Image Segmentation

with Mask R-CNN

Gesina Schwalbe

7th May 2018

Section 1

Definition and Goals

General Goals

- bounding boxes
- classification of each box
- pixel-mask for each box

Datasets

General Goals

- bounding boxes
- classification of each box
- pixel-mask for each box

Datasets

- COCO
- ImageNet

General Goals

- bounding boxes
- classification of each box
- pixel-mask for each box

Datasets

- COCO
- ImageNet

General Goals

- bounding boxes
- classification of each box
- pixel-mask for each box

Datasets

- COCO
- ImageNet

General Goals

- bounding boxes
- classification of each box
- pixel-mask for each box

Datasets

- COCO
- ImageNet

General Goals

- bounding boxes
- classification of each box
- pixel-mask for each box

Datasets

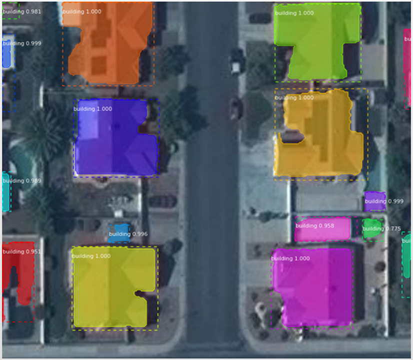
- COCO
- ImageNet

Applications

- Live detection of signs, obstacles in traffic (example video)

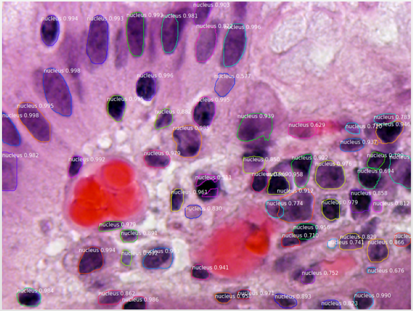
Applications

- Automatic street map enhancement



Applications

- Automatic evaluation of microscopy images



Architectual Requirements

- Master natural images: large nets
- Learn and process fast/efficiently
  (currently best: 32 h wt. 8 GPU learning; 5 fps inference):
  - Special network components
    (CNN, ResNe(X)t, FPN)
  - Share many features (FPN, RPN)
  - Parallelize tasks/components

Architectual Requirements

- Master natural images: large nets
- Learn and process fast/efficiently
  (currently best: 32 h wt. 8 GPU learning; 5 fps inference):
  - Special network components
    (CNN, ResNe(X)t, FPN)
  - Share many features (FPN, RPN)
  - Parallelize tasks/components

Architectural Requirements

- Master natural images: large nets
- Learn and process fast/efficiently
  (currently best: 32 h wt. 8 GPU learning; 5 fps inference):
  - Special network components
    (CNN, ResNe(X)t, FPN)
  - Share many features (FPN, RPN)
  - Parallelize tasks/components

Architectual Requirements

- Master natural images: large nets
- Learn and process fast/efficiently
  (currently best: 32 h wt. 8 GPU learning; 5 fps inference):
  - Special network components
    (CNN, ResNe(X)t, FPN)
  - Share many features (FPN, RPN)
  - Parallelize tasks/components

Architectual Requirements

- Master natural images: large nets
- Learn and process fast/efficiently
  (currently best: 32 h wt. 8 GPU learning; 5 fps inference):
  - Special network components
    (CNN, ResNe(X)t, FPN)
  - Share many features (FPN, RPN)
  - Parallelize tasks/components

Section 2

Architecture Components

What

- Feedforward neural network with only local connections
- Massive weight sharing
- Translation invariance
- (Often) Downscaling of feature space:
  - More translation invariance
  - Multiple scale feature-spaces available

What

- Feedforward neural network with only local connections
- Massive weight sharing
- Translation invariance
- (Often) Downscaling of feature space:
  - More translation invariance
  - Multiple scale feature-spaces available

What

- Feedforward neural network with only local connections
- Massive weight sharing
- Translation invariance
- (Often) Downscaling of feature space:
  - More translation invariance
  - Multiple scale feature-spaces available

What

- Feedforward neural network with only local connections
- Massive weight sharing
- Translation invariance
- (Often) Downscaling of feature space:
  - More translation invariance
  - Multiple scale feature-spaces available

What

- Feedforward neural network with only local connections
- Massive weight sharing
- Translation invariance
- (Often) Downscaling of feature space:
  - More translation invariance
  - Multiple scale feature-spaces available

What

- Feedforward neural network with only local connections
- Massive weight sharing
- Translation invariance
- (Often) Downscaling of feature space:
  - More translation invariance
  - Multiple scale feature-spaces available

Convolution Operation

Linear map $\mathbb{R}^{n_1 \times n_2 \times \cdots n_3} \to \mathbb{R}^{n_1 \times n_2 \times \cdots n_3}$ described by

- a fixed sliding window shape
- a window of that shape with a weight value at each coordinate, called the *kernel*.

(Animation)

Convolution Operation

Linear map $\mathbb{R}^{n_1 \times n_2 \times \cdots n_3} \to \mathbb{R}^{n_1 \times n_2 \times \cdots n_3}$ described by

- a fixed sliding window shape
- a window of that shape with a weight value at each coordinate, called the *kernel*.

(Animation)

Convolution Operation

Linear map $\mathbb{R}^{n_1 \times n_2 \times \cdots n_3} \to \mathbb{R}^{n_1 \times n_2 \times \cdots n_3}$ described by

- a fixed sliding window shape
- a window of that shape with a weight value at each coordinate, called the *kernel*.

(Animation)

Convolution Operation

Linear map $\mathbb{R}^{n_1 \times n_2 \times \cdots n_3} \to \mathbb{R}^{n_1 \times n_2 \times \cdots n_3}$ described by

- a fixed sliding window shape
- a window of that shape with a weight value at each coordinate, called the *kernel*.

(Animation)

Convolutional Layer

Convolution Hyperparameters:

input dimension	without number of filters
size of kernel	= weight-window
padding variant	= border treatment
number of filters	= number of parallel convolutions
stride	= downsampling rate
dilation	
activation function	e.g. linear, ReLu

Convolutional Layer

Convolution Hyperparameters:

input dimension	without number of filters
size of kernel	= weight-window
padding variant	= border treatment
number of filters	= number of parallel convolutions
stride	= downsampling rate
dilation	=
activation function	e.g. linear, ReLu

Convolutional Layer

Convolution Hyperparameters:

input dimension	without number of filters
size of kernel	= weight-window
padding variant	= border treatment
number of filters	= number of parallel convolutions
stride	= downsampling rate
dilation	
activation function	e.g. linear, ReLu

Convolutional Layer

Convolution Hyperparameters:

input dimension	without number of filters
size of kernel	= weight-window
padding variant	= border treatment
number of filters	= number of parallel convolutions
stride	= downsampling rate
dilation	= upsampling rate
activation function	e.g. linear, ReLu

Convolutional Layer

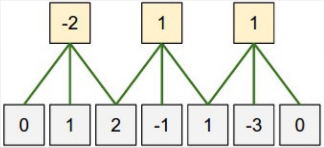
Convolution Hyperparameters:

input dimension	without number of filters
size of kernel	= weight-window
padding variant	= border treatment
number of filters	= number of parallel convolutions
stride	= downsampling rate
dilation	= upsampling rate
activation function	e.g. linear, ReLu

Convolutional Layer

Convolution Hyperparameters:

- **input dimension** without number of filters
- **size of kernel** = weight-window
- **padding variant** = border treatment
- **number of filters** = number of parallel convolutions
- **stride** = downsampling rate



- **dilation** = upsampling rate
- **activation function** e.g. linear, ReLu

Convolutional Layer

Convolution Hyperparameters:

input dimension	without number of filters
size of kernel	= weight-window
padding variant	= border treatment
number of filters	= number of parallel convolutions
stride	= downsampling rate
dilation	= upsampling rate
activation function	e.g. linear, ReLu

Convolutional Layer

Convolution Hyperparameters:

input dimension	without number of filters
size of kernel	= weight-window
padding variant	= border treatment
number of filters	= number of parallel convolutions
stride	= downsampling rate
dilation	= upsampling rate
activation function	e.g. linear, ReLu

Convolutional Layer

Convolution Weights: kernel values, bias

Pooling Layer

Usual pooling functions:

- Average Pooling (linear)
- MaxPooling (non-linear)

Pooling Layer

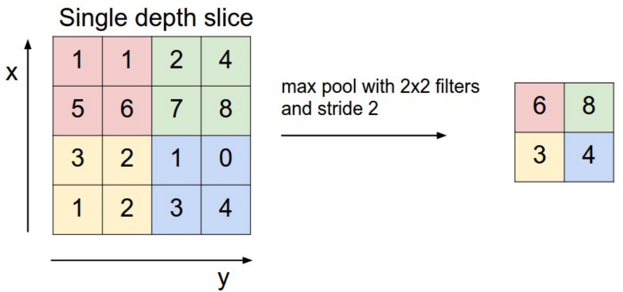
Usual pooling functions:

- Average Pooling (linear)
- MaxPooling (non-linear)

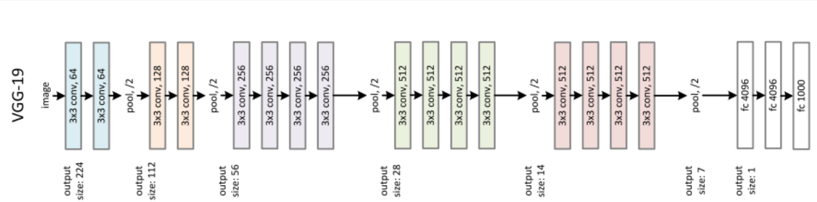
Pooling Layer

Usual pooling functions:

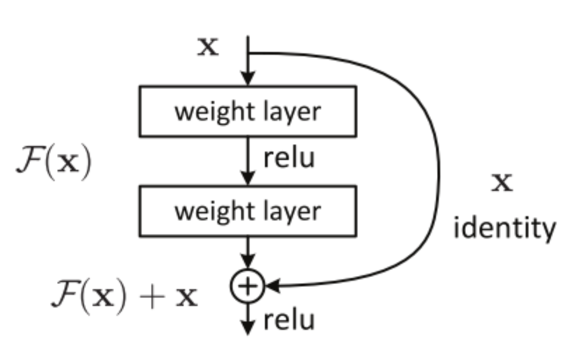
- Average Pooling (linear)
- MaxPooling (non-linear)



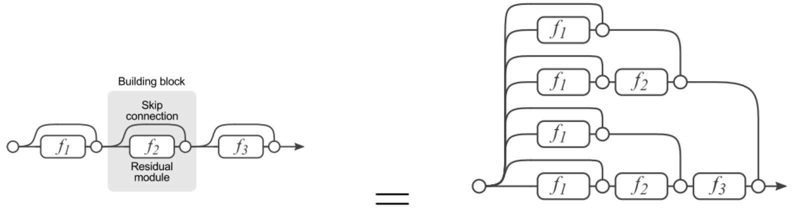
(Deep) CNN


VGG-19
image
3x3 conv, 64
3x3 conv, 64
pool, /2
3x3 conv, 128
3x3 conv, 128
pool, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
fc 4096
fc 4096
fc 1000
output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
output size: 7
output size: 1

Residual CNNs: ResNet

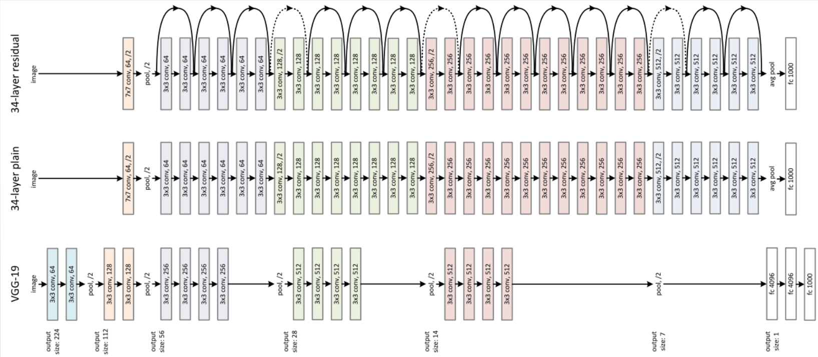


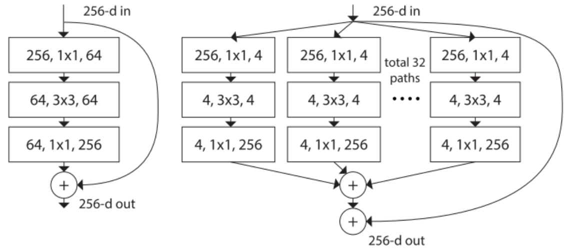
Residual CNNs: ResNet



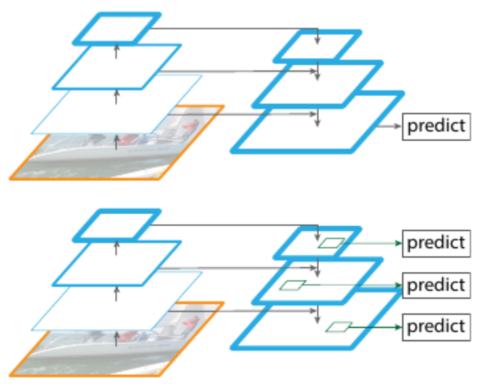
(a) Conventional 3-block residual network

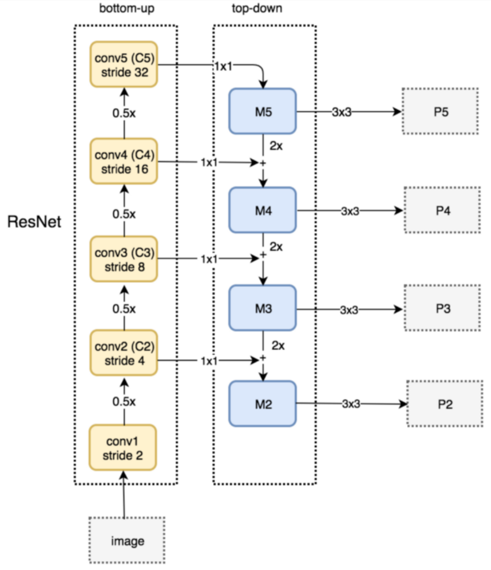
(b) Unraveled view of (a)


34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000
34-layer plain
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000
VGG-19
image
3x3 conv, 64
3x3 conv, 64
pool, /2
3x3 conv, 128
3x3 conv, 128
pool, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
fc 4096
fc 4096
fc 1000
output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
output size: 7
output size: 1


256-d in
256, 1x1, 64
64, 3x3, 64
64, 1x1, 256
+
256-d out
256-d in
256, 1x1, 4
4, 3x3, 4
4, 1x1, 256
256, 1x1, 4
4, 3x3, 4
4, 1x1, 256
total 32 paths
....
256, 1x1, 4
4, 3x3, 4
4, 1x1, 256
+
+
256-d out

More Feature sharing: FPNs




bottom-up
top-down
ResNet
conv5 (C5)
stride 32
0.5x
conv4 (C4)
stride 16
0.5x
conv3 (C3)
stride 8
0.5x
conv2 (C2)
stride 4
0.5x
conv1
stride 2
image
1x1
M5
2x
M4
2x
M3
2x
M2
3x3
P5
P4
P3
P2

Section 3

Model

Predecessor Problems

Object Classification one object per image to one class per image

Object Detection

- (intelligently) choose windows of the image
- classify each window
- maybe enhance the window selection

Predecessor Problems

Object Classification one object per image to one class per image

Object Detection

- (intelligently) choose windows of the image
- classify each window
- maybe enhance the window selection

Predecessor Problems

Object Classification one object per image to one class per image

Object Detection

- (intelligently) choose windows of the image
- classify each window
- maybe enhance the window selection

Predecessor Problems

Object Classification one object per image to one class per image

Object Detection

- (intelligently) choose windows of the image
- classify each window
- maybe enhance the window selection

Predecessor Problems

Object Classification one object per image to one class per image

Object Detection

- (intelligently) choose windows of the image
- classify each window
- maybe enhance the window selection

Components

1. Convolutional backbone: extract important features
2. Region proposal; *in parallel:*
   - Window objectness scoring
3. Frontend; *in parallel:*
   - Classification
   - Masking
   - Bounding Box Optimization

Components

1. Convolutional backbone: extract important features
2. Region proposal; *in parallel:*
   - Window objectness scoring
   - Window correction
3. Frontend; *in parallel:*
   - Classification
   - Masking
   - Bounding Box Optimization

Components

1. Convolutional backbone: extract important features
2. Region proposal; *in parallel:*
   - Window objectness scoring
   - Window correction
3. Frontend; *in parallel:*
   - Classification
   - Masking
   - Bounding Box Optimization

Components

1. Convolutional backbone: extract important features
2. Region proposal; *in parallel:*
   - Window objectness scoring
   - Window correction
3. Frontend; *in parallel:*
   - Classification
   - Masking
   - Bounding Box Optimization

Components

1. Convolutional backbone: extract important features
2. Region proposal; *in parallel:*
   - Window objectness scoring
   - Window correction
3. Frontend; *in parallel:*
   - Classification
   - Masking
   - Bounding Box Optimization

Components

1. Convolutional backbone: extract important features
2. Region proposal; *in parallel:*
   - Window objectness scoring
   - Window correction
3. Frontend; *in parallel:*
   - Classification
   - Masking
   - Bounding Box Optimization

Components

1. Convolutional backbone: extract important features
2. Region proposal; *in parallel:*
    - Window objectness scoring
    - Window correction
3. Frontend; *in parallel:*
    - Classification
    - Masking
    - Bounding Box Optimization

Components

1. Convolutional backbone: extract important features
2. Region proposal; *in parallel:*
    - Window objectness scoring
    - Window correction
3. Frontend; *in parallel:*
    - Classification
    - Masking
    - Bounding Box Optimization

Convolutional Backbone

Goal extract features

Architecture same as for Object Detection

Convolutional Backbone

Goal extract features

Architecture same as for Object Detection

Predecessors/Alternatives

Pixel Merging (e.g. SelectiveSearch)

Window scoring (e.g. Objectness)

Separate NN (e.g. Multibox)

Predecessors/Alternatives

Pixel Merging (e.g. SelectiveSearch)

Window scoring (e.g. Objectness)

Separate NN (e.g. Multibox)

Predecessors/Alternatives

Pixel Merging (e.g. SelectiveSearch)

Window scoring (e.g. Objectness)

Separate NN (e.g. Multibox)

Main Ideas of the Mask R-CNN/Faster R-CNN Approach

Window scoring with enhancements:

Decoupling of classification and window proposals

All Scales at once using different candidate window shapes

Bounding box correction in *parallel* to scoring

Excessive weight sharing amongst same shapes

RoI-Pooling i.e. use convolutional features from backbone and pool each proposal window to fixed size

Main Ideas of the Mask R-CNN/Faster R-CNN Approach

Window scoring with enhancements:

Decoupling of classification and window proposals

All Scales at once using different candidate window shapes

Bounding box correction in *parallel* to scoring

Excessive weight sharing amongst same shapes

RoI-Pooling i.e. use convolutional features from backbone and pool each proposal window to fixed size

Main Ideas of the Mask R-CNN/Faster R-CNN Approach

Window scoring with enhancements:

Decoupling of classification and window proposals

All Scales at once using different candidate window shapes

Bounding box correction in *parallel* to scoring

Excessive weight sharing amongst same shapes

RoI-Pooling i.e. use convolutional features from backbone and pool each proposal window to fixed size

Main Ideas of the Mask R-CNN/Faster R-CNN Approach

Window scoring with enhancements:

Decoupling of classification and window proposals

All Scales at once using different candidate window shapes

Bounding box correction in *parallel* to scoring

Excessive weight sharing amongst same shapes

RoI-Pooling i.e. use convolutional features from backbone and pool each proposal window to fixed size

Main Ideas of the Mask R-CNN/Faster R-CNN Approach

Window scoring with enhancements:

Decoupling of classification and window proposals

All Scales at once using different candidate window shapes

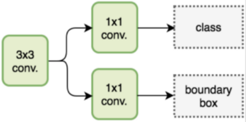
Bounding box correction in *parallel* to scoring

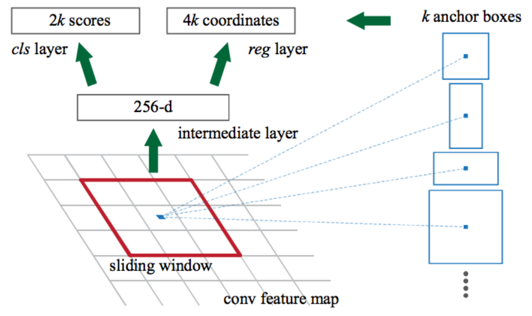
Excessive weight sharing amongst same shapes

RoI-Pooling i.e. use convolutional features from backbone and pool each proposal window to fixed size

Architecture Overview

Shared Conv Layer Sliding window on feature space with fixed set of anchor shapes per window





Architecture Overview

Shared Conv Layer Sliding window on feature space with fixed set of anchor shapes per window

cls*, *reg Per window and anchor shape do in parallel

Architecture Overview

Shared Conv Layer Sliding window on feature space with fixed set of anchor shapes per window

cls*, *reg Per window and anchor shape do in parallel

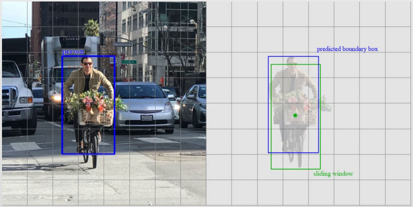
objectness score (*cls*)

Architecture Overview

Shared Conv Layer Sliding window on feature space with fixed set of anchor shapes per window

cls, *reg* Per window and anchor shape do in parallel

objectness score (*cls*)
coordinate correction (*reg*)



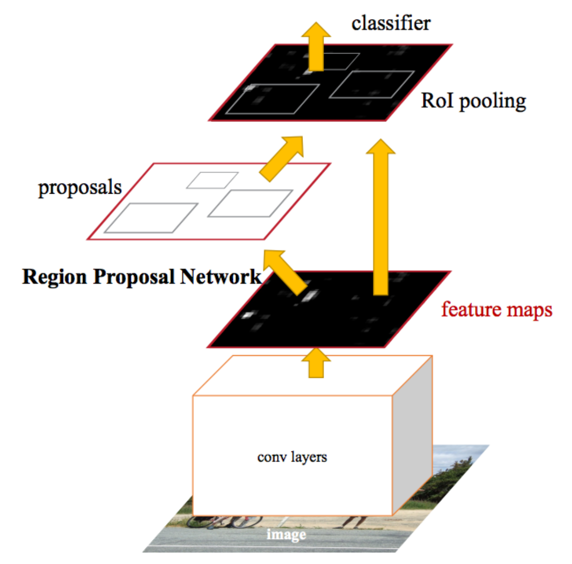
Architecture Overview

Shared Conv Layer Sliding window on feature space with fixed set of anchor shapes per window

cls*, *reg Per window and anchor shape do in parallel

- objectness score (*cls*)
- coordinate correction (*reg*)

Proposal Layer Select best proposals


classifier
RoI pooling
proposals
Region Proposal Network
feature maps
conv layers
image

Coordinate Encoding

Box coordinate (= Proposal Layer output) encoding, all normalized:

$$\begin{pmatrix} x_1, y_1 & \texttt{\# upper left corner} \\ x_2, y_2 & \texttt{\# lower right corner} \end{pmatrix}$$

Coordinate Encoding

Coordinate correction (=*reg* output) encoding:

$$\begin{aligned}
\Big(\, & dx, && \# \ \frac{\texttt{centerpoint x-difference}}{\texttt{anchor width}} \\
& dy, && \# \ \frac{\texttt{centerpoint y-difference}}{\texttt{anchor height}} \\
& \log(dw), && \# \ \log\left(\frac{\texttt{ground-truth box width}}{\texttt{anchor width}}\right) \\
& \log(dh)\,\Big) && \# \ \log\left(\frac{\texttt{ground-truth box height}}{\texttt{anchor height}}\right)
\end{aligned}$$

Metrics

Metric for matching:

$$\mathbf{IoU}(A, B) := \frac{\text{Intersection Area}}{\text{Union Area}}$$

Labels

1: object with ground-truth box b if

- it has best **IoU** for b, else if
- **IoU** for $b \geq 0.7$

-1: no object if not positive and **IoU** ≤ 0.3

0: neutral else (excluded from training)

Balance positive and negative boxes for training!

Labels

1: object with ground-truth box b if

- it has best **IoU** for b, else if
- **IoU** for $b \geq 0.7$

-1: no object if not positive and **IoU** ≤ 0.3

0: neutral else (excluded from training)

Balance positive and negative boxes for training!

Labels

1: object with ground-truth box b if
- it has best **IoU** for b, else if
- **IoU** for $b \geq 0.7$

-1: no object if not positive and **IoU** ≤ 0.3

0: neutral else (excluded from training)

Balance positive and negative boxes for training!

Labels

1: object with ground-truth box b if

- it has best **IoU** for b, else if
- **IoU** for $b \geq 0.7$

-1: no object if not positive and **IoU** ≤ 0.3

0: neutral else (excluded from training)

Balance positive and negative boxes for training!

Labels

1: object with ground-truth box b if
- it has best **IoU** for b, else if
- **IoU** for $b \geq 0.7$

-1: no object if not positive and **IoU** ≤ 0.3

0: neutral else (excluded from training)

Balance positive and negative boxes for training!

Architecture Details

Sliding window Shared Conv layer with “valid”-padding

Architecture Details

Objectness classification Conv layer with

- 1×1-sized kernel
- 2-class softmax activation: (non-object score, object score)

Loss: crossentropy for non-neutral anchors

Architecture Details

Coordinate correction Conv layer with

- 1×1-sized kernel
- $4 \times$ (number of anchors) filters: (dx, dy, dw, dh) coordinate correction for each anchor

Loss: smooth L_1-loss
for positive anchors that do not cross image bounds

Architecture Details

Proposal selection

1. Trim to N best-object-scored anchors.
2. Apply coordinate correction.
3. Clip boxes.
4. Non-maximum suppression:
   - Sort by score.
   - Reject boxes which have high IoU with better scored ones.
   - Trim to best `num_proposals` proposals.

Architecture Details

Proposal selection

1. Trim to N best-object-scored anchors.
2. Apply coordinate correction.
3. Clip boxes.
4. Non-maximum suppression:
   - Sort by score.
   - Reject boxes which have high IoU with better scored ones.
   - Trim to best `num_proposals` proposals.

Architecture Details

Proposal selection

1. Trim to N best-object-scored anchors.
2. Apply coordinate correction.
3. Clip boxes.
4. Non-maximum suppression:
   - Sort by score.
   - Reject boxes which have high IoU with better scored ones.
   - Trim to best `num_proposals` proposals.

Architecture Details

Proposal selection

1. Trim to N best-object-scored anchors.
2. Apply coordinate correction.
3. Clip boxes.
4. Non-maximum suppression:
   1. Sort by score.
   2. Reject boxes which have high IoU with better scored ones.
   3. Trim to best `num_proposals` proposals.

Architecture Details

Proposal selection

1. Trim to N best-object-scored anchors.
2. Apply coordinate correction.
3. Clip boxes.
4. Non-maximum suppression:
   1. Sort by score.
   2. Reject boxes which have high IoU with better scored ones.
   3. Trim to best `num_proposals` proposals.

Architecture Details

Proposal selection

1. Trim to N best-object-scored anchors.
2. Apply coordinate correction.
3. Clip boxes.
4. Non-maximum suppression:
    1. Sort by score.
    2. Reject boxes which have high IoU with better scored ones.
    3. Trim to best `num_proposals` proposals.

Architecture Details

Proposal selection

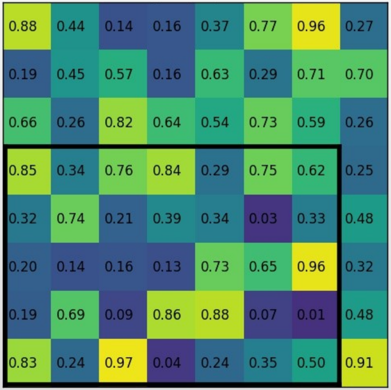
1. Trim to N best-object-scored anchors.
2. Apply coordinate correction.
3. Clip boxes.
4. Non-maximum suppression:
   1. Sort by score.
   2. Reject boxes which have high IoU with better scored ones.
   3. Trim to best `num_proposals` proposals.

Architecture Details

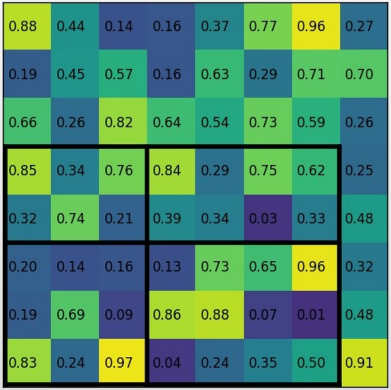
Proposal selection

1. Trim to N best-object-scored anchors.
2. Apply coordinate correction.
3. Clip boxes.
4. Non-maximum suppression:
   1. Sort by score.
   2. Reject boxes which have high IoU with better scored ones.
   3. Trim to best `num_proposals` proposals.

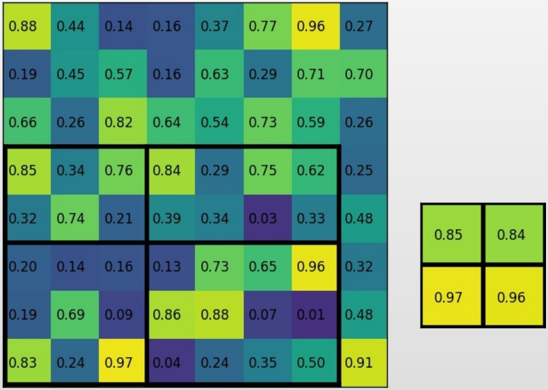
RoI-Pooling


0.88 0.44 0.14 0.16 0.37 0.77 0.96 0.27
0.19 0.45 0.57 0.16 0.63 0.29 0.71 0.70
0.66 0.26 0.82 0.64 0.54 0.73 0.59 0.26
0.85 0.34 0.76 0.84 0.29 0.75 0.62 0.25
0.32 0.74 0.21 0.39 0.34 0.03 0.33 0.48
0.20 0.14 0.16 0.13 0.73 0.65 0.96 0.32
0.19 0.69 0.09 0.86 0.88 0.07 0.01 0.48
0.83 0.24 0.97 0.04 0.24 0.35 0.50 0.91

RoI-Pooling

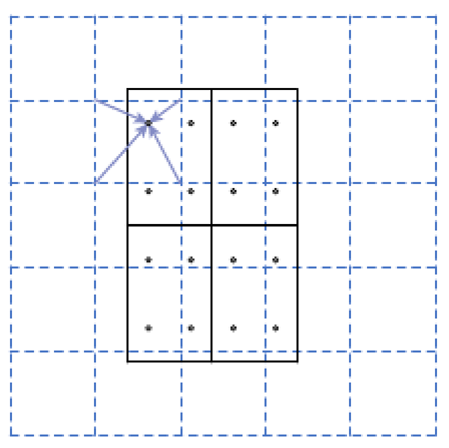

0.88 0.44 0.14 0.16 0.37 0.77 0.96 0.27
0.19 0.45 0.57 0.16 0.63 0.29 0.71 0.70
0.66 0.26 0.82 0.64 0.54 0.73 0.59 0.26
0.85 0.34 0.76 0.84 0.29 0.75 0.62 0.25
0.32 0.74 0.21 0.39 0.34 0.03 0.33 0.48
0.20 0.14 0.16 0.13 0.73 0.65 0.96 0.32
0.19 0.69 0.09 0.86 0.88 0.07 0.01 0.48
0.83 0.24 0.97 0.04 0.24 0.35 0.50 0.91

RoI-Pooling


0.88 0.44 0.14 0.16 0.37 0.77 0.96 0.27
0.19 0.45 0.57 0.16 0.63 0.29 0.71 0.70
0.66 0.26 0.82 0.64 0.54 0.73 0.59 0.26
0.85 0.34 0.76 0.84 0.29 0.75 0.62 0.25
0.32 0.74 0.21 0.39 0.34 0.03 0.33 0.48
0.20 0.14 0.16 0.13 0.73 0.65 0.96 0.32
0.19 0.69 0.09 0.86 0.88 0.07 0.01 0.48
0.83 0.24 0.97 0.04 0.24 0.35 0.50 0.91
0.85 0.84
0.97 0.96

RoI-Align

Bilinear interpolation instead of cropping:



Main Ideas

Decouple

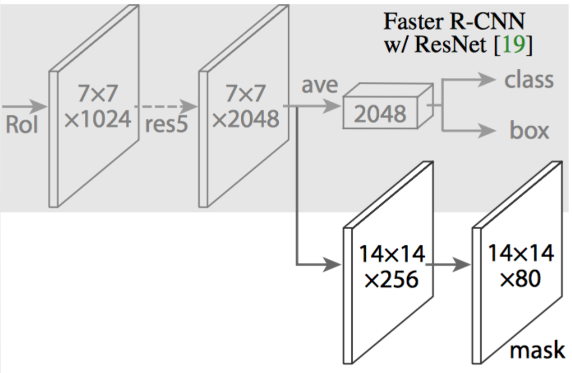
- classification, bounding box optimization, and masks;
- mask predictions for the different classes

Main Ideas

Decouple

- classification, bounding box optimization, and masks;
- mask predictions for the different classes

Architecture



Architecture

Classification fully connected layers ending in softmax (include class "no object")
Loss: multinomial crossentropy

Architecture

Mask generation

1. Few (1–3) Conv Layers, maybe with upscaling parts
2. Conv Layer with
   - a filter for each class
   - sigmoid activation

Loss: binary cross-entropy

Architecture

Bounding box regression Linear regression

Section 4

Training

Steps

1. Backbone
2. RPN
3. Alternating training of RPN and Frontend

Steps

1. Backbone
2. RPN
3. Alternating training of RPN and Frontend

Steps

1. Backbone
2. RPN
3. Alternating training of RPN and Frontend

Backbone Pretraining

Separate ConvNet Training Model

1. Backbone (Conv & Pooling)
2. Dense Layers
3. Classification Softmax-Layer

Training Input

Backbone Pretraining

Separate ConvNet Training Model

1. Backbone (Conv & Pooling)
2. Dense Layers
3. Classification Softmax-Layer

Training Input

Backbone Pretraining

Separate ConvNet Training Model

1. Backbone (Conv & Pooling)
2. Dense Layers
3. Classification Softmax-Layer

Training Input

inputs: one-object images (ca. the size of later bounding boxes)

the single objects' labels

Backbone Pretraining

Separate ConvNet Training Model

1. Backbone (Conv & Pooling)
2. Dense Layers
3. Classification Softmax-Layer

Training Input

inputs one-object images (ca. the size of later bounding boxes)

labels the single objects' labels

Backbone Pretraining

Separate ConvNet Training Model

1. Backbone (Conv & Pooling)
2. Dense Layers
3. Classification Softmax-Layer

Training Input

inputs one-object images (ca. the size of later bounding boxes)

labels the single objects' labels

Backbone Pretraining

Separate ConvNet Training Model

1. Backbone (Conv & Pooling)
2. Dense Layers
3. Classification Softmax-Layer

Training Input

inputs one-object images (ca. the size of later bounding boxes)

labels the single objects' labels

Backbone Pretraining

Separate ConvNet Training Model

1. Backbone (Conv & Pooling)
2. Dense Layers
3. Classification Softmax-Layer

Training Input

inputs one-object images (ca. the size of later bounding boxes)

labels the single objects' labels

Alternating Training

(after RPN is pretrained)

1. Frontend (RPN fixed, backbone not shared)
2. RPN (frontend fixed, shared backbone fixed)
3. Frontend (RPN fixed, shared backbone fixed)
4. ...

Alternating Training

(after RPN is pretrained)

1. Frontend (RPN fixed, backbone not shared)
2. RPN (frontend fixed, shared backbone fixed)
3. Frontend (RPN fixed, shared backbone fixed)
4. ...

Alternating Training

(after RPN is pretrained)

1. Frontend (RPN fixed, backbone not shared)
2. RPN (frontend fixed, shared backbone fixed)
3. Frontend (RPN fixed, shared backbone fixed)
4. ...

Alternating Training

(after RPN is pretrained)

1. Frontend (RPN fixed, backbone not shared)
2. RPN (frontend fixed, shared backbone fixed)
3. Frontend (RPN fixed, shared backbone fixed)
4. ...

Section 5

Implementation Review

5 Implementation Review
- Source Code
- Lessions learned

Example Sources

- Caffe2 reference implementation by facebookresearch: [1]
- Keras implementation by matterport: [2]
- Example for handwritten number detection using autogenerated data based on MNIST: github

Example Sources

- Caffe2 reference implementation by facebookresearch: [1]
- Keras implementation by matterport: [2]
- Example for handwritten number detection using autogenerated data based on MNIST: github

Example Sources

- Caffe2 reference implementation by facebookresearch: [1]
- Keras implementation by matterport: [2]
- Example for handwritten number detection using autogenerated data based on MNIST: github

Keras Implementation Specialties

- Use the functional API!
- Custom layers:
  - Loss Layers: custom losses
  - Proposal Layer: select RPN proposals
  - reshape proposals and mask labels
    use tensorflow function
    `tf.crop_and_resize()`)
- Separate models for training and inference

Keras Implementation Specialties

- Use the functional API!
- Custom layers:

Loss Layers	custom losses
Proposal Layer	select RPN proposals
RoI-Pooling Layer	reshape proposals and mask labels (e.g. use tensorflow function `tf.crop_and_resize()`)

- Separate models for training and inference

Keras Implementation Specialties

- Use the functional API!
- Custom layers:

Loss Layers	custom losses
Proposal Layer	select RPN proposals
RoI-Pooling Layer	reshape proposals and mask labels (e.g. use tensorflow function `tf.crop_and_resize()`)

- Separate models for training and inference

Lessions Learned I

- Always double-check and note down tensor/array dimensions; mind the padding for convolutions.
- Always have a look at *all* input and output data:
  - Do they roughly make sense, e.g. do positive classes have different probability output than negative ones?)
  - Are there error patterns?

  Visualization is your friend. But it will definitely contain bugs, too.
- Always double-check and test your algorithms.
- Always double-check your parameters: Do they fit for your data?

Lessions Learned II

- Convolution is very geometric: Depict your sliding windows, anchor shapes, and downscaling factor(s) to check whether they make sense with your data/object sizes.
- Directly document and update all input and output formats of functions—with a good IDE this makes your life much easier. (And, of course, follow the master rule: Keep your code clean and understandable.)
- Mind your RAM when optimizing data generation/tagging.
- Check your loss and validation values; Does the model actually get better?
- The deeper the network, the *much* more time training will take on CPU.

Lessions Learned III

- Make data as easy (small) as possible; if a human can still classify the data in that format, the network can do so, too. E.g. grayscale instead of several color channels.