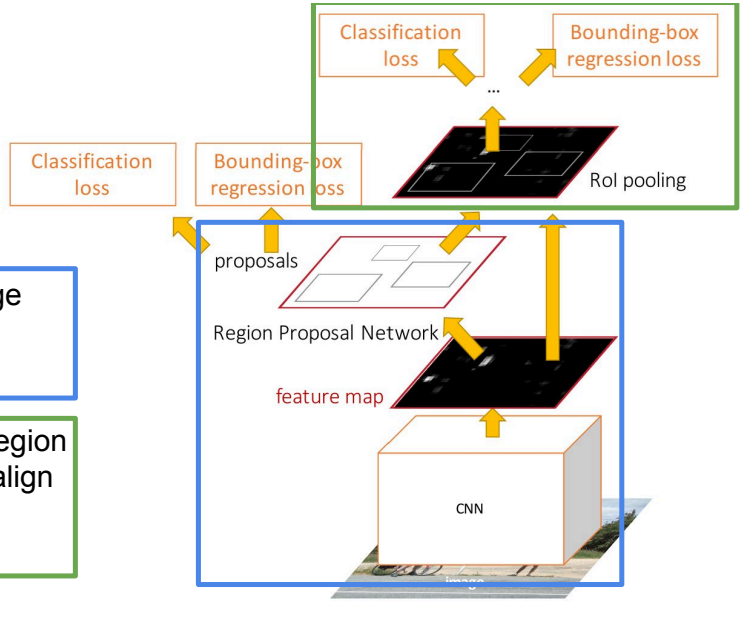
Faster R-CNN:

Make CNN do proposals!

Faster R-CNN is a
Two-stage object detector

First stage: Run once per image
- Backbone network
- Region proposal network

Second stage: Run once per region
- Crop features: RoI pool / align
- Predict object class
- Prediction bbox offset



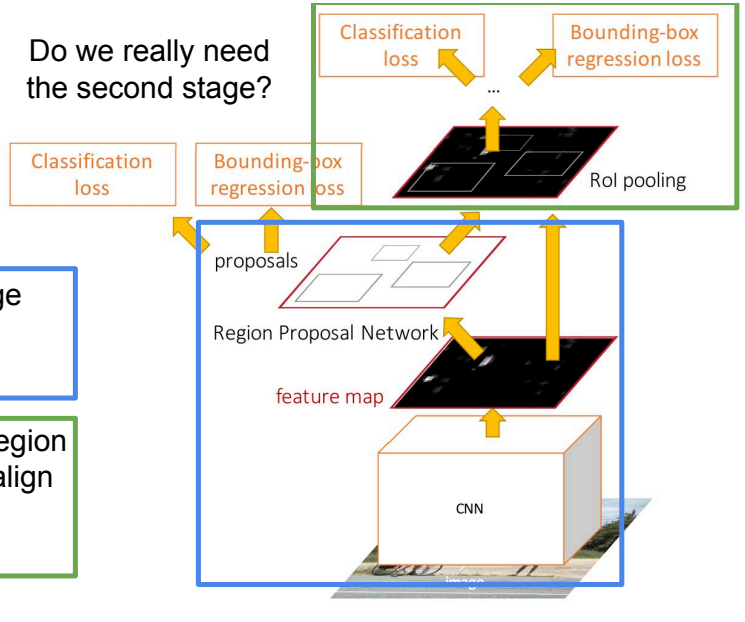
Faster R-CNN:
Make CNN do proposals!

Do we really need the second stage?

Faster R-CNN is a **Two-stage object detector**

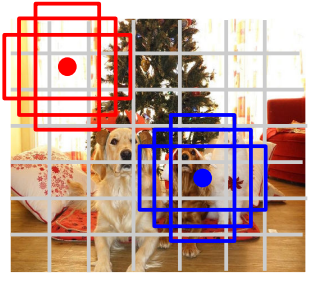
First stage: Run once per image
- Backbone network
- Region proposal network

Second stage: Run once per region
- Crop features: RoI pool / align
- Predict object class
- Prediction bbox offset



Single-Stage Object Detectors: YOLO / SSD / RetinaNet

Input image
3 x H x W



Divide image into grid
7 x 7

Image a set of **base boxes** centered at each grid cell
Here B = 3

Within each grid cell:

- Regress from each of the B base boxes to a final box with 5 numbers: (dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)
- Looks a lot like RPN, but category-specific!

Output:
7 x 7 x (5 * B + C)

Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

Object Detection: Lots of variables ...

Backbone Network
VGG16
ResNet-101
Inception V2
Inception V3
Inception
ResNet
MobileNet

"Meta-Architecture"
Two-stage: Faster R-CNN
Single-stage: YOLO / SSD
Hybrid: R-FCN

Image Size
 # Region Proposals
…

Takeaways
Faster R-CNN is slower but more accurate

SSD is much faster but not as accurate

Bigger / Deeper backbones work better

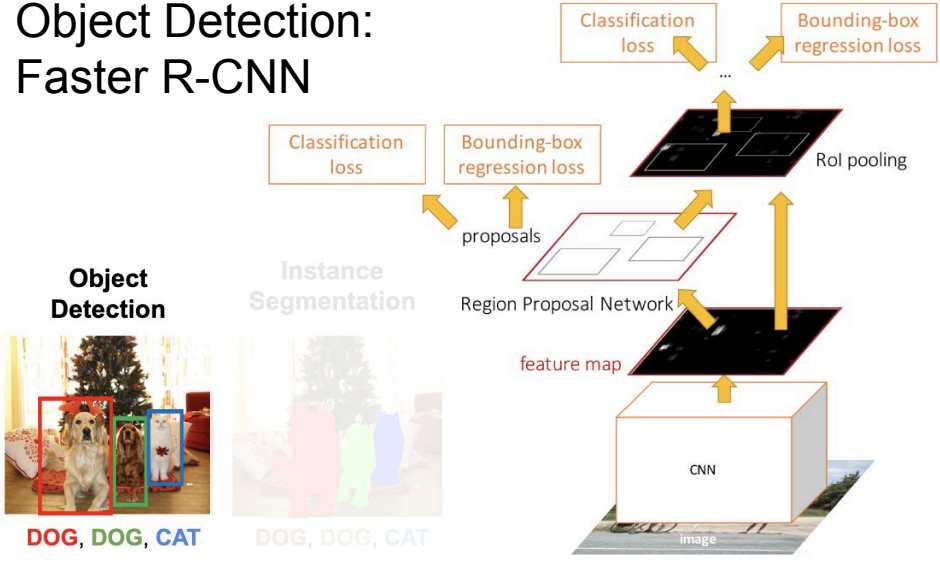
Huang et al, "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017

R-FCN: Dai et al, "R-FCN: Object Detection via Region-based Fully Convolutional Networks", NIPS 2016
Inception-V2: Ioffe and Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", ICML 2015
Inception V3: Szegedy et al, "Rethinking the Inception Architecture for Computer Vision", arXiv 2016
Inception ResNet: Szegedy et al, "Inception-V4, Inception-ResNet and the Impact of Residual Connections on Learning", arXiv 2016
MobileNet: Howard et al, "Efficient Convolutional Neural Networks for Mobile Vision Applications", arXiv 2017

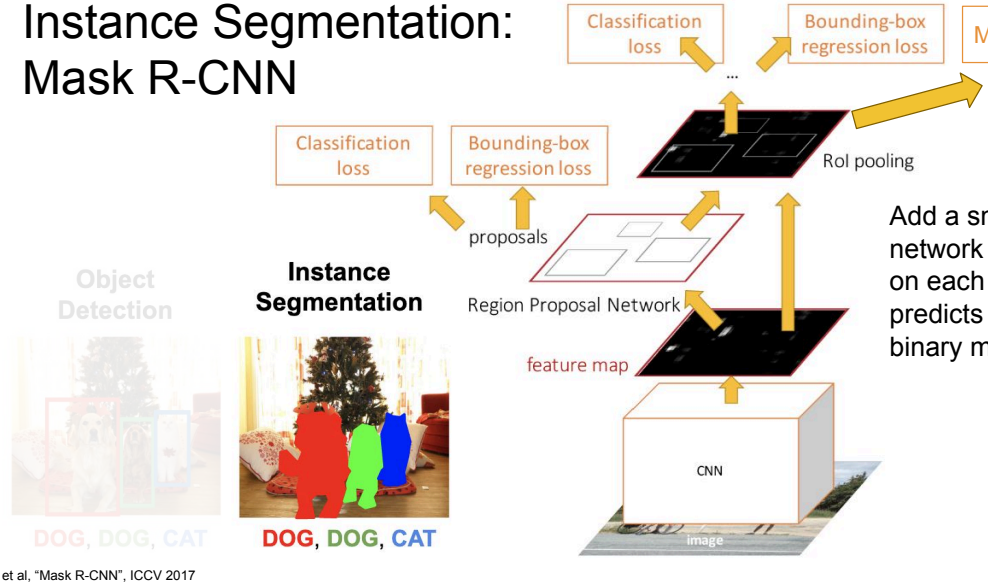
Instance Segmentation

Classification	Semantic Segmentation	Object Detection	**Instance Segmentation**
			
CAT	GRASS, CAT, TREE, SKY	DOG, DOG, CAT	**DOG, DOG, CAT**
No spatial extent	No objects, just pixels	Multiple Object	

Object Detection: Faster R-CNN



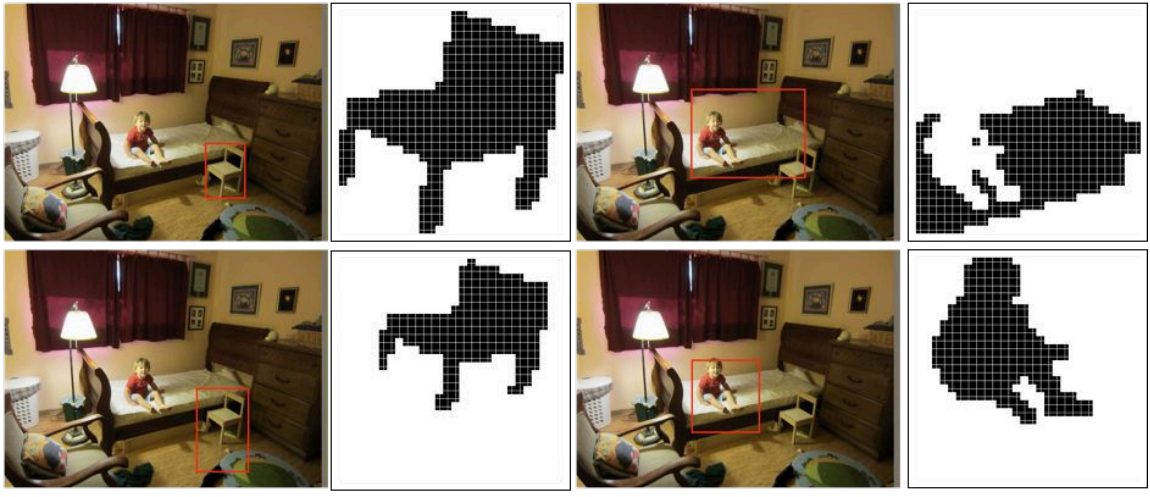
Instance Segmentation: Mask R-CNN



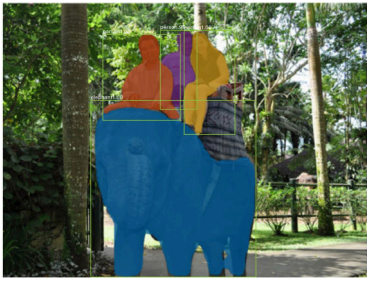
Add a small mask network that operates on each RoI and predicts a 28x28 binary mask

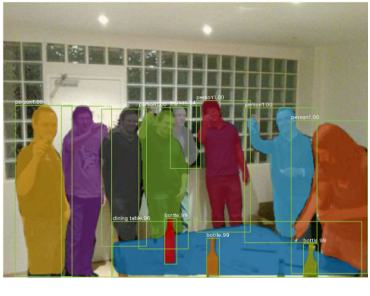
He et al, "Mask R-CNN", ICCV 2017

Mask R-CNN: Example Mask Training Targets



Mask R-CNN: Very Good Results!





He et al, "Mask R-CNN", ICCV 2017

A few types of deep neural networks

- Fully connected
- Convolutional neural network
  - Fully convolutional
  - Residual networks
  - U-Net style (with transposed convolution and skip-links)
  - Region proposal networks (RPN, "Faster R-CNN")
- Recurrent neural network (RNN)
  - LSTM networks [Hochreiter and Schmidhuber (1997)]
- Auto encoders

Recurrent neural networks

- Connections between nodes form directed cycles.
- Create internal memory.
- Used to process sequence of data such as speech, text, video etc.
- Long short term memory (LSTM) commonly used now.

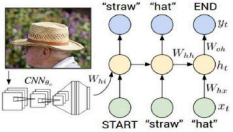
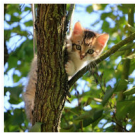


Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.
Reproduced for educational purposes.

A cat sitting on a suitcase on the floor



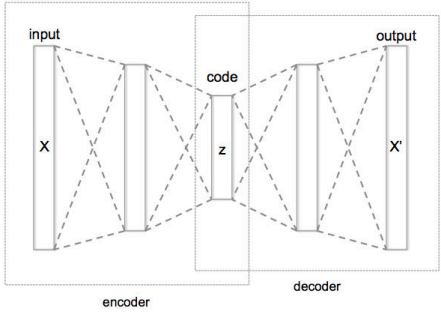
A cat is sitting on a branch

Two people walking on the beach with surfboards

A tennis player action on the c

Figure: Src. cs231n

Auto encoders



- *Unsupervised* feature learning
- Provides dimensionality reduction (compact representation)
- Tries to recreate input after mapping to lower dimension
- Can stack multiple auto encoders to create deeper network

Figure: Src. by Chervinskii - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45555552

Concluding

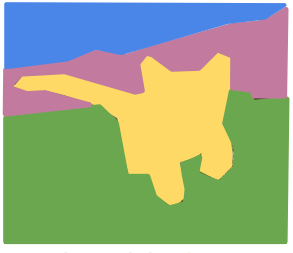
Recap: Lots of computer vision tasks!

Classification

CAT

No spatial extent

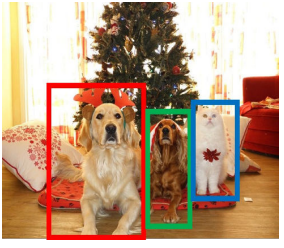
Semantic Segmentation



GRASS, CAT, TREE, SKY

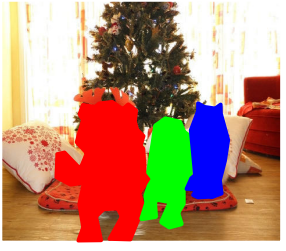
No objects, just pixels

Object Detection



DOG, DOG, CAT

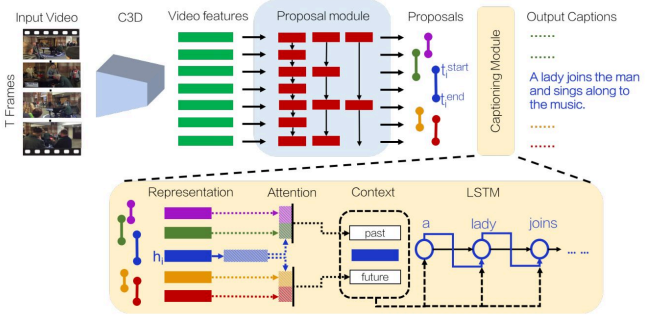
Instance Segmentation

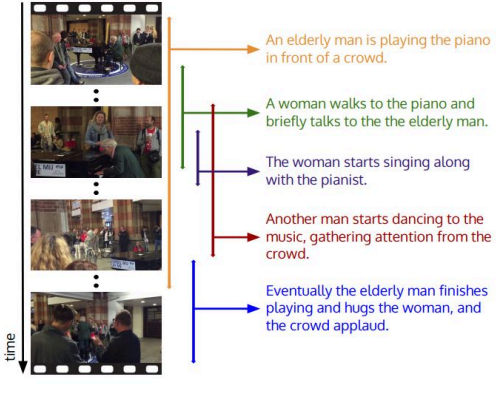


DOG, DOG, CAT

Multiple Object

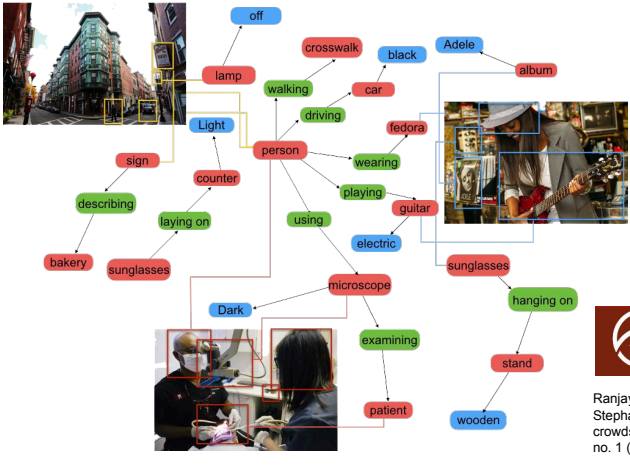
Dense Video Captioning





Ranjay Krishna et al., "Dense-Captioning Events in Videos", ICCV 2017
Figure copyright IEEE, 2017. Reproduced with permission.

Objects + Relationships = Scene Graphs

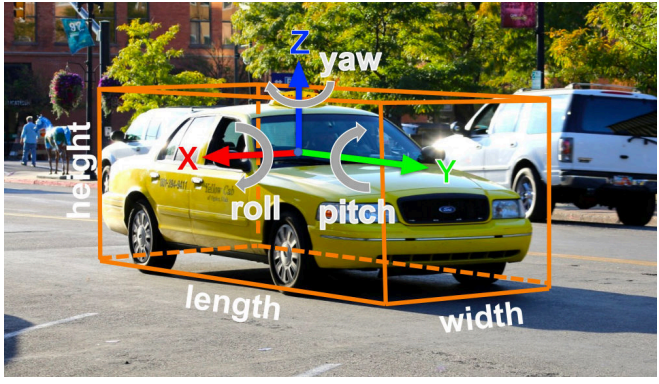


108,077 Images
5.4 Million Region Descriptions
1.7 Million Visual Question Answers
3.8 Million Object Instances
2.8 Million Attributes
2.3 Million Relationships
Everything Mapped to Wordnet Synsets

VISUALGENOME

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations." International Journal of Computer Vision 123, no. 1 (2017): 32-73.

3D Object Detection



2D Object Detection:
2D bounding box
(x, y, w, h)

3D Object Detection:
3D oriented bounding box
(x, y, z, w, h, l, r, p, y)

Simplified bbox: no roll & pitch

Much harder problem than 2D object detection!

UPPSALA UNIVERSITET

A few concepts to summarize lecture 5

Deep learning: End-to-end learning, where the machine learning methods "learns" features.

Shallow learning: Machine learning which relies on engineered features instead of the raw data.

Tensor: In machine learning / data processing a tensor is simply a multidimensional numerical array. *This is not how tensors are defined in mathematics and physics!*

Residual neural network: A network which learns how the output differs from the input. This helps in handling the Vanishing gradient problem and allows deeper networks.

Semantic segmentation: Classification of each pixel in an image with a category label.

Instance segmentation: Segmentation of individual objects in the image.

Transposed convolution: Linear up-sampling (if strided) with an arbitrary (learnable) interpolation kernel. (De-convolution, the inverse of convolution, is something different.)

Skip connection: Connections between nodes in different layers of a neural network that skip one or more layers of nonlinear processing. Facilitating deeper architectures.

Multi-task learning: Solving multiple learning tasks at the same time (usually by defining a multitask loss), while exploiting commonalities and differences across tasks.

Frameworks: PyTorch, Tensorflow, Keras, Matlab... Use the online resources available!

Checkout Google Colab to write and execute Python in your browser.