Introduction to Computer Vision: Object Localization, Detection and Semantic Segmentation

Navasardyan Shant

December 19, 2019

Overview

Overview

Introduction

While *image classification* labels the image depending on the object it contains, *object localization* finds the position of this object. In the case when there are many objects in the image, the task of determining the positions of these objects is called *object detection*.

Introduction

While *image classification* labels the image depending on the object it contains, *object localization* finds the position of this object. In the case when there are many objects in the image, the task of determining the positions of these objects is called *object detection*.

Definition (Object Localization and Detection)

By **determining the position of an object** we mean finding the coordinates of the minimal rectangle, which contains the object.

Introduction

While *image classification* labels the image depending on the object it contains, *object localization* finds the position of this object. In the case when there are many objects in the image, the task of determining the positions of these objects is called *object detection*.

Definition (Object Localization and Detection)

By **determining the position of an object** we mean finding the coordinates of the minimal rectangle, which contains the object.

- Determining the position of one (only) object in an image with a label is called **object localization**,

Introduction

While *image classification* labels the image depending on the object it contains, *object localization* finds the position of this object. In the case when there are many objects in the image, the task of determining the positions of these objects is called *object detection*.

Definition (Object Localization and Detection)

By **determining the position of an object** we mean finding the coordinates of the minimal rectangle, which contains the object.

- Determining the position of one (only) object in an image with a label is called **object localization**,
- Determining the positions of all objects (and labeling them) in an image is called **object detection**.

So the difference is the **number** of objects.

Introduction

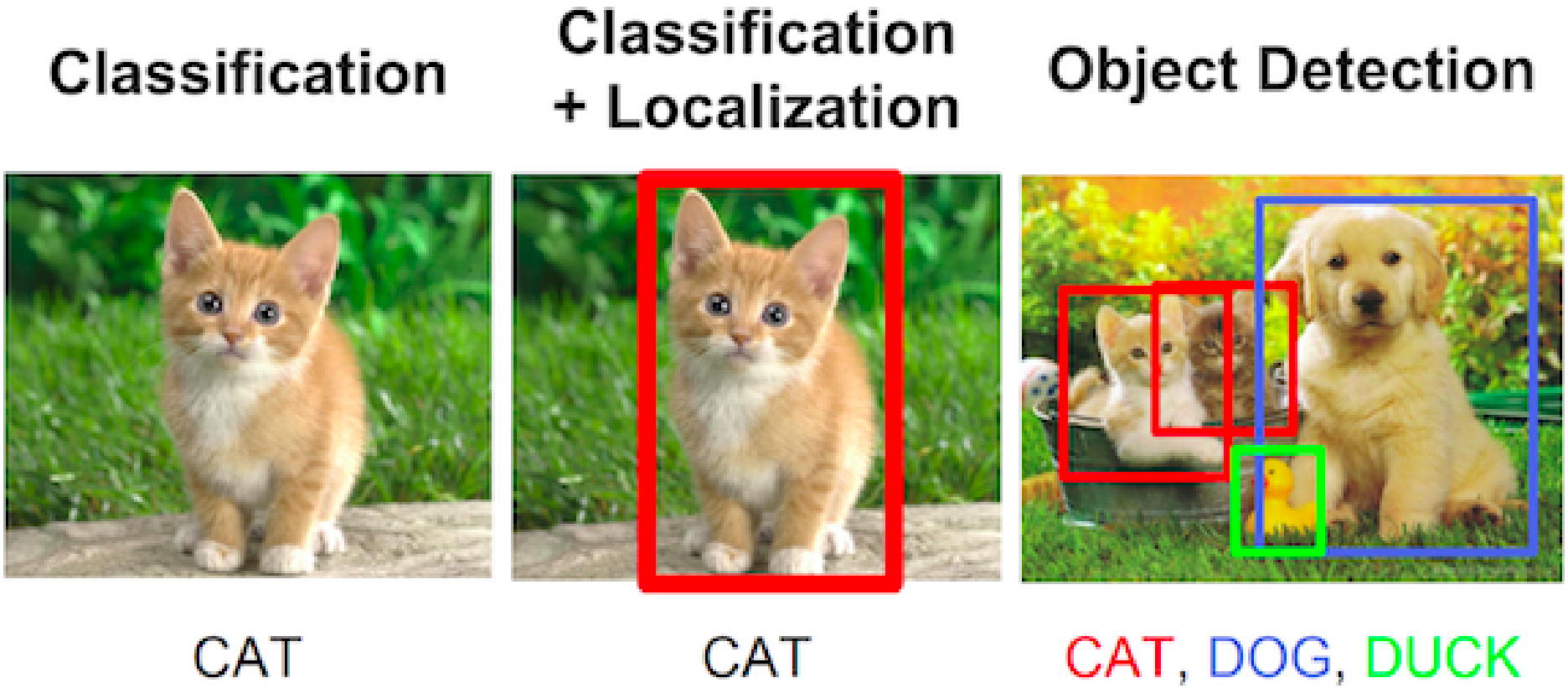


Figure: In this image you can see the difference between image classification, localization and object detection

Intersection over Union (IoU)

Before starting localization and detection algorithms, we define an important measure between two *"regions in the image"*.

Intersection over Union (IoU)

Before starting localization and detection algorithms, we define an important measure between two *"regions in the image"*. In general these regions can be free-form regions and can be given by two binary masks. In our case these regions will be two *bounding boxes*, i.e. rectangles with the sides parallel to the image sides. One can easily generalize the definition of the intersection over union for any free-form regions.

Intersection over Union (IoU)

Before starting localization and detection algorithms, we define an important measure between two *"regions in the image"*. In general these regions can be free-form regions and can be given by two binary masks. In our case these regions will be two *bounding boxes*, i.e. rectangles with the sides parallel to the image sides. One can easily generalize the definition of the intersection over union for any free-form regions.

Definition (Intersection over Union)

Let B_1 and B_2 are two bounding boxes in the image. Then[a]

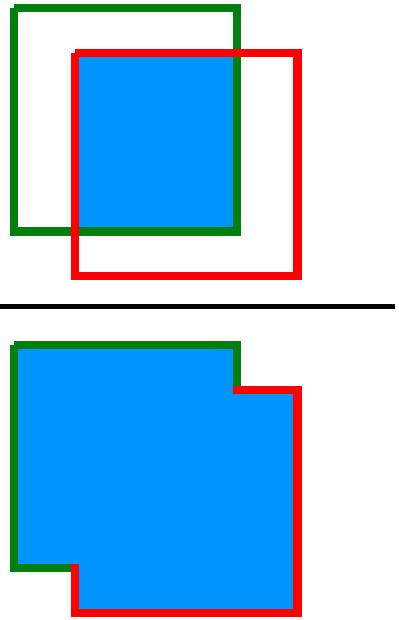
$$IoU(B_1, B_2) = \frac{area(B_1 \cap B_2)}{area(B_1 \cup B_2)}$$

is called the **Intersection over Union** between B_1 and B_2.

[a] By $area(X)$ we mean the area of the region X.

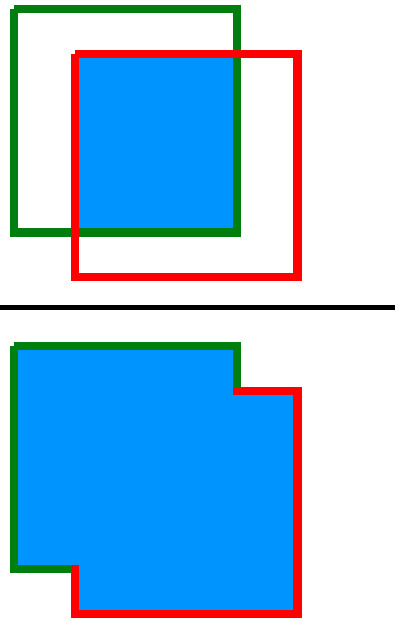
Intersection over Union

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} =$$



Intersection over Union

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} =$$



Question

Let the bounding boxes B_1 and B_2 are given by coordinates of their top-left, bottom-right angles:

$$B_1 = ((t_1, l_1), (b_1, r_1)), \quad B_2 = ((t_2, l_2), (b_2, r_2)).$$

How the $IoU(B_1, B_2)$ can be calculated in terms of these coordinates?

Overview

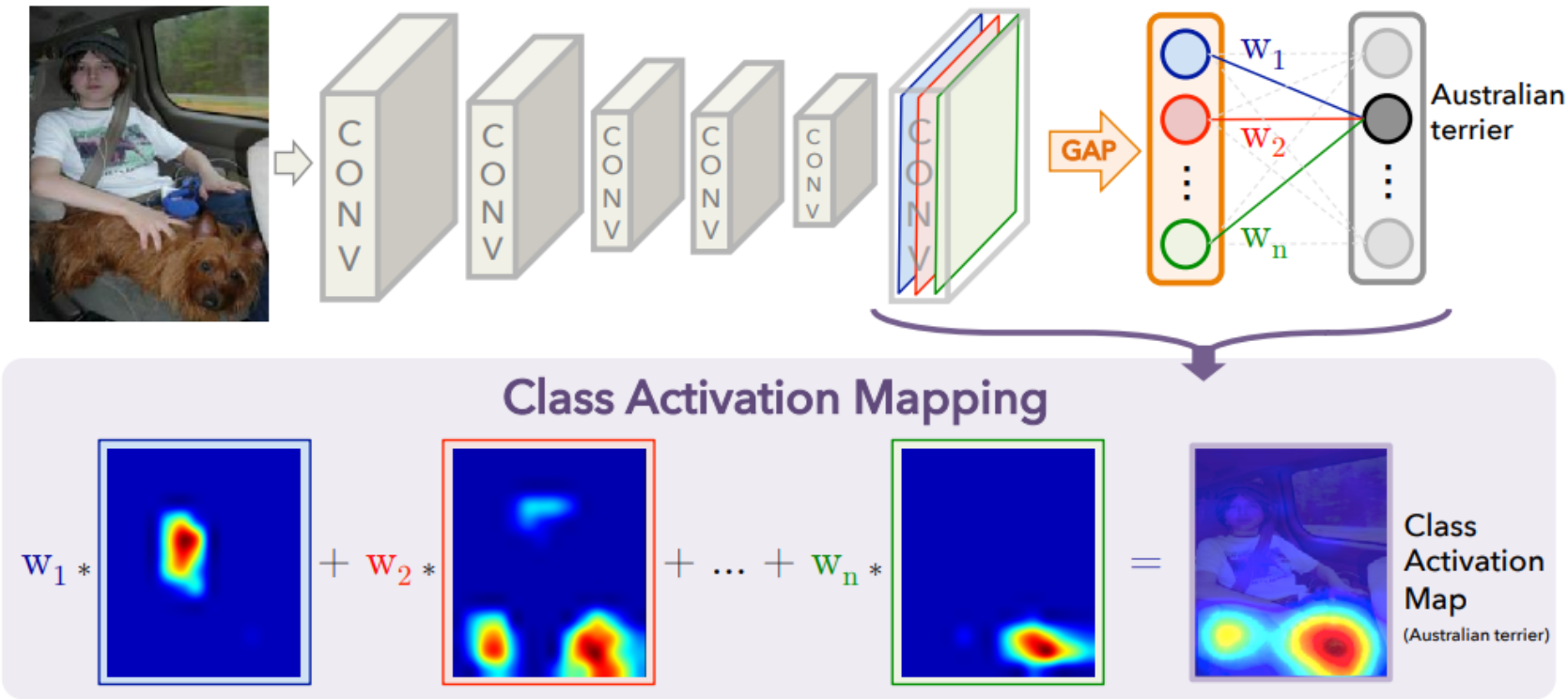
Object Localization with Class Activation Mapping (CAM)

The deep layers of image classification networks have the ability to *activate* (have higher values) on pixels of *"more important parts"* of the objects in the image. These activations can be used for localizing the objects in images. We will consider an algorithm[1], which uses the concept of *Class Activation Mapping*.

[1] Zhou, Bolei, Aditya Khosla, Àgata Lapedriza, Aude Oliva and Antonio Torralba. "Learning Deep Features for Discriminative Localization." 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015): 2921-2929.

Class Activation Mapping

Let's consider an image classification network, say AlexNet, Vgg, or GoogLeNet. We will replace the part with the fully-connected layers before the final output with the *Global Average Pooling (GAP)* layer followed by a fully-connected layer and Softmax. The concept of *Class Activation Mapping* is illustrated in the image below.



Class Activation Mapping

Formally, let's after the last convolution block (here convolution block consists of a convolution followed by an activation layer, also, optionally, a normalization layer can be between the convolution and the activation layer) of the chosen classification network for a fixed image we get the feature $f \in \mathbb{R}^{H \times W \times K}$.

Class Activation Mapping

Formally, let's after the last convolution block (here convolution block consists of a convolution followed by an activation layer, also, optionally, a normalization layer can be between the convolution and the activation layer) of the chosen classification network for a fixed image we get the feature $f \in \mathbb{R}^{H\times W\times K}$. Let's denote by f_k the k^{th} feature-map of f $(k = 0, \ldots, K-1)$. Then, after the Global Average Pooling layer we will get the vector $F \in \mathbb{R}^K$, such that

$$F_k = \frac{1}{H \cdot W} \sum_{x=0}^{H-1} \sum_{y=0}^{W-1} f_k(x, y).$$

Class Activation Mapping

Formally, let's after the last convolution block (here convolution block consists of a convolution followed by an activation layer, also, optionally, a normalization layer can be between the convolution and the activation layer) of the chosen classification network for a fixed image we get the feature $f \in \mathbb{R}^{H\times W\times K}$. Let's denote by f_k the k^{th} feature-map of f $(k = 0, \ldots, K-1)$. Then, after the Global Average Pooling layer we will get the vector $F \in \mathbb{R}^K$, such that

$$F_k = \frac{1}{H \cdot W} \sum_{x=0}^{H-1} \sum_{y=0}^{W-1} f_k(x, y).$$

Then, after a fully connected layer with number of units equal to the number of classes, say C, we will get a vector $S \in \mathbb{R}^C$, such that for every $c = 0, \ldots, C-1$

$$S_c = \sum_{k=0}^{K-1} \omega_k^c F_k.$$

Class Activation Mapping

Then, finally, the softmax layer is applied to get per-class probabilities for the object in the image:

$$Softmax_c = \frac{e^{S_c}}{\sum_{c'} e^{S_{c'}}}.$$

Class Activation Mapping

Then, finally, the softmax layer is applied to get per-class probabilities for the object in the image:

$$Softmax_c = \frac{e^{S_c}}{\sum_{c'} e^{S_{c'}}}.$$

So we get

$$S_c = \sum_{k=0}^{K-1} \omega_k^c F_k = \frac{1}{HW} \sum_{k=0}^{K-1} \omega_k^c \sum_{x,y} f_k(x,y)$$

Class Activation Mapping

Then, finally, the softmax layer is applied to get per-class probabilities for the object in the image:

$$Softmax_c = \frac{e^{S_c}}{\sum_{c'} e^{S_{c'}}}.$$

So we get

$$S_c = \sum_{k=0}^{K-1} \omega_k^c F_k = \frac{1}{HW} \sum_{k=0}^{K-1} \omega_k^c \sum_{x,y} f_k(x,y)$$

$$= \frac{1}{HW} \sum_{x,y} \sum_{k=0}^{K-1} \omega_k^c f_k(x,y).$$

Class Activation Mapping

Then, finally, the softmax layer is applied to get per-class probabilities for the object in the image:

$$Softmax_c = \frac{e^{S_c}}{\sum_{c'} e^{S_{c'}}}.$$

So we get

$$S_c = \sum_{k=0}^{K-1} \omega_k^c F_k = \frac{1}{HW} \sum_{k=0}^{K-1} \omega_k^c \sum_{x,y} f_k(x,y)$$

$$= \frac{1}{HW} \sum_{x,y} \sum_{k=0}^{K-1} \omega_k^c f_k(x,y).$$

Let's denote by $M_c = \sum_k \omega_k^c f_k$. M_c is called the **class activation map** for the class c. Then we get

$$S_c = \frac{1}{HW} \sum_{x,y} M_c(x,y).$$

Class Activation Mapping

Note

S can be obtained with the Global Average Pooling layer on the tensor $M = (M_c)_{c=0}^{C-1} \in \mathbb{R}^{H \times W \times C}$.

Class Activation Mapping

Note

S can be obtained with the Global Average Pooling layer on the tensor $M = (M_c)_{c=0}^{C-1} \in \mathbb{R}^{H \times W \times C}$.

Since S is the result of the last fully-connected layer, the activations in the feature-maps of M will determine the output probability distribution.

Class Activation Mapping

Note

S can be obtained with the Global Average Pooling layer on the tensor $M = (M_c)_{c=0}^{C-1} \in \mathbb{R}^{H \times W \times C}$.

Since S is the result of the last fully-connected layer, the activations in the feature-maps of M will determine the output probability distribution.

The visual patterns are present in the feature f also. In the feature maps M_c we just take the weighted linear combination of the feature-maps f_k, to get directly corresponding class activation maps.

Class Activation Mapping

Here are some results of visualizing the class activation maps M_c.

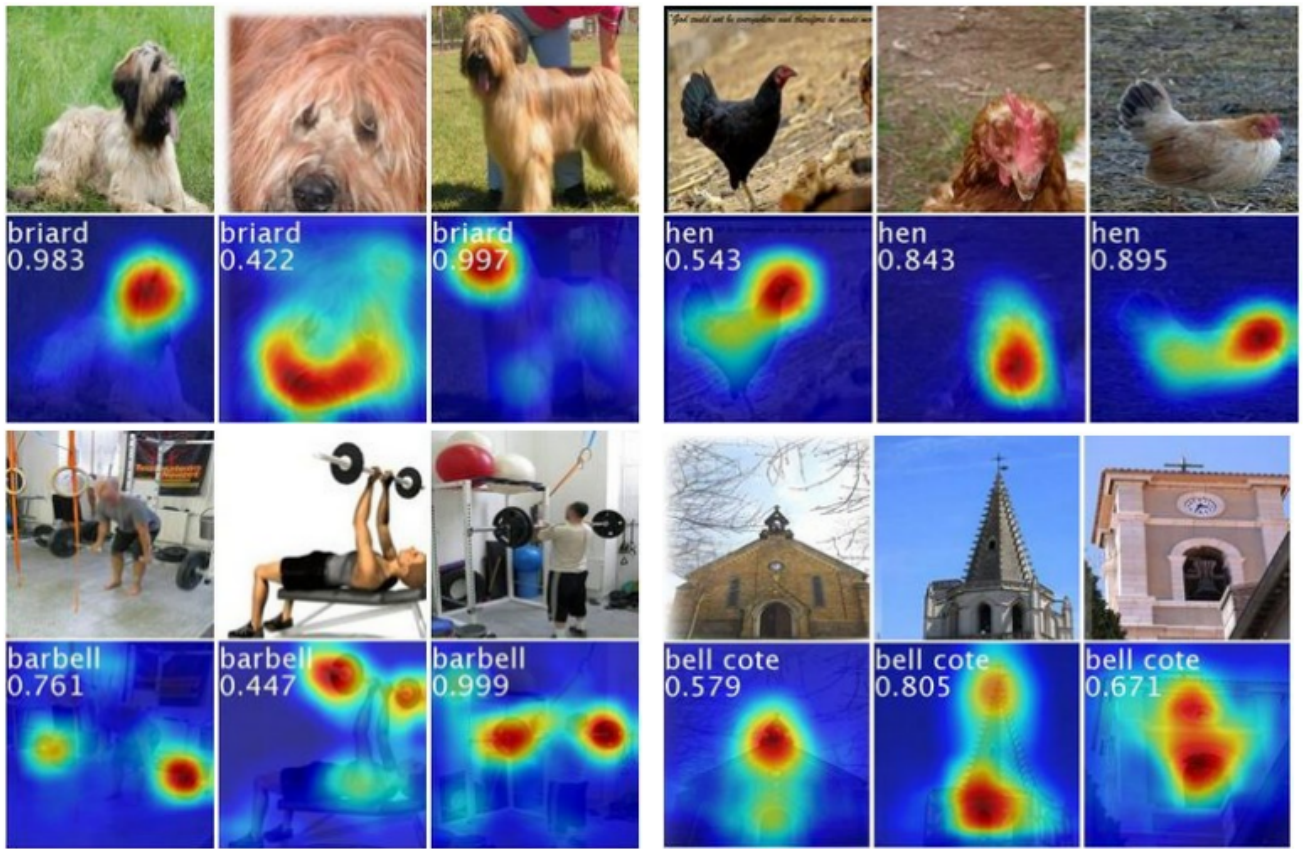
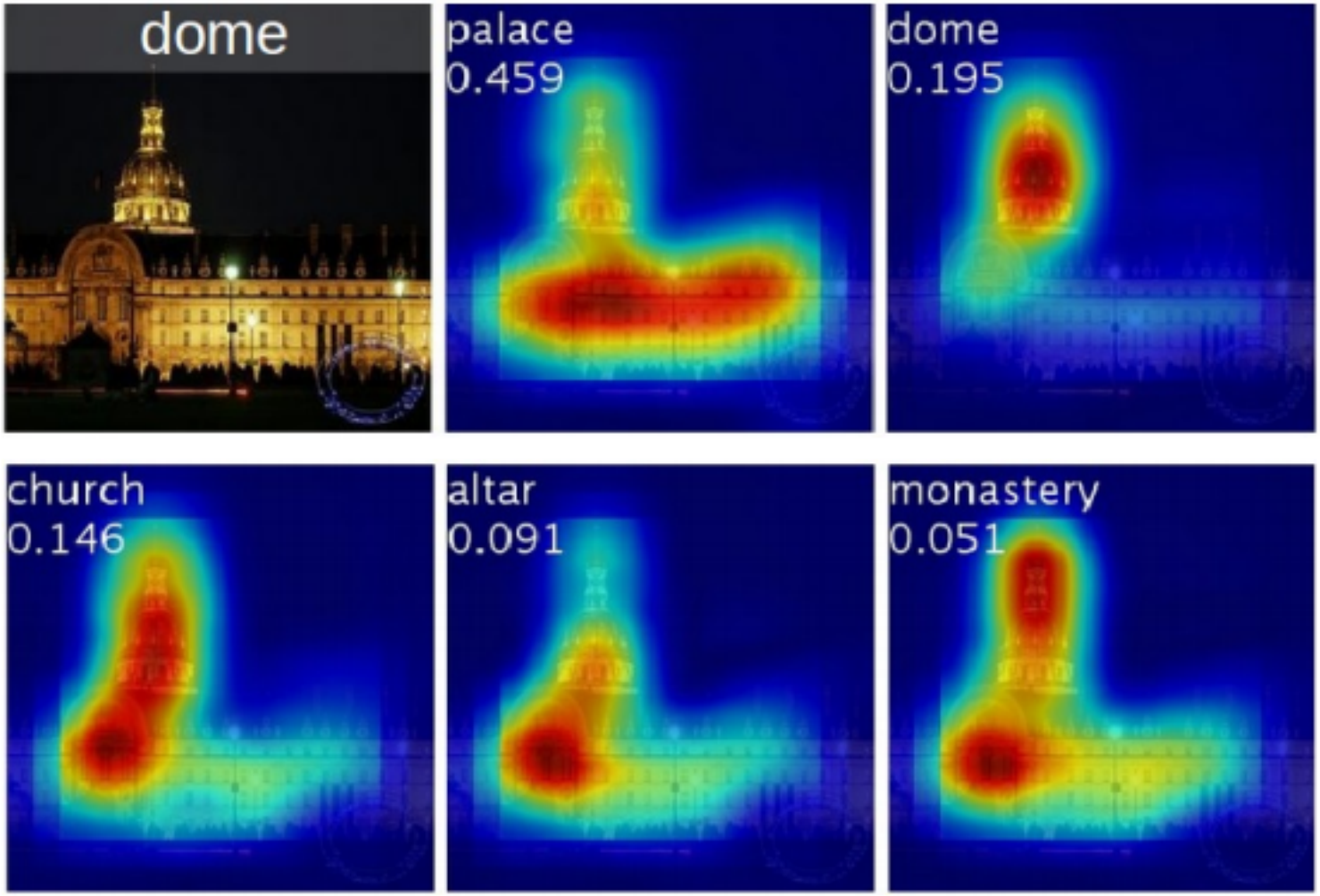


Figure: For each of these four classes are chosen three examples and their corresponding class activation maps are visualized. In order to visualize, the activation maps M_c are upscaled to the original image sizes.

Class Activation Mapping

Also, the class activation maps for other classes can be visualized:



Object Localization with Class Activation Maps

For an image I, if one wants to localize an object of the class c, it can be done as follows.

Object Localization with Class Activation Maps

For an image I, if one wants to localize an object of the class c, it can be done as follows.

- Take the activation map $M_p \in \mathbb{R}^{H \times W \times 1}$.

Object Localization with Class Activation Maps

For an image I, if one wants to localize an object of the class c, it can be done as follows.

- Take the activation map $M_p \in \mathbb{R}^{H \times W \times 1}$.
- Resize M_c to the original size of the image.

Object Localization with Class Activation Maps

For an image I, if one wants to localize an object of the class c, it can be done as follows.

- Take the activation map $M_p \in \mathbb{R}^{H \times W \times 1}$.
- Resize M_c to the original size of the image.
- Determine the pixels which have value larger than a threshold t (in the original paper the threshold t is defined as $0.2max(M_c)$.) and label them with 1s. The rest of pixels label with 0s, and obtain a binary mask $\hat{M}_c$.

Object Localization with Class Activation Maps

For an image I, if one wants to localize an object of the class c, it can be done as follows.

- Take the activation map $M_p \in \mathbb{R}^{H \times W \times 1}$.
- Resize M_c to the original size of the image.
- Determine the pixels which have value larger than a threshold t (in the original paper the threshold t is defined as $0.2max(M_c).$) and label them with 1s. The rest of pixels label with 0s, and obtain a binary mask $\hat{M}_c$.
- Determine the biggest connected component in the binary mask $\hat{M}_c$, and take it into its bounding box.

Object Localization with Class Activation Maps

For an image I, if one wants to localize an object of the class c, it can be done as follows.

- Take the activation map $M_p \in \mathbb{R}^{H \times W \times 1}$.
- Resize M_c to the original size of the image.
- Determine the pixels which have value larger than a threshold t (in the original paper the threshold t is defined as $0.2max(M_c).$) and label them with 1s. The rest of pixels label with 0s, and obtain a binary mask $\hat{M}_c$.
- Determine the biggest connected component in the binary mask $\hat{M}_c$, and take it into its bounding box.
- Return the coordinates of the bounding box.

Object Localization with CAM: Examples

Below you can see some examples of object localization. After classifying the object in the image, we take the class of this object and localize in the image the object of this class.

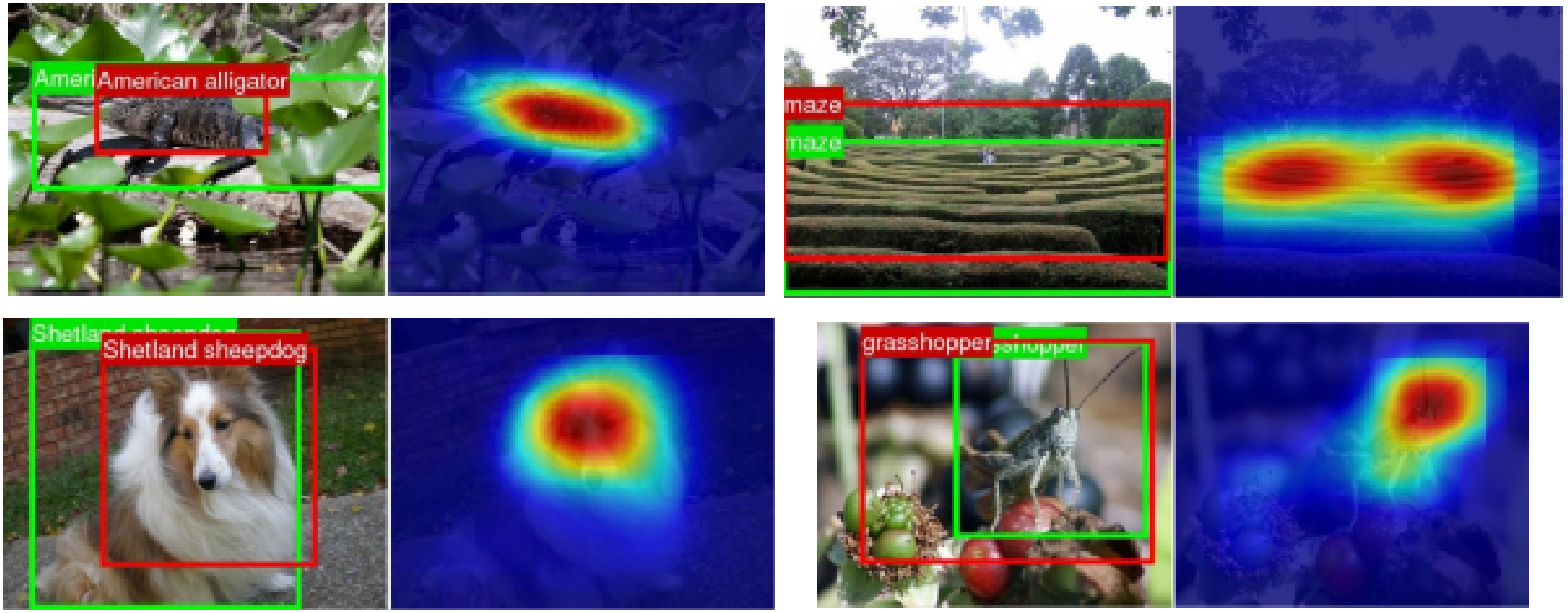


Figure: Red box is the predicted bounding box, green box is the ground truth bounding box.

Localization With Sliding Windows

A naive way of localization is to fix a window size, move the window across the image and predict the class (with its probability score) for every window. The prediction can be done with an image classification network. Then, we can just take the window with the maximal probability score.

Localization With Sliding Windows

A naive way of localization is to fix a window size, move the window across the image and predict the class (with its probability score) for every window. The prediction can be done with an image classification network. Then, we can just take the window with the maximal probability score.
The methods which use the concept of moving windows across the image, are called methods with **sliding windows**.
There are two problems with this approach:

Localization With Sliding Windows

A naive way of localization is to fix a window size, move the window across the image and predict the class (with its probability score) for every window. The prediction can be done with an image classification network. Then, we can just take the window with the maximal probability score.
The methods which use the concept of moving windows across the image, are called methods with **sliding windows**.
There are two problems with this approach:

- Only the objects with the same size as the window can be detected.

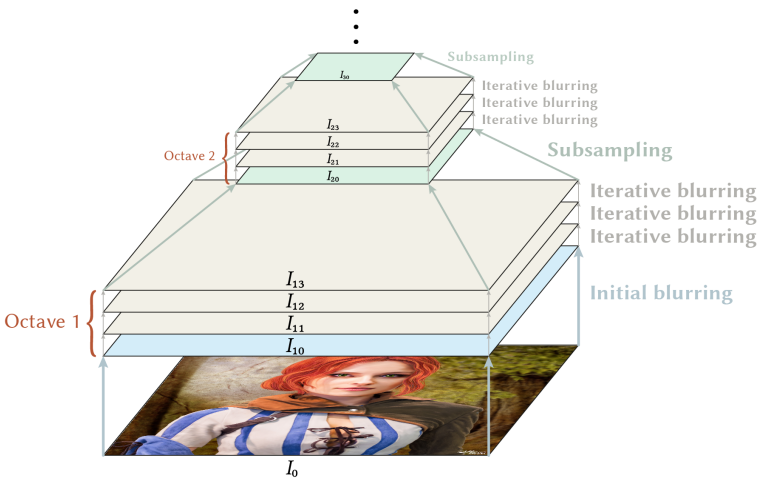
Localization With Sliding Windows

A naive way of localization is to fix a window size, move the window across the image and predict the class (with its probability score) for every window. The prediction can be done with an image classification network. Then, we can just take the window with the maximal probability score.
The methods which use the concept of moving windows across the image, are called methods with **sliding windows**.
There are two problems with this approach:

- Only the objects with the same size as the window can be detected.
- Making prediction for all sliding windows is computationally costly.

Below some techniques are introduced to solve these problems.

The Scale-Space Concept[2]



The *scale-space* is the concept of resizing the image to various sizes and making a "pyramid of images". When for localization problem the window-size is fixed, we can move this fixed-size window across images of all scales in the scale-space. So by doing this, we expect to localize objects regardless their sizes.

This approach of the scale-space increase the computational complexity, so it mostly becomes not usable in production.

[2]Shanmugamani, Rajalingappaa, Abdul Ghani Abdul Rahman, Stephen Maurice Moore, and Nishanth Koganti. 2018. Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using TensorFlow and Keras. Birmingham: Packt Publishing. https://public.ebookcentral.proquest.com/choice/publicfullrecord.aspx?p=5254596.

Fully Convolutional Alternative for Sliding Windows

The method of *sliding-windows* can be effectively replaced by a method with a fully convolutional network[3]. Below we describe the algorithm proposed in the paper.

[3]Sermanet, Pierre, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus and Yann LeCun. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks." CoRR abs/1312.6229 (2013): n. pag.

Fully Convolutional Alternative for Sliding Windows

The method of *sliding-windows* can be effectively replaced by a method with a fully convolutional network[3]. Below we describe the algorithm proposed in the paper. First, a simple classifier network is trained (in the original paper the neural network is a modification of the *AlexNet*) on the fixed size 221×221 (the 221×221 patches are randomly cropped from images in the dataset after resizing them to make the size of the small side 256).

[3] Sermanet, Pierre, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus and Yann LeCun. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks." CoRR abs/1312.6229 (2013): n. pag.

OverFeat

Note

In the paper authors take max-poolings as downsampling layers. Moreover, the max-pooling layers are applied with non-overlapping windows (stride equals to the size of the window). Below we also assume that *the last downsampling layer* (here by downsampling layer we mean a layer with a stride larger than 1) is a *max-pooling* layer with non-overlapping windows of size 3×3. Try to extend this assumption in case of any kind of downsampling layers.

OverFeat

Note

In the paper authors take max-poolings as downsampling layers. Moreover, the max-pooling layers are applied with non-overlapping windows (stride equals to the size of the window). Below we also assume that *the last downsampling layer* (here by downsampling layer we mean a layer with a stride larger than 1) is a *max-pooling* layer with non-overlapping windows of size 3×3. Try to extend this assumption in case of any kind of downsampling layers.

After training the network on the classification dataset, we use *multi-scale classification* approach during the inference of the network. Below we describe the multi-scale classification.

OverFeat: Multi-Scale Classification

Let's denote by l the number of the last downsampling layer of the trained classification network (in our case it is a max-pooling layer with non-overlapping windows of size 3×3).

OverFeat: Multi-Scale Classification

Let's denote by l the number of the last downsampling layer of the trained classification network (in our case it is a max-pooling layer with non-overlapping windows of size 3×3). We scale the image with 6 scales, pass the resulting 6 images to the classifier and consider the outputs of the layer $l-1$, i.e. before the max-pooling in the layer l. The table below shows the resolutions we get in the case of the network from the paper (here $l = 5$):

Scale	Input size	Layer 5 pre-pool
1	245x245	17x17
2	281x317	20x23
3	317x389	23x29
4	389x461	29x35
5	425x497	32x35
6	461x569	35x44

OverFeat: Multi-Scale Classification

Let's denote the resulting features by $f^1, \dots f^6$. For each scale s, let $f^s = (f^s_{i,j})^{H_s-1 \times W_s-1}_{i,j=0,0}$, where $f^s_{i,j} \in \mathbb{R}^D$ for every i, j. For simplicity let's omit the upper index s and write just $f^s = f$. Consider the tensors $\hat{f}^{\Delta_x, \Delta_y}$ for all $\Delta_x, \Delta_y = 0, 1, 2$:

$$\hat{f}^{\Delta_x, \Delta_y}_{i,j} = max\{f_{3i+k+\Delta_x, 3j+k+\Delta_y} : \ k = 0, 1, 2\}.$$

Note

As you can notice, heights and widths of the output tensors from the layer l are $2(mod3)$ (for all scales), so the tensors $\hat{f}^{\Delta_x, \Delta_y}$ are correctly defined. We will proceed with the cases like this, but one can easily generalize the approach to any case.

OverFeat: Multi-Scale Classification

Note

We get 9 tensors $\hat{f}^{\Delta_x, \Delta_y}$, each of which can be obtained by max-pooling layer on the *corresponding* part of the tensor f. The $1d$ projections of these parts are visualized in the image below:

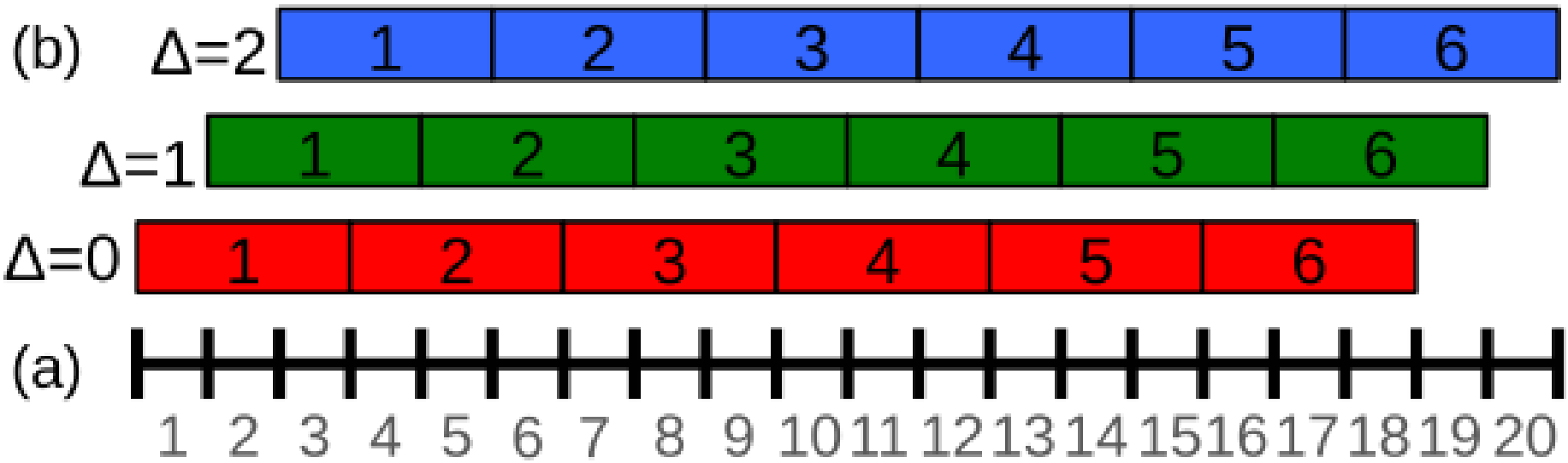


Figure: (a) is one dimension of the layer l, in (b) you can see the parts of the layer l, which are used to get max-pooling features.

OverFeat: Multi-Scale Classification

Then on each tensor $\hat{f}^{\Delta_x, \Delta_y}$ we apply the rest part of trained classifier, in which fully connected layers can be implemented as convolutional layers (in the paper the classifier is composed of 3 fully connected layers, the first one of which is implemented as a convolutional layer with 5×5 kernel, the other two are 1×1 convolutions).

OverFeat: Multi-Scale Classification

Then on each tensor $\hat{f}^{\Delta_x,\Delta_y}$ we apply the rest part of trained classifier, in which fully connected layers can be implemented as convolutional layers (in the paper the classifier is composed of 3 fully connected layers, the first one of which is implemented as a convolutional layer with 5×5 kernel, the other two are 1×1 convolutions). So, we get 9 tensors, let's denote them by $g^{\Delta_x,\Delta_y} \in \mathbb{R}^{h_s \times w_s \times C}$. Then we obtain a tensor $g \in \mathbb{R}^{3h_s \times 3w_s \times C}$ by the following way:

$$g_{i,j} = \hat{f}^{i(mod3),j(mod3)}_{[\frac{i}{3}],[\frac{j}{3}]}.$$

OverFeat: Multi-Scale Classification

Then on each tensor $\hat{f}^{\Delta_x,\Delta_y}$ we apply the rest part of trained classifier, in which fully connected layers can be implemented as convolutional layers (in the paper the classifier is composed of 3 fully connected layers, the first one of which is implemented as a convolutional layer with 5×5 kernel, the other two are 1×1 convolutions). So, we get 9 tensors, let's denote them by $g^{\Delta_x,\Delta_y} \in \mathbb{R}^{h_s \times w_s \times C}$. Then we obtain a tensor $g \in \mathbb{R}^{3h_s \times 3w_s \times C}$ by the following way:

$$g_{i,j} = \hat{f}^{i(mod3),j(mod3)}_{[\frac{i}{3}],[\frac{j}{3}]}.$$

Then we apply the *Global Max-Pooling* on the tensors g (for every scale s we have a tensor $g = g^s$) to get 6 vectors of C-dimensions. To obtain the final result we average these 6 vectors, followed by taking the argmax of this vector.

OverFeat: Localization

Now, when we have described a method for inference with multi-scale classification, before the *Global Max-Pooling* layer we can associate to each pixel in the tensor g its corresponding *receptive field* in the initial image (for every scale $s = 1, \ldots, 6$). These receptive fields we call **fields of view**.

OverFeat: Localization

Now, when we have described a method for inference with multi-scale classification, before the *Global Max-Pooling* layer we can associate to each pixel in the tensor g its corresponding *receptive field* in the initial image (for every scale $s = 1, \ldots, 6$). These receptive fields we call **fields of view**. Further, we train a regression network to predict more accurate bounding boxes for each *field of view at each location and class*. The output of the regression network at each location and each class is 4 numbers, indicating the 4 coordinates of upper-left and bottom-right corners of the bounding box.

OverFeat: Localization, Training the Regressor

After training the classification network, we take its part until the pooling layer l and make a regression network on top of the layer l. The regression network also consists of multi-scaling part as the classification network. In the paper a 5×5 convolution followed by two 1×1 convolutions are applied to the features $\hat{f}^{\Delta_x, \Delta_y}$ (for each shift Δ_x, Δ_y and for each scale $s = 1, \ldots, 6$). Then a 1×1 convolution with number of channels $4C$ is applied on each $\hat{f}^{\Delta_x, \Delta_y}$, which will be treated as bounding box coordinates for each *field of view* and each *class*.

OverFeat: Localization, Training the Regressor

Training the regressor

We train only the regression part of the network. The training loss is l_2 loss between the predicted and ground truth bounding box coordinates (only for the ground truth class of the object). Here by ground truth bounding box we mean the *intersection* of the initial ground truth bounding box with a field of view associated with the particular bounding box. Only the examples which have more than 0.5 Intersection over Union between field of view and initial ground truth bounding box, are taken into account in the loss function.

OverFeat: Localization, Training the Regressor

Training the regressor

We train only the regression part of the network. The training loss is l_2 loss between the predicted and ground truth bounding box coordinates (only for the ground truth class of the object). Here by ground truth bounding box we mean the *intersection* of the initial ground truth bounding box with a field of view associated with the particular bounding box. Only the examples which have more than 0.5 Intersection over Union between field of view and initial ground truth bounding box, are taken into account in the loss function.

Note

As you can notice, we do not take into account the bounding boxes, the fields of view have less than 0.5 *IoU* with the ground truth bounding box. It is quite logical, since in these cases, the receptive field does not include enough from the main object for its localization.

OverFeat: Localization, Prediction of Bounding Boxes

Once the regressor is trained, we can extract features which share the classifier and the regressor, then

[4] In the paper there is another algorithm for obtaining bounding boxes called *greedy merge strategy*

OverFeat: Localization, Prediction of Bounding Boxes

Once the regressor is trained, we can extract features which share the classifier and the regressor, then

- by continuing with the classifier, we obtain top-k classes for each field of view for each scale.

[4] In the paper there is another algorithm for obtaining bounding boxes called *greedy merge strategy*

OverFeat: Localization, Prediction of Bounding Boxes

Once the regressor is trained, we can extract features which share the classifier and the regressor, then

- by continuing with the classifier, we obtain top-k classes for each field of view for each scale.
- by continuing with the regressor, we obtain the bounding boxes for all classes for each field of view and each scale.

[4] In the paper there is another algorithm for obtaining bounding boxes called *greedy merge strategy*

OverFeat: Localization, Prediction of Bounding Boxes

Once the regressor is trained, we can extract features which share the classifier and the regressor, then

- by continuing with the classifier, we obtain top-k classes for each field of view for each scale.
- by continuing with the regressor, we obtain the bounding boxes for all classes for each field of view and each scale.

So we get many bounding boxes, and in order to solve the localization problem we need to obtain several bounding boxes from existing ones (several in general, in the cases if we are able to capture two instances of the same class, or the algorithm can be extended to the object detection algorithm).

[4] In the paper there is another algorithm for obtaining bounding boxes called *greedy merge strategy*

OverFeat: Localization, Prediction of Bounding Boxes

Once the regressor is trained, we can extract features which share the classifier and the regressor, then

- by continuing with the classifier, we obtain top-k classes for each field of view for each scale.
- by continuing with the regressor, we obtain the bounding boxes for all classes for each field of view and each scale.

So we get many bounding boxes, and in order to solve the localization problem we need to obtain several bounding boxes from existing ones (several in general, in the cases if we are able to capture two instances of the same class, or the algorithm can be extended to the object detection algorithm). A method to obtain the desired bounding boxes, which is called *non-maximum suppression*, is described below[4].

[4] In the paper there is another algorithm for obtaining bounding boxes called *greedy merge strategy*

Overview

Non-Maximum Suppression (NMS)

Now we will describe a method to *suppress not relevant bounding boxes* from many bounding box *"proposals"*. Here by saying *"region proposal"* (or just "proposal") we mean the proposed bounding boxes for suppression.

Non-Maximum Suppression (NMS)

Now we will describe a method to *suppress not relevant bounding boxes* from many bounding box *"proposals"*. Here by saying *"region proposal"* (or just "proposal") we mean the proposed bounding boxes for suppression. From the previous section we get many proposals *for each class*. For now let's fix a class and consider only the proposals of this class. We will do the procedure described below for each class.

Non-Maximum Suppression (NMS)

Algorithm of Non-Maximum Suppression

Let the list of proposals B and the list of the scores S of these proposals are given (the score is the probability score for the fixed class). Let N be a hyperparameter from $(0, 1)$ (it will serve as a threshold value).

Non-Maximum Suppression (NMS)

Algorithm of Non-Maximum Suppression

Let the list of proposals B and the list of the scores S of these proposals are given (the score is the probability score for the fixed class). Let N be a hyperparameter from $(0, 1)$ (it will serve as a threshold value).

- Initialize an empty list D.

Non-Maximum Suppression (NMS)

Algorithm of Non-Maximum Suppression

Let the list of proposals B and the list of the scores S of these proposals are given (the score is the probability score for the fixed class). Let N be a hyperparameter from $(0, 1)$ (it will serve as a threshold value).

- Initialize an empty list D.
- Find the largest score in S, and the corresponding proposal: $m = argmax(S),\ \hat{b} = B[m]$.

Non-Maximum Suppression (NMS)

Algorithm of Non-Maximum Suppression

Let the list of proposals B and the list of the scores S of these proposals are given (the score is the probability score for the fixed class). Let N be a hyperparameter from $(0, 1)$ (it will serve as a threshold value).

- Initialize an empty list D.
- Find the largest score in S, and the corresponding proposal: $m = argmax(S),\ \hat{b} = B[m]$.
- Add the proposal $\hat{b}$ to the list D and remove $\hat{b}$ from B.

Non-Maximum Suppression (NMS)

Algorithm of Non-Maximum Suppression

Let the list of proposals B and the list of the scores S of these proposals are given (the score is the probability score for the fixed class). Let N be a hyperparameter from $(0, 1)$ (it will serve as a threshold value).

- Initialize an empty list D.
- Find the largest score in S, and the corresponding proposal: $m = argmax(S),\ \hat{b} = B[m]$.
- Add the proposal $\hat{b}$ to the list D and remove $\hat{b}$ from B.
- For all elements $b \in B$ consider the $IoU(\hat{b}, b)$ and compare it with the predefined threshold N.

Non-Maximum Suppression (NMS)

Algorithm of Non-Maximum Suppression

Let the list of proposals B and the list of the scores S of these proposals are given (the score is the probability score for the fixed class). Let N be a hyperparameter from $(0, 1)$ (it will serve as a threshold value).

- Initialize an empty list D.
- Find the largest score in S, and the corresponding proposal: $m = argmax(S),\ \hat{b} = B[m]$.
- Add the proposal $\hat{b}$ to the list D and remove $\hat{b}$ from B.
- For all elements $b \in B$ consider the $IoU(\hat{b}, b)$ and compare it with the predefined threshold N.
- If $IoU(\hat{b}, b) \geq N$, remove b from B

Non-Maximum Suppression (NMS)

Algorithm of Non-Maximum Suppression

Let the list of proposals B and the list of the scores S of these proposals are given (the score is the probability score for the fixed class). Let N be a hyperparameter from $(0, 1)$ (it will serve as a threshold value).

- Initialize an empty list D.
- Find the largest score in S, and the corresponding proposal: $m = argmax(S),\ \hat{b} = B[m]$.
- Add the proposal $\hat{b}$ to the list D and remove $\hat{b}$ from B.
- For all elements $b \in B$ consider the $IoU(\hat{b}, b)$ and compare it with the predefined threshold N.
- If $IoU(\hat{b}, b) \geq N$, remove b from B
- Repeat starting from the second step until B is not empty.

Non-Maximum Suppression (NMS)

The algorithm of non-maximum suppression highly depends on the *threshold value* N. If we take, for example, $N = 0.5$, in some cases, even for high confidence values (scores), some proposals $b \in B$ will be suppressed as their intersection over union with $\hat{b}$ are above N. On the other hand, some proposals with low scores and no intersection with high-scored proposals will be kept. For solving the last problem, we can introduce also a *score threshold*, and for the first problem, we slightly modify the algorithm of non-maximum suppression[5].
The image below illustrates the problem for non-maximum suppression:

[5] Bodla, Navaneeth & Singh, Bharat & Chellappa, Rama & Davis, Larry. (2017). Improving Object Detection With One Line of Code.

Non-Maximum Suppression

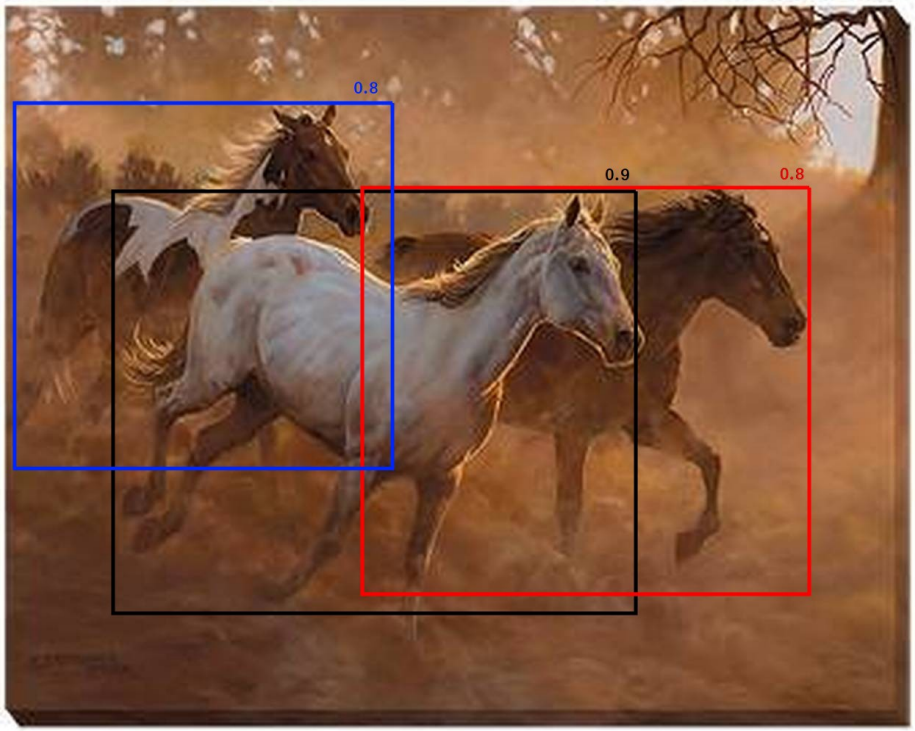


Figure: In this case all three proposals are correct and we want to keep them all. But the non-maximum suppression will return only one proposal - the one with the highest score.

In order to avoid the situations like this, we introduce another algorithm, called **soft non-maximum suppression**.

Soft Non-Maximum Suppression

Let the list of proposals B and the list of the scores S of these proposals are given. Let N and T be hyperparameters from $(0, 1)$ (N - intersection threshold, T - score threshold).

Soft Non-Maximum Suppression

Let the list of proposals B and the list of the scores S of these proposals are given. Let N and T be hyperparameters from $(0, 1)$ (N - intersection threshold, T - score threshold).

- Initialize an empty list D.

Soft Non-Maximum Suppression

Let the list of proposals B and the list of the scores S of these proposals are given. Let N and T be hyperparameters from $(0, 1)$ (N - intersection threshold, T - score threshold).

- Initialize an empty list D.
- Find the largest score in S and the corresponding proposal: $m = argmax(S)$, $\hat{b} = B[m]$.

Soft Non-Maximum Suppression

Let the list of proposals B and the list of the scores S of these proposals are given. Let N and T be hyperparameters from $(0, 1)$ (N - intersection threshold, T - score threshold).

- Initialize an empty list D.
- Find the largest score in S and the corresponding proposal: $m = argmax(S)$, $\hat{b} = B[m]$.
- Add the proposal $\hat{b}$ to the list D and remove $\hat{b}$ from B.

Soft Non-Maximum Suppression

Let the list of proposals B and the list of the scores S of these proposals are given. Let N and T be hyperparameters from $(0, 1)$ (N - intersection threshold, T - score threshold).

- Initialize an empty list D.
- Find the largest score in S and the corresponding proposal: $m = argmax(S)$, $\hat{b} = B[m]$.
- Add the proposal $\hat{b}$ to the list D and remove $\hat{b}$ from B.
- For all elements $b \in B$ consider the $IoU(\hat{b}, b)$ and compare it with the predefined threshold N.

Soft Non-Maximum Suppression

Let the list of proposals B and the list of the scores S of these proposals are given. Let N and T be hyperparameters from $(0, 1)$ (N - intersection threshold, T - score threshold).

- Initialize an empty list D.
- Find the largest score in S and the corresponding proposal: $m = argmax(S)$, $\hat{b} = B[m]$.
- Add the proposal $\hat{b}$ to the list D and remove $\hat{b}$ from B.
- For all elements $b \in B$ consider the $IoU(\hat{b}, b)$ and compare it with the predefined threshold N.
- if $IoU(\hat{b}, b) \geq N$, set the corresponding score $s \leftarrow s(1 - IoU(\hat{b}, b))$.

Soft Non-Maximum Suppression

Let the list of proposals B and the list of the scores S of these proposals are given. Let N and T be hyperparameters from $(0, 1)$ (N - intersection threshold, T - score threshold).

- Initialize an empty list D.
- Find the largest score in S and the corresponding proposal: $m = argmax(S)$, $\hat{b} = B[m]$.
- Add the proposal $\hat{b}$ to the list D and remove $\hat{b}$ from B.
- For all elements $b \in B$ consider the $IoU(\hat{b}, b)$ and compare it with the predefined threshold N.
- if $IoU(\hat{b}, b) \geq N$, set the corresponding score $s \leftarrow s(1 - IoU(\hat{b}, b))$.
- Remove all proposals from B, which have less score than the predefined threshold T.

Soft Non-Maximum Suppression

Let the list of proposals B and the list of the scores S of these proposals are given. Let N and T be hyperparameters from $(0, 1)$ (N - intersection threshold, T - score threshold).

- Initialize an empty list D.
- Find the largest score in S and the corresponding proposal: $m = argmax(S)$, $\hat{b} = B[m]$.
- Add the proposal $\hat{b}$ to the list D and remove $\hat{b}$ from B.
- For all elements $b \in B$ consider the $IoU(\hat{b}, b)$ and compare it with the predefined threshold N.
- if $IoU(\hat{b}, b) \geq N$, set the corresponding score $s \leftarrow s(1 - IoU(\hat{b}, b))$.
- Remove all proposals from B, which have less score than the predefined threshold T.
- Repeat starting from the second step until B is not empty.

Overview

R-CNN

The algorithm we describe below[6] is also based on the idea of passing some rectangle crops from image to a classifier network. These crops we call **region proposals**.

[6] Ren, Shaoqing & He, Kaiming & Girshick, Ross & Sun, Jian. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence. 39. 10.1109/TPAMI.2016.2577031.

R-CNN

The algorithm we describe below[6] is also based on the idea of passing some rectangle crops from image to a classifier network. These crops we call **region proposals**.

The algorithm is composed of the following parts, each of which will be detailed later:

- For each image, proposing the *region proposals*.

[6] Ren, Shaoqing & He, Kaiming & Girshick, Ross & Sun, Jian. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence. 39. 10.1109/TPAMI.2016.2577031.

R-CNN

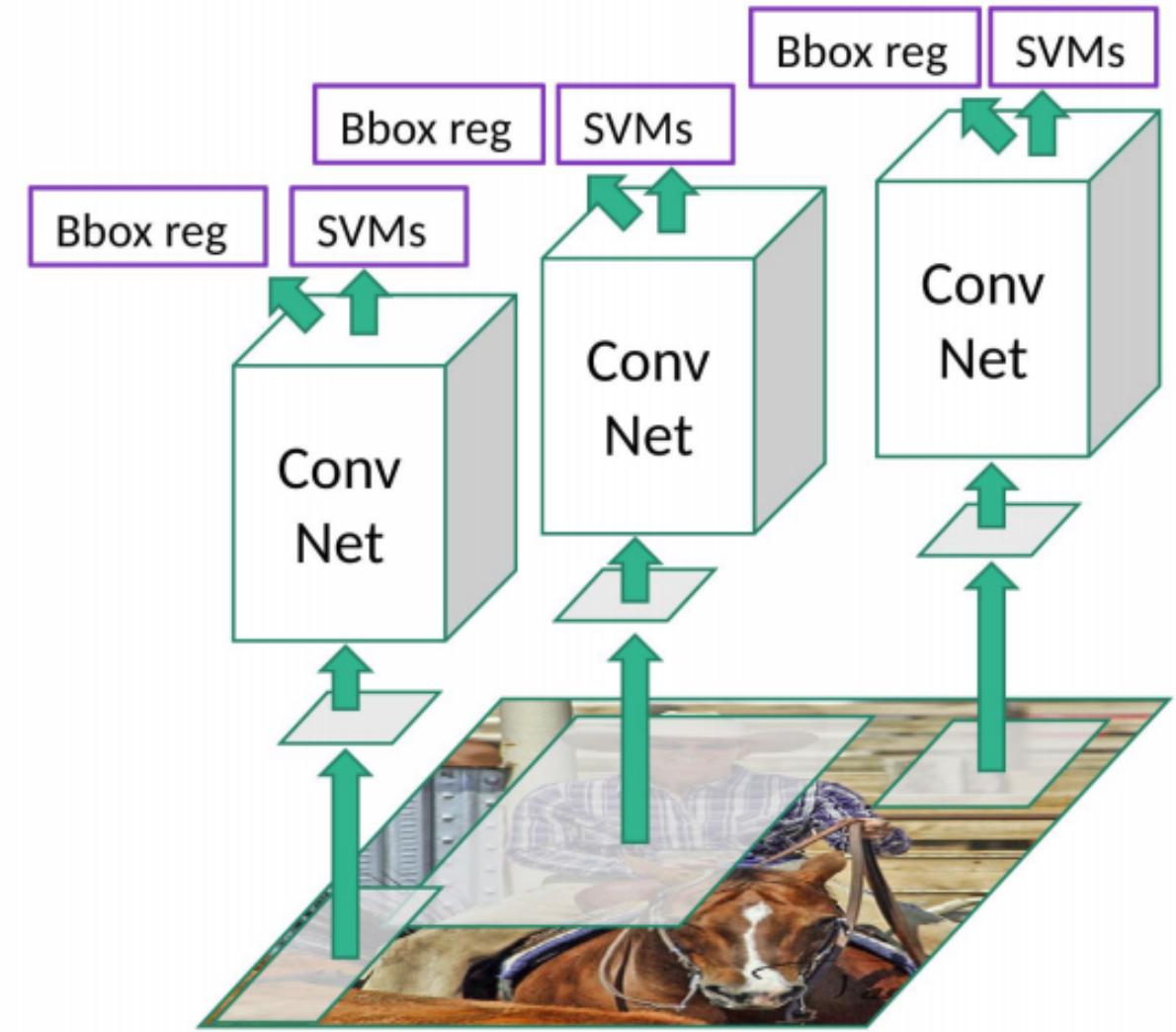
The algorithm we describe below[6] is also based on the idea of passing some rectangle crops from image to a classifier network. These crops we call **region proposals**.

The algorithm is composed of the following parts, each of which will be detailed later:

- For each image, proposing the *region proposals*.
- Resizing the region proposals to the size of the input of a pre-trained network, train the network on these resized proposals to classify among the N classes which are included in an object-detection dataset (the last fully-connected layer will be changed in order to have $N + 1$ units: N for classes, 1 for background).

[6] Ren, Shaoqing & He, Kaiming & Girshick, Ross & Sun, Jian. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence. 39. 10.1109/TPAMI.2016.2577031.

R-CNN


Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Conv Net
Conv Net
Conv Net

R-CNN

R-CNN: Training

- After training the classifier pass the resized region proposals to the classifier and save the extracted features from some layer.

R-CNN: Training

- After training the classifier pass the resized region proposals to the classifier and save the extracted features from some layer.
- On the saved features (one feature for each region proposal) train for each class an *SVM* to classify if the region proposal contains an object of that class or nor.

R-CNN: Training

- After training the classifier pass the resized region proposals to the classifier and save the extracted features from some layer.
- On the saved features (one feature for each region proposal) train for each class an SVM to classify if the region proposal contains an object of that class or nor.
- After training the SVMs, train a regression branch on top of the saved features. For each region proposal the output of the regression branch will be a vector of $4N$ dimensions, where the $i^{th}, \ldots, (i+3)^{th}$ elements show the transformation of the proposal to get a precise bounding box for the i^{th} class.

R-CNN: Inference

- After training the regression branch we end up with the training procedure. The inference for an image is done as follows:

R-CNN: Inference

- After training the regression branch we end up with the training procedure. The inference for an image is done as follows:
  - Find the region proposals.

R-CNN: Inference

- After training the regression branch we end up with the training procedure. The inference for an image is done as follows:
  - Find the region proposals.
  - Pass each region proposal to the feature extractor followed by the *SVM*s, which will return a vector of N dimensions, determine the class with the maximal score (the argmax of this N-dimensional vector).

R-CNN: Inference

- After training the regression branch we end up with the training procedure. The inference for an image is done as follows:
  - Find the region proposals.
  - Pass each region proposal to the feature extractor followed by the *SVM*s, which will return a vector of N dimensions, determine the class with the maximal score (the argmax of this N-dimensional vector).
  - For each class apply *non-maximum suppression* on the region proposals by using the *objectness scores* (the probability that the region proposal contains an object of that particular class) from the *SVM*s. Then consider only remaining proposals.

R-CNN: Inference

- After training the regression branch we end up with the training procedure. The inference for an image is done as follows:
  - Find the region proposals.
  - Pass each region proposal to the feature extractor followed by the *SVM*s, which will return a vector of N dimensions, determine the class with the maximal score (the argmax of this N-dimensional vector).
  - For each class apply *non-maximum suppression* on the region proposals by using the *objectness scores* (the probability that the region proposal contains an object of that particular class) from the *SVM*s. Then consider only remaining proposals.
  - For each region proposal and its corresponding class (determined in the previous step), say i, predict the transformation by using the regressor output, i.e. if the output of the regressor is a vector $X = (X_1, \ldots, X_{4N})$, then the predicted transformation parameters are $X_i, \ldots, X_{i+3}$.

R-CNN: Inference

- After training the regression branch we end up with the training procedure. The inference for an image is done as follows:
  - Find the region proposals.
  - Pass each region proposal to the feature extractor followed by the *SVM*s, which will return a vector of N dimensions, determine the class with the maximal score (the argmax of this N-dimensional vector).
  - For each class apply *non-maximum suppression* on the region proposals by using the *objectness scores* (the probability that the region proposal contains an object of that particular class) from the *SVM*s. Then consider only remaining proposals.
  - For each region proposal and its corresponding class (determined in the previous step), say i, predict the transformation by using the regressor output, i.e. if the output of the regressor is a vector $X = (X_1, \ldots, X_{4N})$, then the predicted transformation parameters are $X_i, \ldots, X_{i+3}$.
  - Apply the transformation $(X_i, \ldots, X_{i+3})$ to the initial region proposal to get its corresponding (refined) bounding box.

R-CNN: Region Proposals

Let's detail the steps described above. We start from a method to get *region proposals* from a single image. The method is called **selective search**[7]

[7] Uijlings, Jasper & Sande, K. & Gevers, T. & Smeulders, Arnold. (2013). Selective Search for Object Recognition. International Journal of Computer Vision. 104. 154-171. 10.1007/s11263-013-0620-5.

R-CNN: Region Proposals

Let's detail the steps described above. We start from a method to get *region proposals* from a single image. The method is called **selective search**[7]

Note

In a region proposal method, we aim to propose relatively less regions but also have among them all regions containing the objects in the image.

[7] Uijlings, Jasper & Sande, K. & Gevers, T. & Smeulders, Arnold. (2013). Selective Search for Object Recognition. International Journal of Computer Vision. 104. 154-171. 10.1007/s11263-013-0620-5.

R-CNN: Region Proposals

Let's detail the steps described above. We start from a method to get *region proposals* from a single image. The method is called **selective search**[7]

Note

In a region proposal method, we aim to propose relatively less regions but also have among them all regions containing the objects in the image.

Below we describe the *selective search* algorithm.

[7] Uijlings, Jasper & Sande, K. & Gevers, T. & Smeulders, Arnold. (2013). Selective Search for Object Recognition. International Journal of Computer Vision. 104. 154-171. 10.1007/s11263-013-0620-5.

Selective Search

Selective search starts by *over-segmenting* the image by an *agglomerative clustering* algorithm[8] on the image we have discussed earlier in this course.

[8] Felzenszwalb, Pedro & Huttenlocher, Daniel. (2004). Efficient Graph-Based Image Segmentation. International Journal of Computer Vision. 59. 167-181. 10.1023/B:VISI.0000022288.19776.77.

[9] https://www.learnopencv.com/selective-search-for-object-detection-cpp-python/

Selective Search

Selective search starts by *over-segmenting* the image by an *agglomerative clustering* algorithm[8] on the image we have discussed earlier in this course. Here is an over-segmented result of the mentioned segmentation algorithm[9]:

[8] Felzenszwalb, Pedro & Huttenlocher, Daniel. (2004). Efficient Graph-Based Image Segmentation. International Journal of Computer Vision. 59. 167-181. 10.1023/B:VISI.0000022288.19776.77.

[9] https://www.learnopencv.com/selective-search-for-object-detection-cpp-python/

Selective Search

The selective search algorithm takes the over-segmented result as initial input and perform the steps described below (each of which will be detailed later):

- Initialize the list of the proposal bounding boxes as an empty list: $B = [\,]$.

Selective Search

The selective search algorithm takes the over-segmented result as initial input and perform the steps described below (each of which will be detailed later):

- Initialize the list of the proposal bounding boxes as an empty list: $B = [\,]$.
- Consider the over-segmented image S.

Selective Search

The selective search algorithm takes the over-segmented result as initial input and perform the steps described below (each of which will be detailed later):

- Initialize the list of the proposal bounding boxes as an empty list: $B = [\,]$.
- Consider the over-segmented image S.
- For each segment in S consider the minimal bounding box, which contains the segment and add this bounding box to the list B.

Selective Search

The selective search algorithm takes the over-segmented result as initial input and perform the steps described below (each of which will be detailed later):

- Initialize the list of the proposal bounding boxes as an empty list: $B = [\,]$.
- Consider the over-segmented image S.
- For each segment in S consider the minimal bounding box, which contains the segment and add this bounding box to the list B.
- Obtain new segments by merging the adjacent segments from S by their similarity. Update S by the new segments.

Selective Search

The selective search algorithm takes the over-segmented result as initial input and perform the steps described below (each of which will be detailed later):

- Initialize the list of the proposal bounding boxes as an empty list: $B = [\,]$.
- Consider the over-segmented image S.
- For each segment in S consider the minimal bounding box, which contains the segment and add this bounding box to the list B.
- Obtain new segments by merging the adjacent segments from S by their similarity. Update S by the new segments.
- Repeat starting from the step 3.

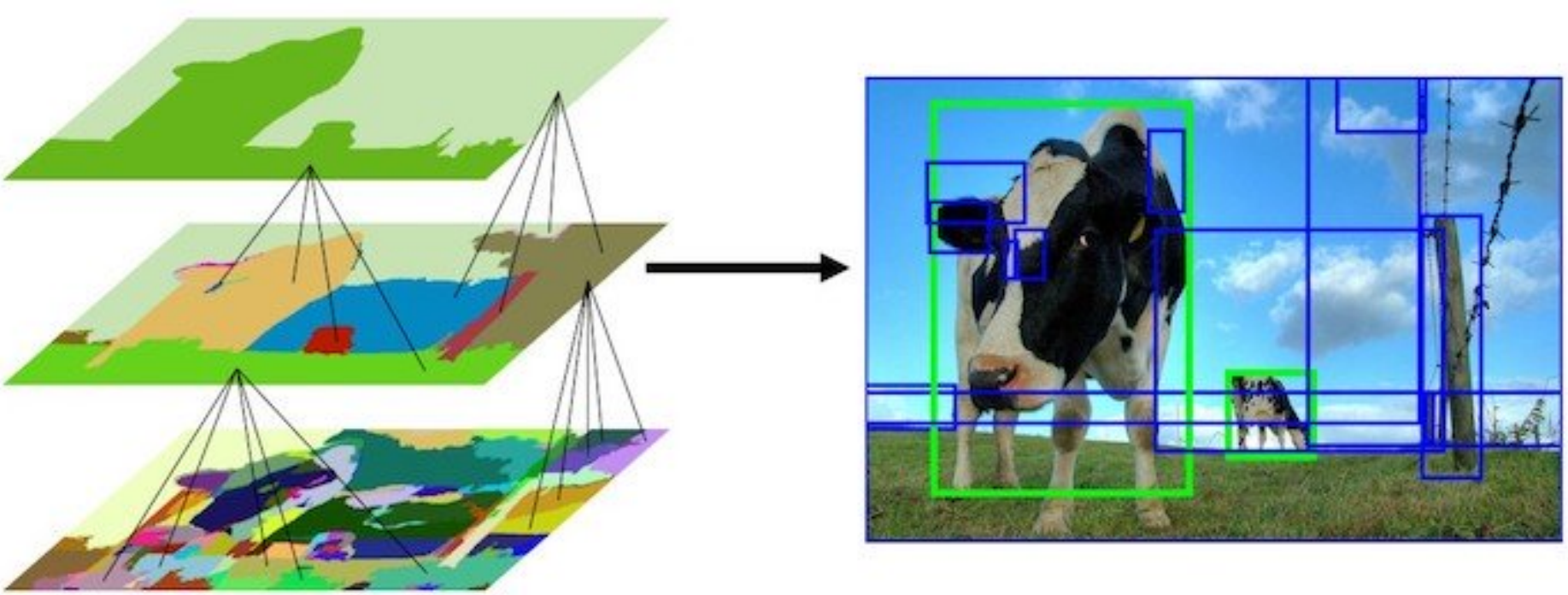
Selective Search

The selective search algorithm takes the over-segmented result as initial input and perform the steps described below (each of which will be detailed later):

- Initialize the list of the proposal bounding boxes as an empty list: $B = [\,]$.
- Consider the over-segmented image S.
- For each segment in S consider the minimal bounding box, which contains the segment and add this bounding box to the list B.
- Obtain new segments by merging the adjacent segments from S by their similarity. Update S by the new segments.
- Repeat starting from the step 3.

In the image below you can see the these steps (the bounding boxes of the over-segmented image are not shown).

Selective Search



Selective Search: Similarity of The Segments

Now we describe a method to merge adjacent segments. For this we introduce the concept of the *similarity of two regions*. Then we merge two regions if they have the highest similarity (among all pairs of regions).

Selective Search: Similarity of The Segments

Now we describe a method to merge adjacent segments. For this we introduce the concept of the *similarity of two regions*. Then we merge two regions if they have the highest similarity (among all pairs of regions). The similarity is composed of 4 components:

- color similarity,

Selective Search: Similarity of The Segments

Now we describe a method to merge adjacent segments. For this we introduce the concept of the *similarity of two regions*. Then we merge two regions if they have the highest similarity (among all pairs of regions). The similarity is composed of 4 components:

- color similarity,
- shape compatibility,

Selective Search: Similarity of The Segments

Now we describe a method to merge adjacent segments. For this we introduce the concept of the *similarity of two regions*. Then we merge two regions if they have the highest similarity (among all pairs of regions). The similarity is composed of 4 components:

- color similarity,
- shape compatibility,
- size similarity,

Selective Search: Similarity of The Segments

Now we describe a method to merge adjacent segments. For this we introduce the concept of the *similarity of two regions*. Then we merge two regions if they have the highest similarity (among all pairs of regions). The similarity is composed of 4 components:

- color similarity,
- shape compatibility,
- size similarity,
- texture similarity.

Selective Search: Similarity of The Segments

Now we describe a method to merge adjacent segments. For this we introduce the concept of the *similarity of two regions*. Then we merge two regions if they have the highest similarity (among all pairs of regions). The similarity is composed of 4 components:

- color similarity,
- shape compatibility,
- size similarity,
- texture similarity.

Color Similarity

For each color channel we construct a 25-bin intensity histogram, resulting with $25 \cdot 3 = 75$-dimensional arrays for each region r. For the region r_i we denote the resulting array by $(c_1^i, \ldots, c_{75}^i)$, and define the **color similarity** by

$$s_{color}(r_i, r_j) = \sum_{k=1}^{75} min(c_k^i, c_k^j).$$

Selective Search: Similarity of The Segments

Size Similarity

We define the **size similarity** as

$$s_{size}(r_i, r_j) = 1 - \frac{area(r_i) + area(r_j)}{area(im)},$$

where $area(r)$ is the area of the region r and im is the image. This encourages the smaller regions to be merged first.

Selective Search: Similarity of The Segments

Size Similarity

We define the **size similarity** as

$$s_{size}(r_i, r_j) = 1 - \frac{area(r_i) + area(r_j)}{area(im)},$$

where $area(r)$ is the area of the region r and im is the image. This encourages the smaller regions to be merged first.

Shape Compatibility

We define the **shape compatibility** as

$$s_{shape}(r_i, r_j) = 1 - \frac{area(BB_{ij}) - area(r_i) - area(r_j)}{area(im)},$$

where BB_{ij} is the minimal bounding box containing the regions r_i and r_j. This encourages to merge first the regions which *fit* into each other well.

Selective Search: Similarity of The Segments

Texture Similarity

For each color channel we consider the *Gaussian derivatives* for all 8 directions, and construct 10-bin histogram for each of image derivative, resulting with $10 \cdot 8 \cdot 3 = 240$ -dimensional array for each region r. For the region r_i we denote the resulting array by $(t_1^i, \ldots, t_{240}^i)$, and define the **texture similarity** by

$$s_{texture}(r_i, r_j) = \sum_{k=1}^{240} min(t_k^i, t_k^j).$$

Selective Search: Similarity of The Segments

Texture Similarity

For each color channel we consider the *Gaussian derivatives* for all 8 directions, and construct 10-bin histogram for each of image derivative, resulting with $10 \cdot 8 \cdot 3 = 240$ -dimensional array for each region r. For the region r_i we denote the resulting array by $(t_1^i, \ldots, t_{240}^i)$, and define the **texture similarity** by

$$s_{texture}(r_i, r_j) = \sum_{k=1}^{240} min(t_k^i, t_k^j).$$

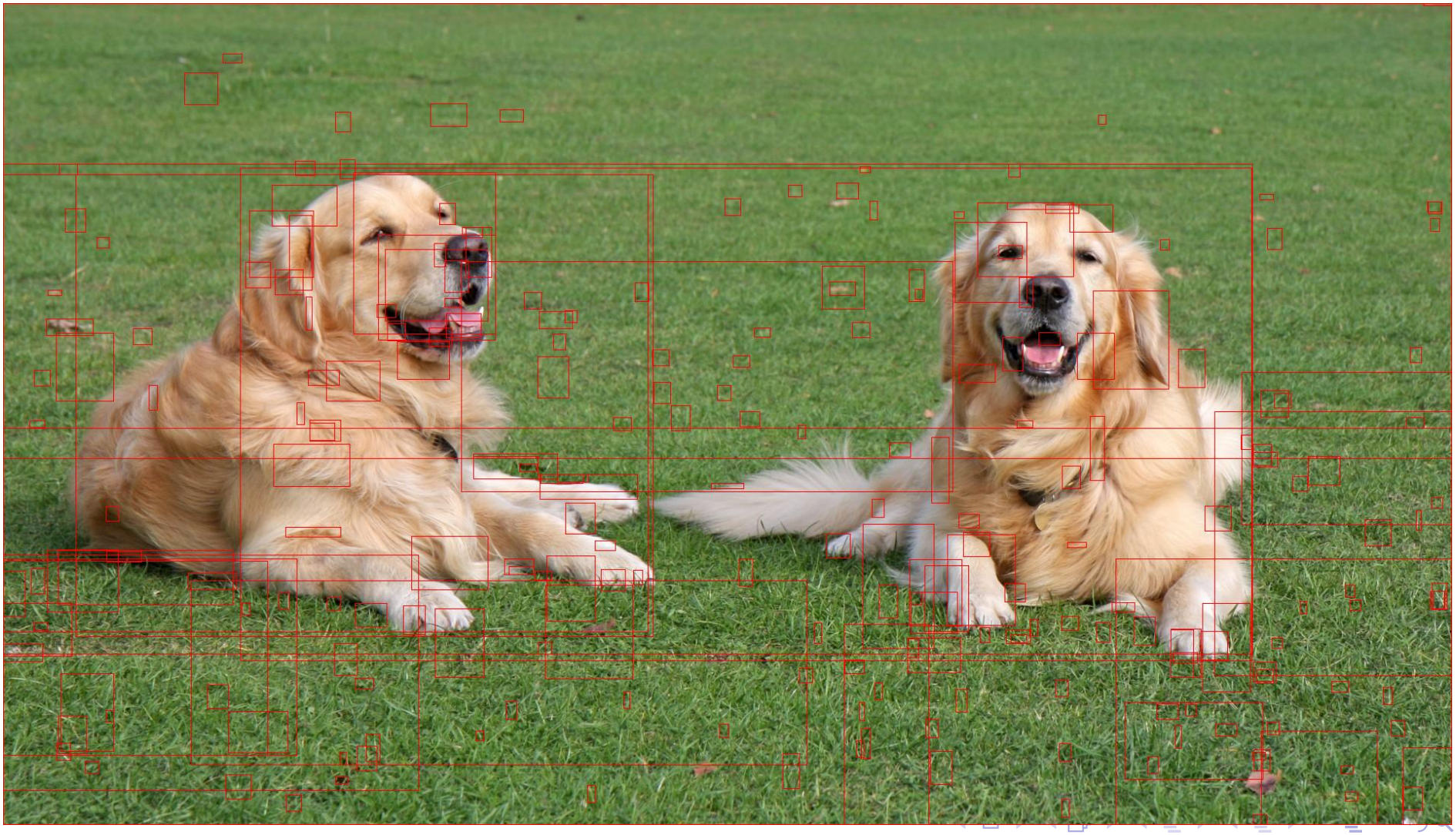
Total Similarity

The total similarity can be calculated as a weighted sum of the above-mentioned similarities:

$$s(r_i, r_j) = a_1 s_{color}(r_i, r_j) + a_2 s_{size}(r_i, r_j) + a_3 s_{shape}(r_i, r_j) + a_4 s_{texture}(r_i, r_j).$$

Selective Search

Here is a result of the region proposal algorithm *selective search* (here are shown some of the resulting bounding boxes):



Training the Classifier on The Proposals

We take a pre-trained network, say *AlexNet*, replace its last ImageNet-specific fully-connected 1000-unit layer with a $N+1$ layer, where N is the number of classes we want to detect and train the obtined network on the region proposals obtained in the previous step.

For training on the proposals, we need to label them at first. We label a region proposal P as follows:

Training the Classifier on The Proposals

We take a pre-trained network, say *AlexNet*, replace its last ImageNet-specific fully-connected 1000-unit layer with a $N+1$ layer, where N is the number of classes we want to detect and train the obtined network on the region proposals obtained in the previous step.

For training on the proposals, we need to label them at first. We label a region proposal P as follows:

- Consider $IoU(B_{gt}, P)$ for all ground truth bounding boxes B_{gt}.

Training the Classifier on The Proposals

We take a pre-trained network, say *AlexNet*, replace its last ImageNet-specific fully-connected 1000-unit layer with a $N+1$ layer, where N is the number of classes we want to detect and train the obtined network on the region proposals obtained in the previous step.

For training on the proposals, we need to label them at first. We label a region proposal P as follows:

- Consider $IoU(B_{gt}, P)$ for all ground truth bounding boxes B_{gt}.
- Determine the bounding box B (among the ground truth bounding boxes B_{gt}), for which

$$IoU(B, P) = \max_{B_{gt}} IoU(B_{gt}, P).$$

Training the Classifier on The Proposals

We take a pre-trained network, say *AlexNet*, replace its last ImageNet-specific fully-connected 1000-unit layer with a $N+1$ layer, where N is the number of classes we want to detect and train the obtined network on the region proposals obtained in the previous step.

For training on the proposals, we need to label them at first. We label a region proposal P as follows:

- Consider $IoU(B_{gt}, P)$ for all ground truth bounding boxes B_{gt}.
- Determine the bounding box B (among the ground truth bounding boxes B_{gt}), for which

$$IoU(B, P) = \max_{B_{gt}} IoU(B_{gt}, P).$$

- If $IoU(B, P) \geq 0.5$, label P by the class of the object contained in the bounding box B, otherwise label P by the backgroung class.

Extracting Features from The Classifier for Each Region Proposal

After training the network, we extract features for each region proposal in every image and save them.

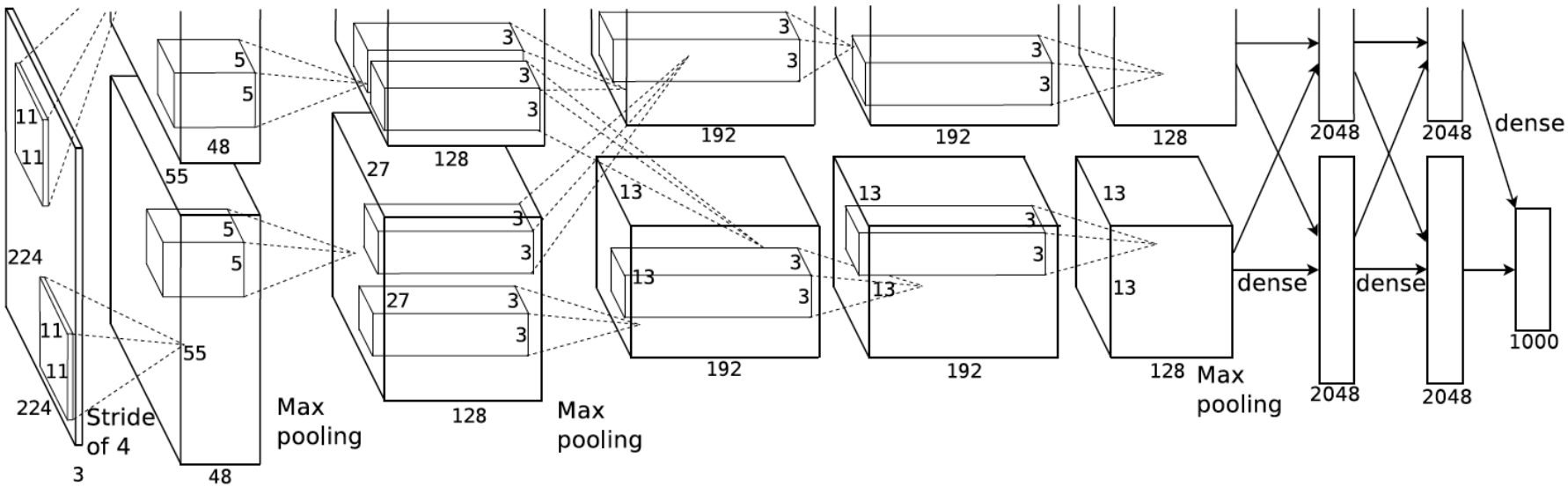


Figure: This is the architecture of *AlexNet*. We replace its last 1000-unit dense layer by a $(N+1)$-unit dense layer.

Training *SVM*s on The Extracted Features

Then for each class we train an *SVM* to predict if the region proposal contains an object of that class or not.

Training the *SVM*s

For training an *SVM* for each class, first we need to choose a data related to that particular class.

Training *SVM*s on The Extracted Features

Then for each class we train an *SVM* to predict if the region proposal contains an object of that class or not.

Training the *SVM*s

For training an *SVM* for each class, first we need to choose a data related to that particular class. We take only the ground truth bounding boxes of that class, resize them and feed to the classifier to extract features. These features are labeled as positive examples, i.e. there is object of the fixed class.

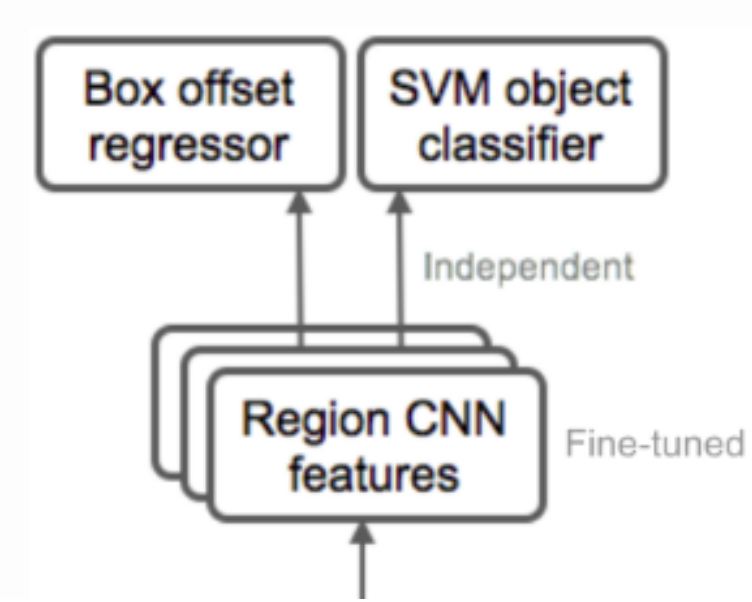
Training *SVM*s on The Extracted Features

Then for each class we train an *SVM* to predict if the region proposal contains an object of that class or not.

Training the *SVM*s

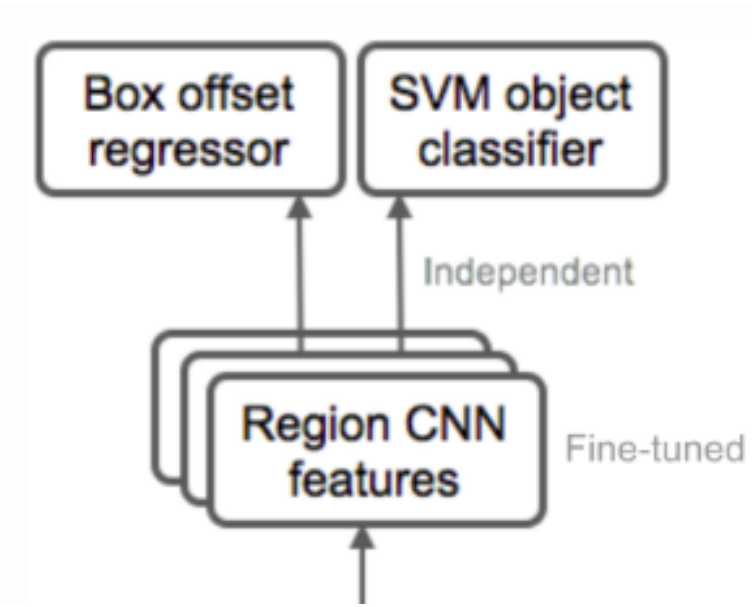
For training an *SVM* for each class, first we need to choose a data related to that particular class. We take only the ground truth bounding boxes of that class, resize them and feed to the classifier to extract features. These features are labeled as positive examples, i.e. there is object of the fixed class. In order to get negative examples, we choose region proposals, which have less than 0.3 *IoU* with any ground truth bounding box, resize them and feed to the classifier to extract features. We label these features as negative examples, i.e. there is no object of the fixed class.

Training The Regression Branch



Besides the *SVM*s, a regression branch is trained on the extracted features from the proposals. The output of the regression branch is 4 numbers (for each class), which indicates *a transformation* (details are described later) to apply on the coordinates of the region proposal to refine the sides of the proposal (for each class).

Training The Regression Branch



Besides the *SVM*s, a regression branch is trained on the extracted features from the proposals. The output of the regression branch is 4 numbers (for each class), which indicates *a transformation* (details are described later) to apply on the coordinates of the region proposal to refine the sides of the proposal (for each class).

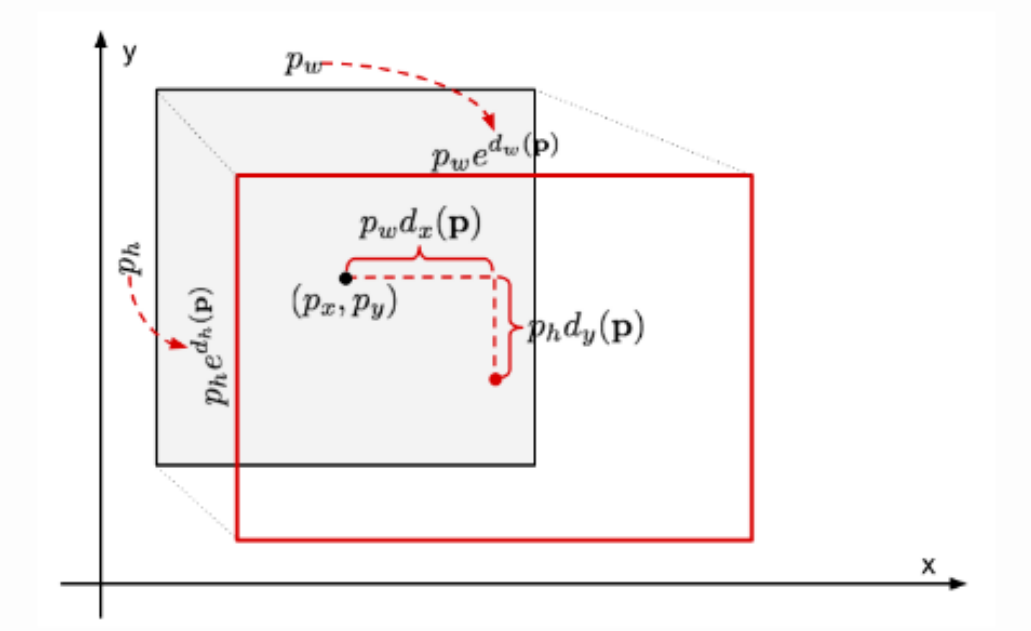
At first we describe the transformation of the region proposal to obtain the corresponding refined bounding box.

The Transformation of The Region Proposal to The Corresponding Bounding Box

Let $P = (P_x, P_y, P_w, P_h)$ be a region proposal, where P_x, P_y are the coordinates of its center, P_h and P_w are the height and the width of the proposal P respectively.

The Transformation of The Region Proposal to The Corresponding Bounding Box

Let $P = (P_x, P_y, P_w, P_h)$ be a region proposal, where P_x, P_y are the coordinates of its center, P_h and P_w are the height and the width of the proposal P respectively. Let we have predicted from the output of the regression layer (for the corresponding class, predicted from the branch of the *SVM*s) the *transformation* $T = (d_x(P), d_y(P), d_w(P), d_h(P))$. Then the resulting refined bounding box is given by its center $(\hat{G}_x, \hat{G}_y)$ and sides $\hat{G}_w, \hat{G}_h$:



$$\hat{G}_x = P_w d_x(P) + P_x,$$

$$\hat{G}_y = P_h d_y(P) + P_y,$$

$$\hat{G}_w = P_w e^{d_w(P)},$$

$$\hat{G}_h = P_h e^{d_h(P)}.$$

Training The Regression Branch

Now let's describe the algorithm of training the regression branch, i.e. obtaining the function $d_*(P)$ (where $*$ is one of x, y, h, w).

Training The Regression Branch

Now let's describe the algorithm of training the regression branch, i.e. obtaining the function $d_*(P)$ (where $*$ is one of x, y, h, w). Let $\phi(P)$ is the feature extracted from the classifier's layer on which both branches are constructed: the branch of *SVM*s and the regression branch. Then we obtain $d_*(P)$ with one *learnable* fully connected layer.

Training The Regression Branch

Now let's describe the algorithm of training the regression branch, i.e. obtaining the function $d_*(P)$ (where $*$ is one of x, y, h, w). Let $\phi(P)$ is the feature extracted from the classifier's layer on which both branches are constructed: the branch of *SVM*s and the regression branch. Then we obtain $d_*(P)$ with one *learnable* fully connected layer. We train the fully connected layer by the L_2 loss and L_2-regularization.

Note

For training this layer, we only need to take the *positive* region proposals, i.e. those, which contains an object. So at first we need to specify the positive ones.

Training The Regression Branch

Now let's describe the algorithm of training the regression branch, i.e. obtaining the function $d_*(P)$ (where $*$ is one of x, y, h, w). Let $\phi(P)$ is the feature extracted from the classifier's layer on which both branches are constructed: the branch of *SVM*s and the regression branch. Then we obtain $d_*(P)$ with one *learnable* fully connected layer. We train the fully connected layer by the L_2 loss and L_2-regularization.

Note

For training this layer, we only need to take the *positive* region proposals, i.e. those, which contains an object. So at first we need to specify the positive ones. So the proposal P is referred as *positive*, if it has the maximum *IoU* with a ground truth bounding box G, and this *IoU* is at least 0.6. We assign the bounding box G and its ground truth class c to the proposal P.

Training The Regression Branch

After assigning G and c to P, we determine the transformation T, which will be applied on P in order to get G:

Training The Regression Branch

After assigning G and c to P, we determine the transformation T, which will be applied on P in order to get G:

$$t_x = (G_x - P_x)/P_w, \quad t_w = log(G_w/P_w),$$

$$t_y = (G_y - P_y)/P_h, \quad t_h = log(G_h/P_h).$$

Training The Regression Branch

After assigning G and c to P, we determine the transformation T, which will be applied on P in order to get G:

$$t_x = (G_x - P_x)/P_w, \quad t_w = log(G_w/P_w),$$

$$t_y = (G_y - P_y)/P_h, \quad t_h = log(G_h/P_h).$$

Then we add the following term to the total loss:

$$\sum_{*=x,y,h,w} (d_*^c(P) - t_*)^2,$$

where $(d_x^c(P), d_y^c(P), d_w^c(P), d_h^c(P))$ is the transformation predicted for the proposal P *for the class* c (the predictions for other classes are not participating in the loss).

Overview

Fast R-CNN

There are several drawbacks of the $R - CNN$ algorithm:

[10] Girshick, Ross B.. "Fast R-CNN." 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 1440-1448.

Fast R-CNN

There are several drawbacks of the $R - CNN$ algorithm:

- Training is multi-stage.

[10] Girshick, Ross B.. "Fast R-CNN." 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 1440-1448.

Fast R-CNN

There are several drawbacks of the $R - CNN$ algorithm:

- Training is multi-stage.
- Training is expensive in space and time.

[10] Girshick, Ross B.. "Fast R-CNN." 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 1440-1448.

Fast R-CNN

There are several drawbacks of the $R - CNN$ algorithm:

- Training is multi-stage.
- Training is expensive in space and time.
- Object detection is slow in the inference.

We describe an algorithm[10] called **Fast R-CNN** which solves the issues above in some sense. So, in case of *Fast R-CNN*:

[10] Girshick, Ross B.. "Fast R-CNN." 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 1440-1448.

Fast R-CNN

There are several drawbacks of the $R - CNN$ algorithm:

- Training is multi-stage.
- Training is expensive in space and time.
- Object detection is slow in the inference.

We describe an algorithm[10] called **Fast R-CNN** which solves the issues above in some sense. So, in case of *Fast R-CNN*:

- we have an *end-to-end* training pipeline,

[10] Girshick, Ross B.. "Fast R-CNN." 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 1440-1448.

Fast R-CNN

There are several drawbacks of the $R - CNN$ algorithm:

- Training is multi-stage.
- Training is expensive in space and time.
- Object detection is slow in the inference.

We describe an algorithm[10] called **Fast R-CNN** which solves the issues above in some sense. So, in case of *Fast R-CNN*:

- we have an *end-to-end* training pipeline,
- training is faster and requires less memory,

[10] Girshick, Ross B.. "Fast R-CNN." 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 1440-1448.

Fast R-CNN

There are several drawbacks of the $R - CNN$ algorithm:

- Training is multi-stage.
- Training is expensive in space and time.
- Object detection is slow in the inference.

We describe an algorithm[10] called **Fast R-CNN** which solves the issues above in some sense. So, in case of *Fast R-CNN*:

- we have an *end-to-end* training pipeline,
- training is faster and requires less memory,
- inference is faster.

R-CNN is slow because it passes every region proposal to the convolutional network without sharing computations.

[10] Girshick, Ross B.. "Fast R-CNN." 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 1440-1448.

Fast R-CNN

The *Fast R-CNN* algorithm is composed of the following steps.

Fast R-CNN

The *Fast R-CNN* algorithm is composed of the following steps.

- For each image obtain the region proposals with the selective search algorithm.

Fast R-CNN

The *Fast R-CNN* algorithm is composed of the following steps.

- For each image obtain the region proposals with the selective search algorithm.
- An image is fed into a convolutional network to extract features from a deep layer.

Fast R-CNN

The *Fast R-CNN* algorithm is composed of the following steps.

- For each image obtain the region proposals with the selective search algorithm.
- An image is fed into a convolutional network to extract features from a deep layer.
- For each region proposal get its *"spatially corresponding"* region from the features obtained in the previous step.

Fast R-CNN

The *Fast R-CNN* algorithm is composed of the following steps.

- For each image obtain the region proposals with the selective search algorithm.
- An image is fed into a convolutional network to extract features from a deep layer.
- For each region proposal get its *"spatially corresponding"* region from the features obtained in the previous step.
- A special *RoI (Region of Interest) pooling* layer is introduced to extract a fixed-sized vector for each region proposal.

Fast R-CNN

The *Fast R-CNN* algorithm is composed of the following steps.

- For each image obtain the region proposals with the selective search algorithm.
- An image is fed into a convolutional network to extract features from a deep layer.
- For each region proposal get its *"spatially corresponding"* region from the features obtained in the previous step.
- A special *RoI (Region of Interest) pooling* layer is introduced to extract a fixed-sized vector for each region proposal.
- After some fully-connected layers the obtained feature-vectors are fed into two branches: *classification* and *regression* branches.

Fast R-CNN

The *Fast R-CNN* algorithm is composed of the following steps.

- For each image obtain the region proposals with the selective search algorithm.
- An image is fed into a convolutional network to extract features from a deep layer.
- For each region proposal get its *"spatially corresponding"* region from the features obtained in the previous step.
- A special *RoI (Region of Interest) pooling* layer is introduced to extract a fixed-sized vector for each region proposal.
- After some fully-connected layers the obtained feature-vectors are fed into two branches: *classification* and *regression* branches.
- In training time update the weights of the network w.r.t. the losses on the outputs of the two mentioned branches.

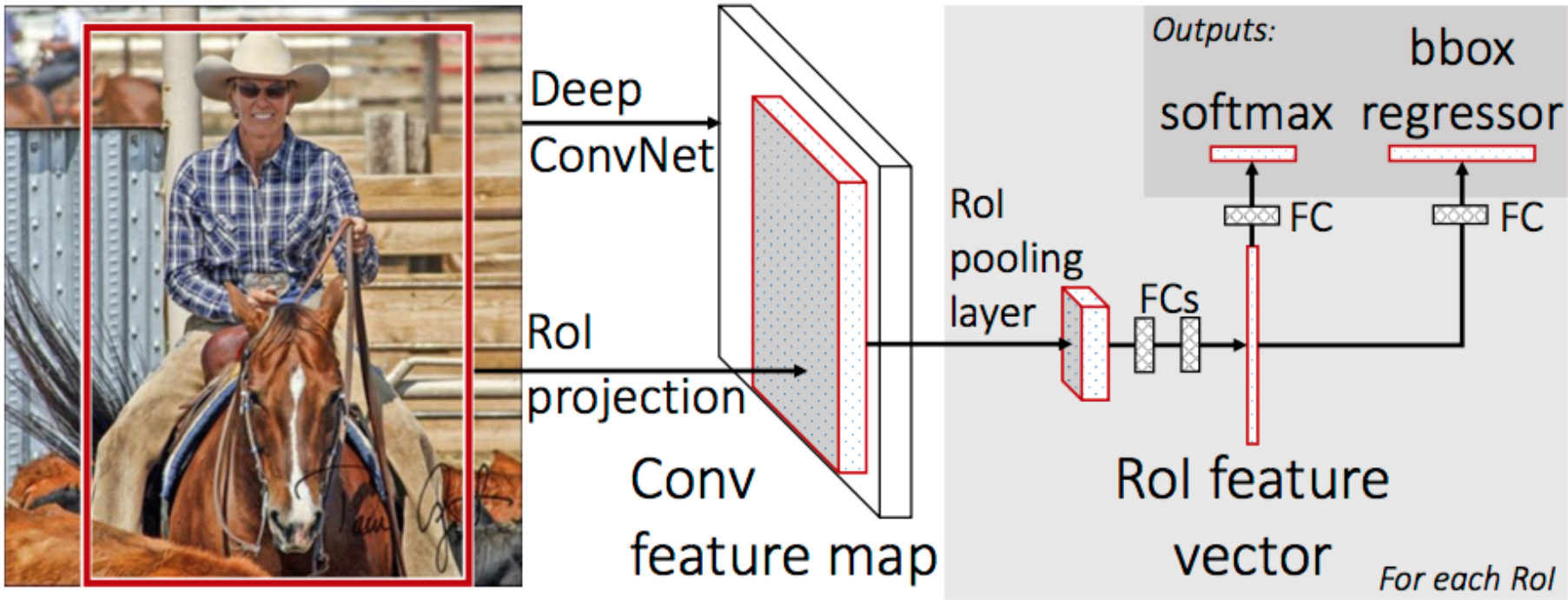
Fast R-CNN

The *Fast R-CNN* algorithm is composed of the following steps.

- For each image obtain the region proposals with the selective search algorithm.
- An image is fed into a convolutional network to extract features from a deep layer.
- For each region proposal get its *"spatially corresponding"* region from the features obtained in the previous step.
- A special *RoI (Region of Interest) pooling* layer is introduced to extract a fixed-sized vector for each region proposal.
- After some fully-connected layers the obtained feature-vectors are fed into two branches: *classification* and *regression* branches.
- In training time update the weights of the network w.r.t. the losses on the outputs of the two mentioned branches.
- In inference time per-class non-maximum suppression is applied.

Fast R-CNN

These steps are shown in the image below:



Fast R-CNN

As we have mentioned, the Fast R-CNN also takes the proposals suggested by the selective search algorithm, but instead of resizing them and feeding to the classification network, it finds the **spatially corresponding** regions on a deep layer of the network.

Fast R-CNN

As we have mentioned, the Fast R-CNN also takes the proposals suggested by the selective search algorithm, but instead of resizing them and feeding to the classification network, it finds the **spatially corresponding** regions on a deep layer of the network.

Question

How this correspondence can be defined?

Fast R-CNN

As we have mentioned, the Fast R-CNN also takes the proposals suggested by the selective search algorithm, but instead of resizing them and feeding to the classification network, it finds the **spatially corresponding** regions on a deep layer of the network.

Question

How this correspondence can be defined?

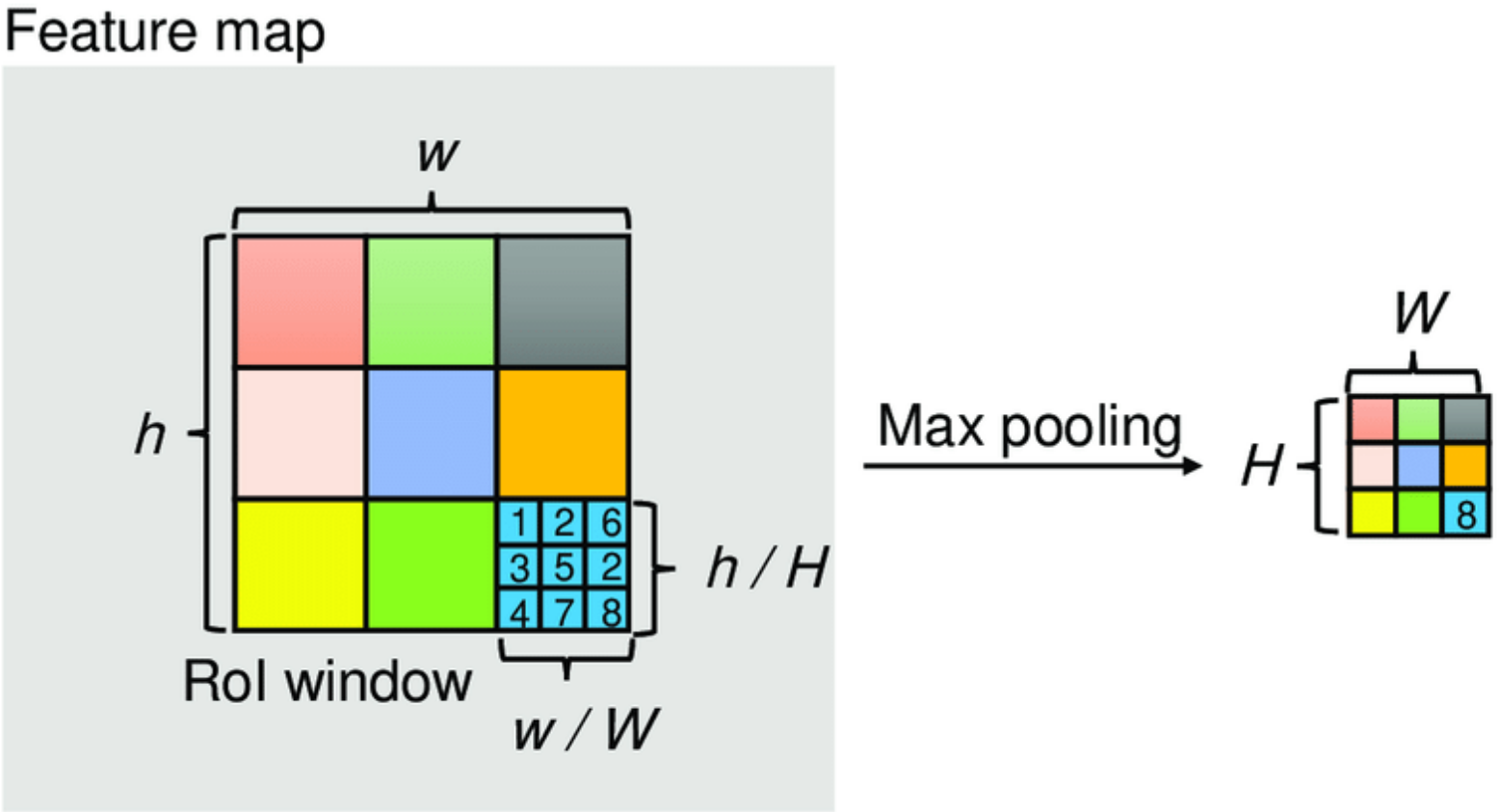
The corresponding regions to the proposals we call *regions of interest*. After getting the regions of interest on the features, we apply the *RoI pooling* layer, which is described below.

RoI (Region of Interest) Pooling Layer

RoI pooling originally is defined as *RoI max pooling* layer. It takes the $h \times w \times C$-sized RoI, and divide it into $H \times W \times C$ grid of sub-windows of approximate size $h/H \times w/W \times C$. In each sub-window a max pooling is applied, resulting with $H \times W \times C$ feature map. This process is visualized below.

RoI (Region of Interest) Pooling Layer

RoI pooling originally is defined as *RoI max pooling* layer. It takes the $h \times w \times C$-sized RoI, and divide it into $H \times W \times C$ grid of sub-windows of approximate size $h/H \times w/W \times C$. In each sub-window a max pooling is applied, resulting with $H \times W \times C$ feature map. This process is visualized below.



Fast R-CNN

After the RoI pooling layers some fully connected layers are applied followed by two sibling branches:

Fast R-CNN

After the RoI pooling layers some fully connected layers are applied followed by two sibling branches:

- classification branch with $K + 1$ classes,

Fast R-CNN

After the RoI pooling layers some fully connected layers are applied followed by two sibling branches:

- classification branch with $K + 1$ classes,
- bounding box regression branch, which gives 4 transformation parameters for all K classes as in the case of *R-CNN* algorithm.

Faster R-CNN

The Fast R-CNN algorithm has a drawback concerning the region proposals, which are obtained by the *selective search* algorithm. In order to make this step faster, the *Faster R-CNN*[11] algorithm proposes to get the *region proposals* by a fully convolutional network.
The network which proposes regions is called *Region Proposal Network (RPN)*, which is described below.

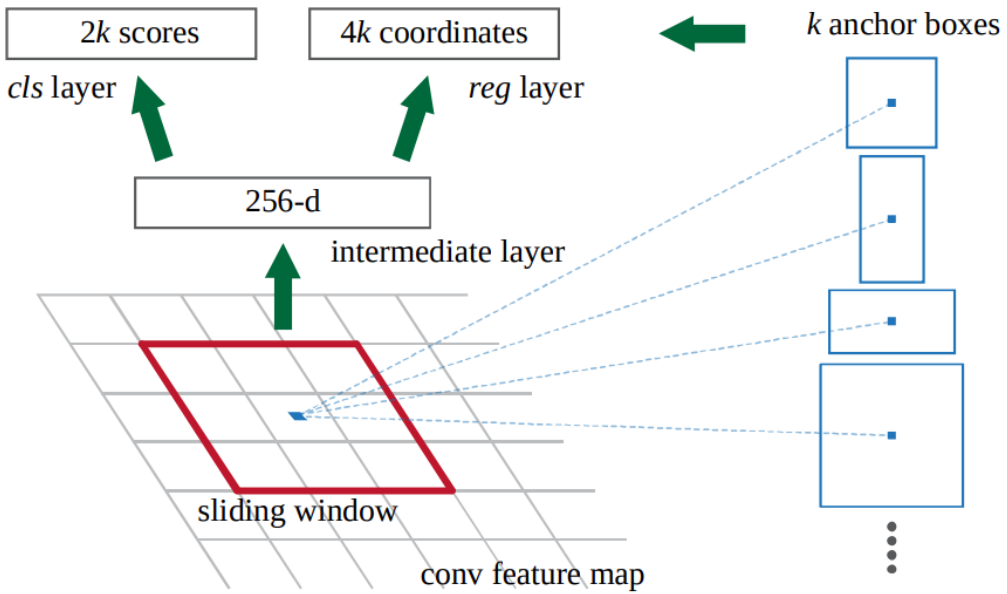
[11] Ren, Shaoqing & He, Kaiming & Girshick, Ross & Sun, Jian. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence. 39. 10.1109/TPAMI.2016.2577031.

Region Proposal Network (RPN)

The *Region Proposal Network* takes the image as input and produces region proposals with their *objectness scores* (objectness score shows the probability of containing an object in the proposal).

Region Proposal Network (RPN)

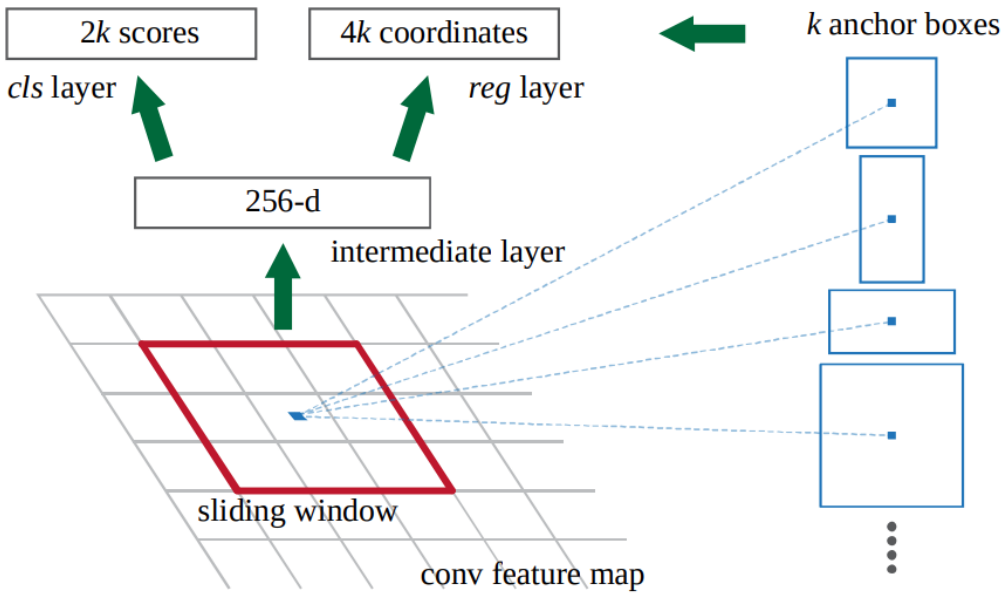
The *Region Proposal Network* takes the image as input and produces region proposals with their *objectness scores* (objectness score shows the probability of containing an object in the proposal). The architecture of the *RPN* is the following:



Region Proposal Network (RPN)

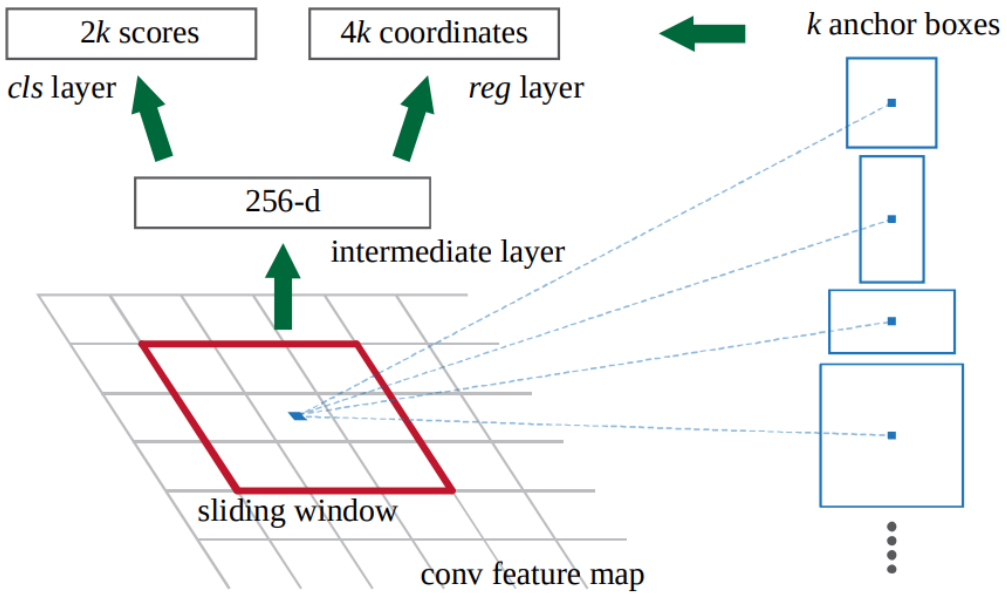
The *Region Proposal Network* takes the image as input and produces region proposals with their *objectness scores* (objectness score shows the probability of containing an object in the proposal). The architecture of the *RPN* is the following:

- A fully convolutional part on the image.



Region Proposal Network (RPN)

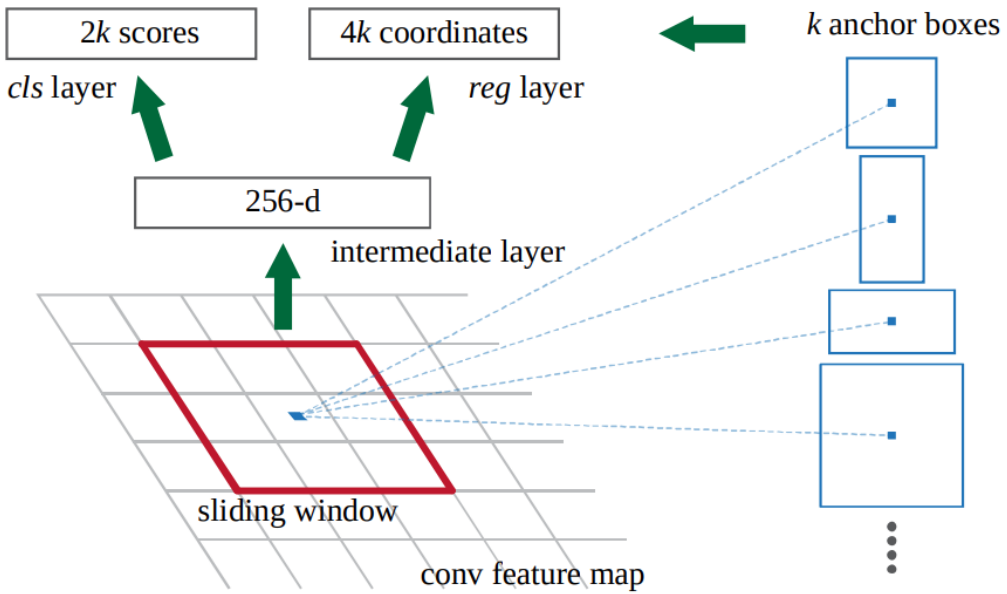
The *Region Proposal Network* takes the image as input and produces region proposals with their *objectness scores* (objectness score shows the probability of containing an object in the proposal). The architecture of the *RPN* is the following:



- A fully convolutional part on the image.
- On the obtained feature map a convolutional layer with the kernel size $n \times n$ (for some n).

Region Proposal Network (RPN)

The *Region Proposal Network* takes the image as input and produces region proposals with their *objectness scores* (objectness score shows the probability of containing an object in the proposal). The architecture of the *RPN* is the following:



- A fully convolutional part on the image.
- On the obtained feature map a convolutional layer with the kernel size $n \times n$ (for some n).
- Two sibling branches, implemented as 1×1 convolutions, one with $2k$ filters, the other with $4k$ filters (for some k).

The Anchors

After obtaining the feature map in the first step of the architecture of RPN, at each pixel p we consider a small window of size $n \times n$ ($n = 3$) centered in the pixel p. We also consider k windows centered in p and of various sizes. The predictions of the RPN will show

The Anchors

After obtaining the feature map in the first step of the architecture of RPN, at each pixel p we consider a small window of size $n \times n$ ($n = 3$) centered in the pixel p. We also consider k windows centered in p and of various sizes. The predictions of the RPN will show

- the objectness score for each of k windows in the *classification layer* (the layer with $2k$ feature maps, 2 stands for two-class softmax layer),

The Anchors

After obtaining the feature map in the first step of the architecture of RPN, at each pixel p we consider a small window of size $n \times n$ ($n = 3$) centered in the pixel p. We also consider k windows centered in p and of various sizes. The predictions of the RPN will show

- the objectness score for each of k windows in the *classification layer* (the layer with $2k$ feature maps, 2 stands for two-class softmax layer),
- the bounding box transformation for each of k windows, which will be applied to the *spatially corresponding* regions (of the k windows) in the image.

The Anchors

After obtaining the feature map in the first step of the architecture of RPN, at each pixel p we consider a small window of size $n \times n$ ($n = 3$) centered in the pixel p. We also consider k windows centered in p and of various sizes. The predictions of the RPN will show

- the objectness score for each of k windows in the *classification layer* (the layer with $2k$ feature maps, 2 stands for two-class softmax layer),
- the bounding box transformation for each of k windows, which will be applied to the *spatially corresponding* regions (of the k windows) in the image.

Anchors

We call the k windows (or their spatially corresponding bounding boxes in the input image) mentioned above, **anchors**.

The Anchors

After obtaining the feature map in the first step of the architecture of RPN, at each pixel p we consider a small window of size $n \times n$ ($n = 3$) centered in the pixel p. We also consider k windows centered in p and of various sizes. The predictions of the RPN will show

- the objectness score for each of k windows in the *classification layer* (the layer with $2k$ feature maps, 2 stands for two-class softmax layer),
- the bounding box transformation for each of k windows, which will be applied to the *spatially corresponding* regions (of the k windows) in the image.

Anchors

We call the k windows (or their spatially corresponding bounding boxes in the input image) mentioned above, **anchors**.

The training of the RPN is done by the similar way as the training of the Fast R-CNN network.

Results of RPN

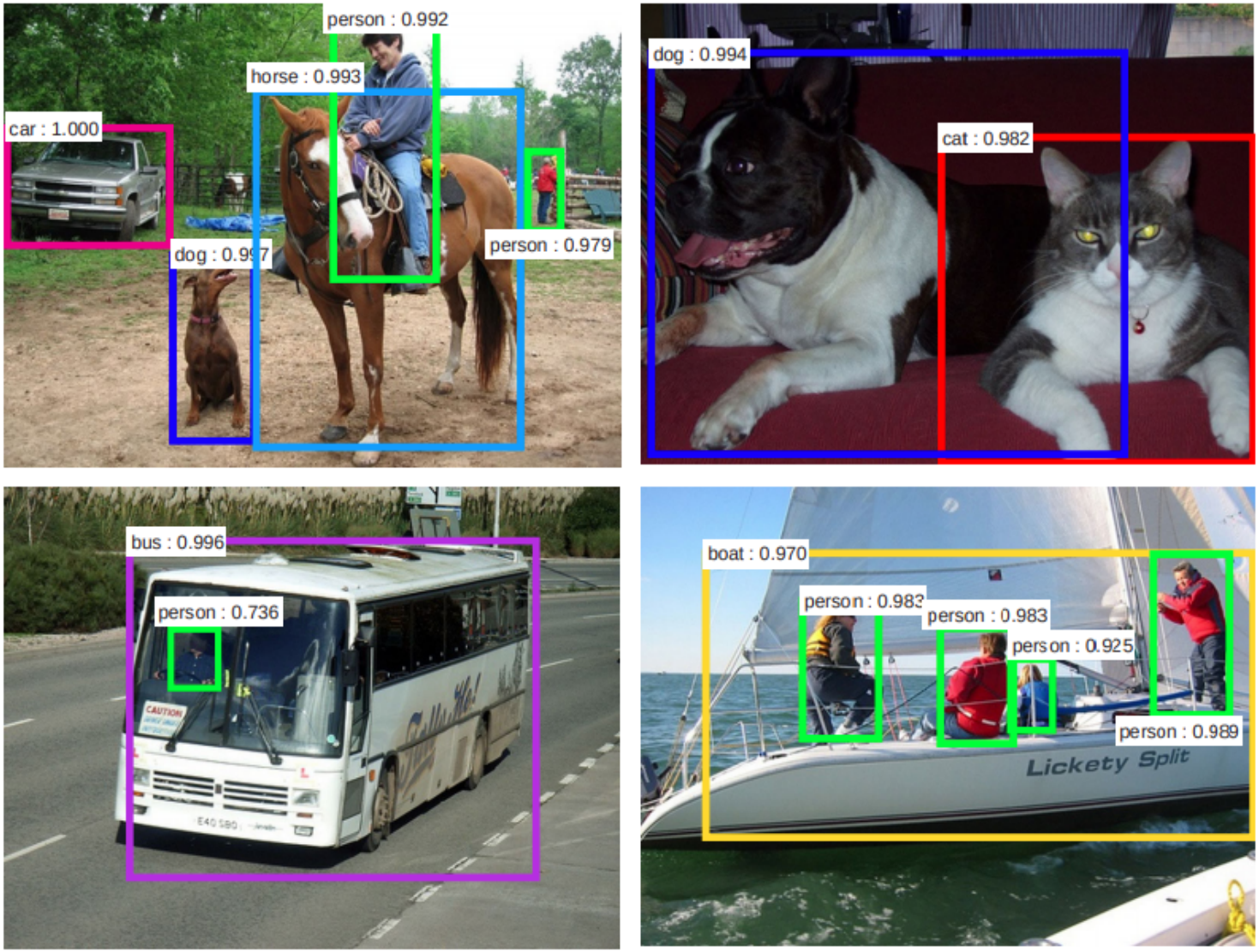


Figure: RPN detection examples.

Sharing Features for RPN and Fast R-CNN

The whole architecture of the *Faster R-CNN* network is shown below:

Sharing Features for RPN and Fast R-CNN

The whole architecture of the *Faster R-CNN* network is shown below:

- After getting the region proposals from the RPN network, we apply non-maximum suppression in order to reduce the number of proposals.

Sharing Features for RPN and Fast R-CNN

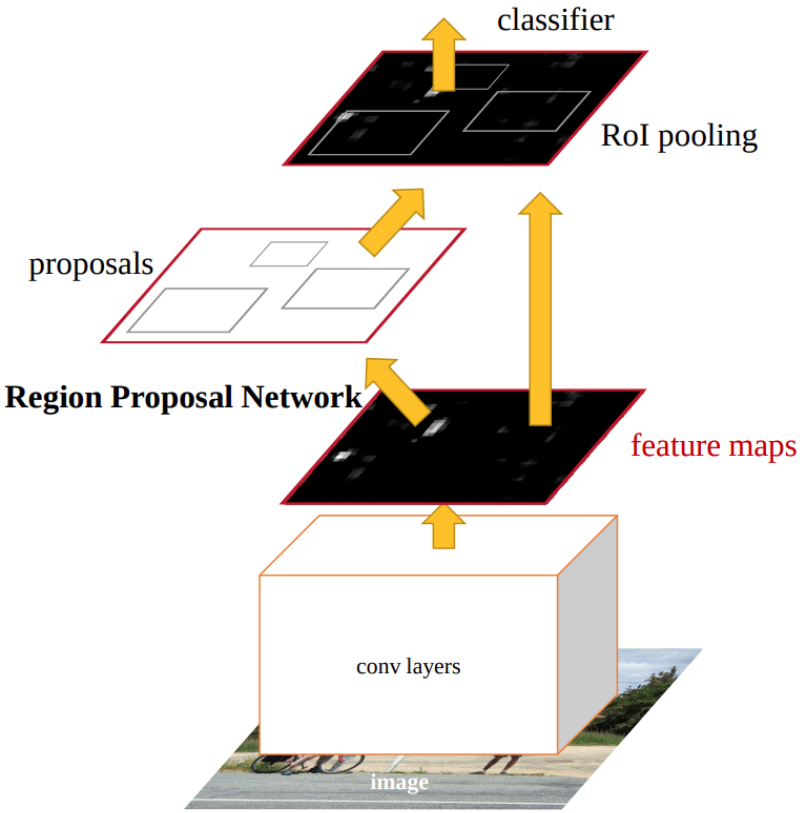
The whole architecture of the *Faster R-CNN* network is shown below:

- After getting the region proposals from the RPN network, we apply non-maximum suppression in order to reduce the number of proposals.
- The proposals are then fed to the RoI pooling layer followed by the rest architecture of the Fast R-CNN.

Sharing Features for RPN and Fast R-CNN

The whole architecture of the *Faster R-CNN* network is shown below:

- After getting the region proposals from the RPN network, we apply non-maximum suppression in order to reduce the number of proposals.
- The proposals are then fed to the RoI pooling layer followed by the rest architecture of the Fast R-CNN.

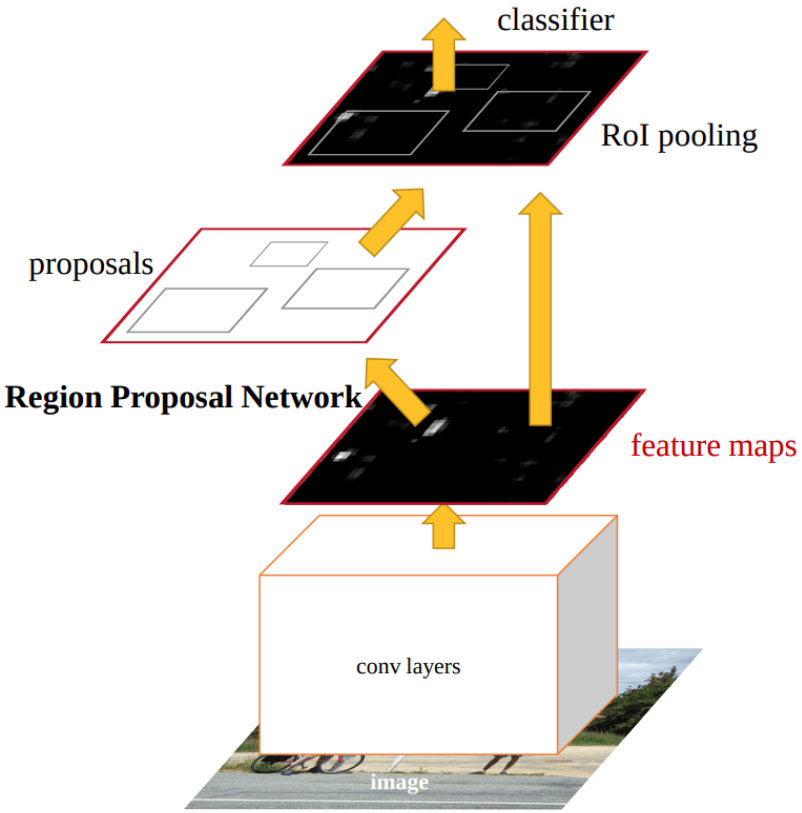


- The training of RPN and Fast R-CNN is done by the following way:

Sharing Features for RPN and Fast R-CNN

The whole architecture of the *Faster R-CNN* network is shown below:

- After getting the region proposals from the RPN network, we apply non-maximum suppression in order to reduce the number of proposals.
- The proposals are then fed to the RoI pooling layer followed by the rest architecture of the Fast R-CNN.

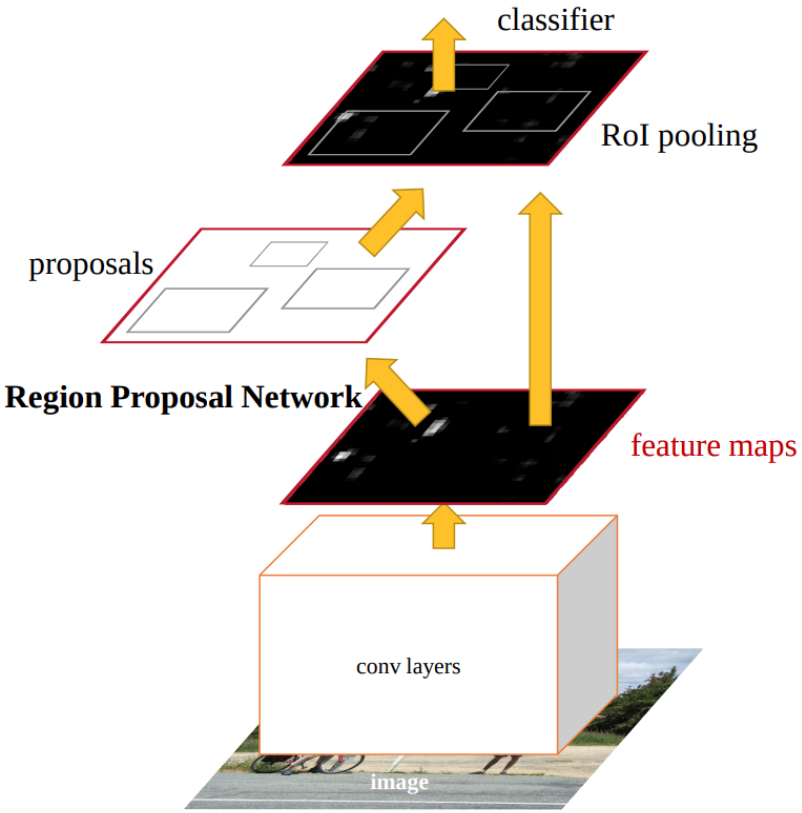


- The training of RPN and Fast R-CNN is done by the following way:
  - at first RPN is trained,

Sharing Features for RPN and Fast R-CNN

The whole architecture of the *Faster R-CNN* network is shown below:

- After getting the region proposals from the RPN network, we apply non-maximum suppression in order to reduce the number of proposals.
- The proposals are then fed to the RoI pooling layer followed by the rest architecture of the Fast R-CNN.

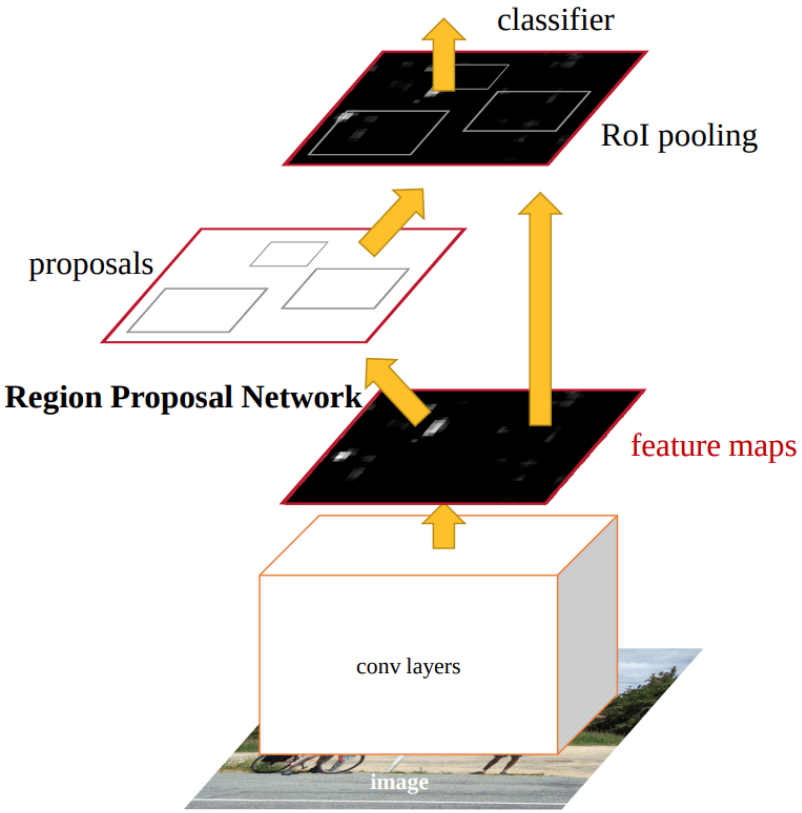


- The training of RPN and Fast R-CNN is done by the following way:
  - at first RPN is trained,
  - the common part is initialized from the previous step and train the Fast R-CNN,

Sharing Features for RPN and Fast R-CNN

The whole architecture of the *Faster R-CNN* network is shown below:

- After getting the region proposals from the RPN network, we apply non-maximum suppression in order to reduce the number of proposals.
- The proposals are then fed to the RoI pooling layer followed by the rest architecture of the Fast R-CNN.

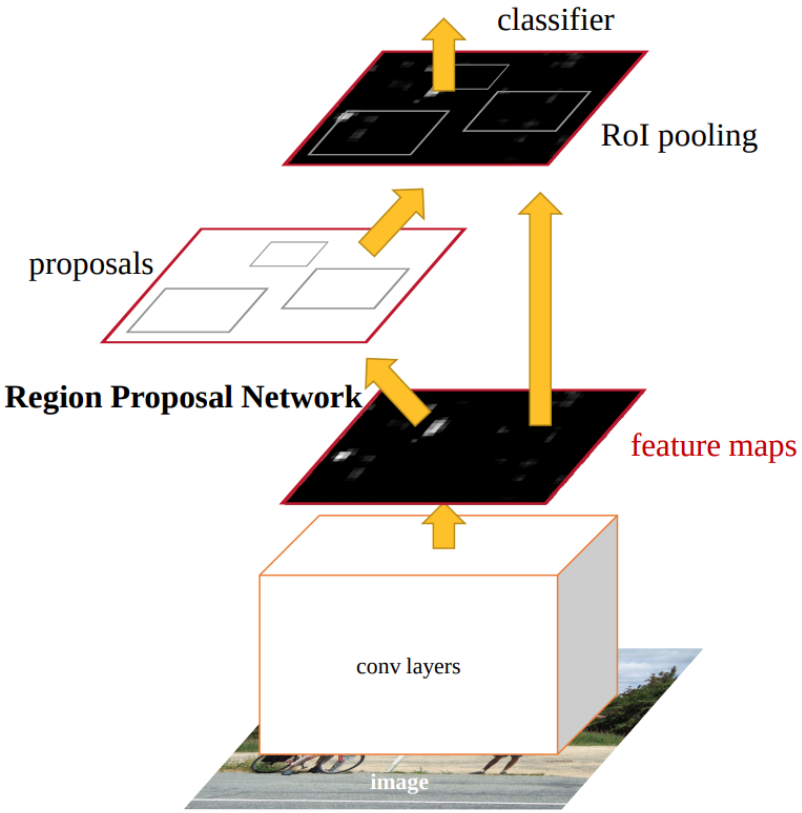


- The training of RPN and Fast R-CNN is done by the following way:
  - at first RPN is trained,
  - the common part is initialized from the previous step and train the Fast R-CNN,
  - the common part is initialized from the previous step and train the RPN,

Sharing Features for RPN and Fast R-CNN

The whole architecture of the *Faster R-CNN* network is shown below:

- After getting the region proposals from the RPN network, we apply non-maximum suppression in order to reduce the number of proposals.
- The proposals are then fed to the RoI pooling layer followed by the rest architecture of the Fast R-CNN.



- The training of RPN and Fast R-CNN is done by the following way:
  - at first RPN is trained,
  - the common part is initialized from the previous step and train the Fast R-CNN,
  - the common part is initialized from the previous step and train the RPN,
  - this process is iterated.

Faster R-CNN results

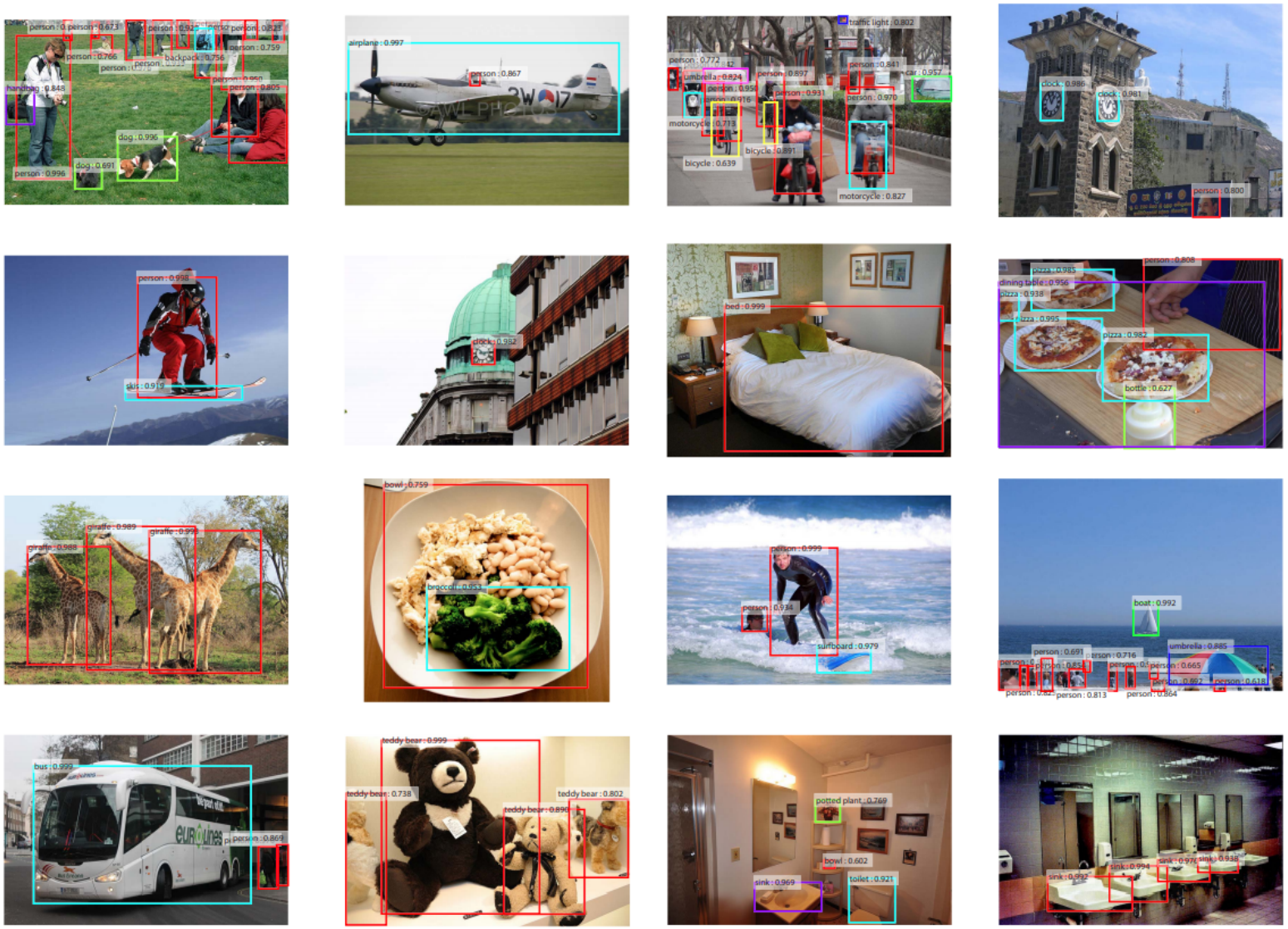


Figure: Some results of the Faster R-CNN algorithm.

Overview

Instance Segmentation: Mask R-CNN

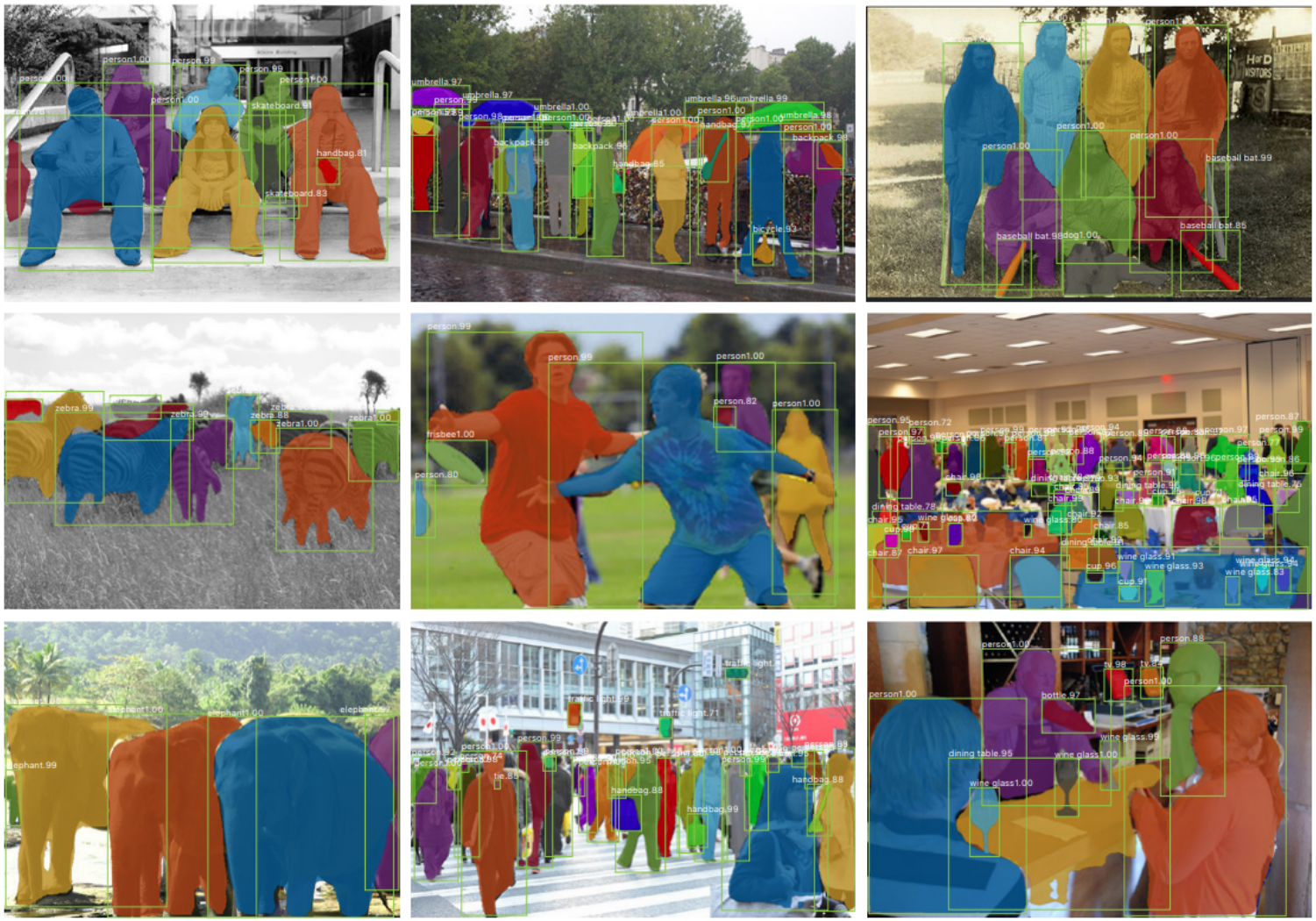
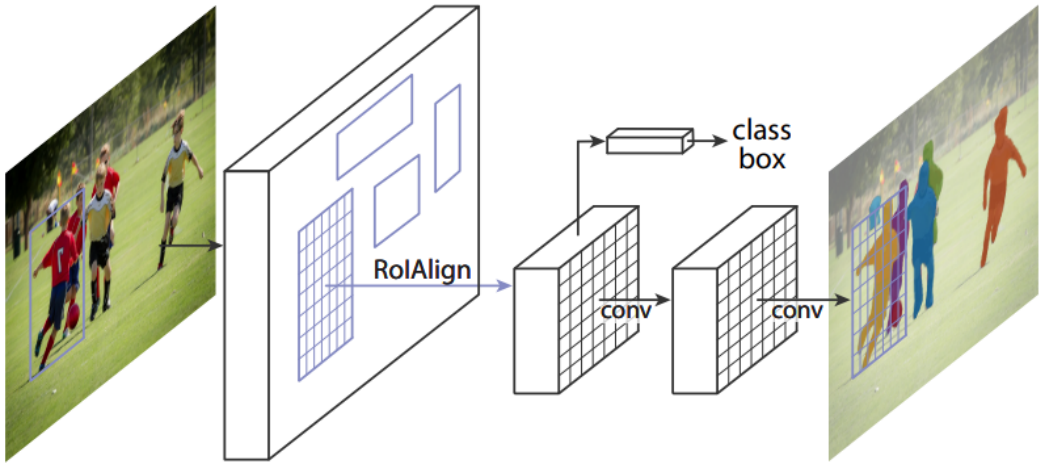


Figure: Some instance segmentation examples.

Mask R-CNN

The *Mask R-CNN* algorithm[12] extends the *Faster R-CNN* algorithm by an instance segmentation branch after extracting features for the *RoIPool* layer:



Note

Here instead of *RoIPool* layer we use *RoIAlign* layer.

[12] He, Kaiming, Georgia Gkioxari, Piotr Dollár and Ross B. Girshick. "Mask R-CNN." 2017 IEEE International Conference on Computer Vision (ICCV) (2017): 2980-2988.

RoIAlign

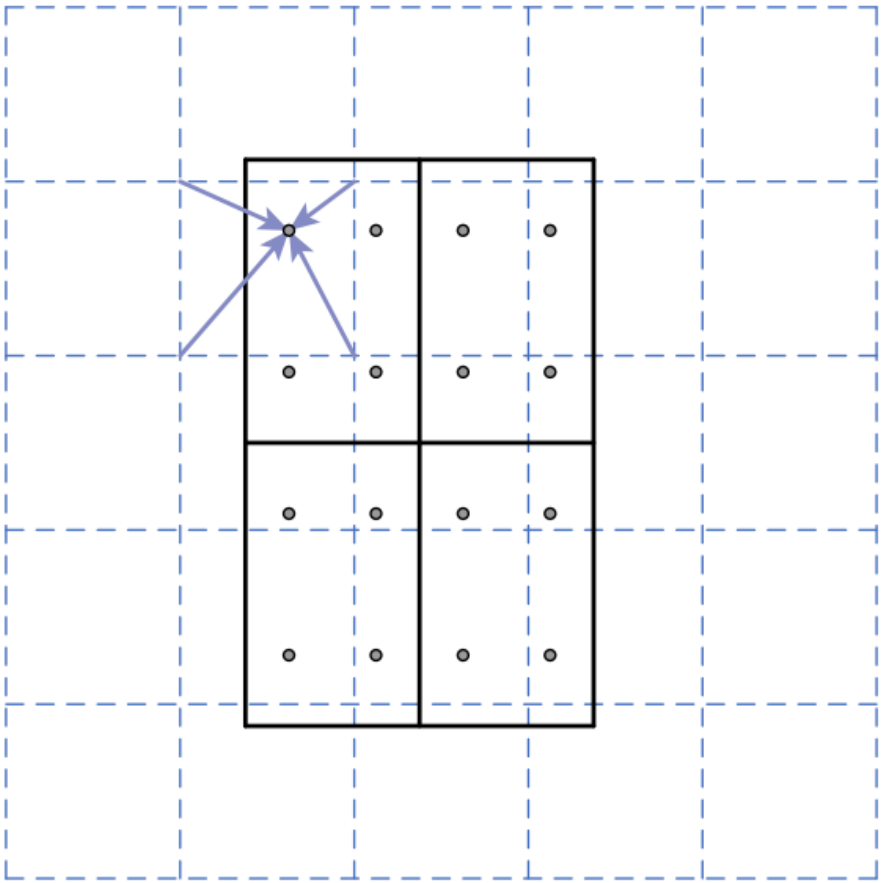


Figure: We use bilinear interpolation to compute the exact values of the input features at four *regularly sampled* locations in each RoI bin, and aggregate the result (using max or average)

Introduction to Computer Vision: Object Localization, Detection and Semantic Segmentation

Navasardyan Shant

December 19, 2019