Proposal networks in object detection

Förslagsnätverk för objektdetektering

Mikael Grossman

Abstract

Locating and extracting useful data from images is a task that has been revolutionized in the last decade as the computing power has risen to such a level to use deep neural networks with success. A type of neural network that uses the convolutional operation called convolutional neural network (CNN) is suited for image related tasks. Using the convolution operation creates opportunities for the network to learn their own filters, that previously had to be hand engineered. For locating objects in an image the state-of-the-art Faster R-CNN model predicts objects in two parts. Firstly, the region proposal network (RPN) extracts regions from the picture where it is likely to find an object. Secondly, a detector verifies the likelihood of an object being in that region.

For this thesis we review the current literature on artificial neural networks, object detection methods, proposal methods and present our new way of generating proposals. By replacing the RPN with our network, the multiscale proposal network (MPN), we increase the average precision (AP) with 12% and reduce the computation time per image by 10%.

Keywords

Deep learning, Machine learning, Computer vision, Applied mathematics, Statistics, Artificial Neural Networks, Object detection, Faster R-CNN, RPN, Proposal Network

Abstrakt

Lokalisering av användbar data från bilder är något som har revolutionerats under det senaste decenniet när datorkraften har ökat till en nivå då man kan använda artificiella neurala nätverk i praktiken. En typ av ett neuralt nätverk som använder faltning passar utmärkt till bilder eftersom det ger möjlighet för nätverket att skapa sina egna filter som tidigare skapades för hand. För lokalisering av objekt i bilder används huvudsakligen Faster R-CNN arkitekturen. Den fungerar i två steg, först skapar RPN boxar som innehåller regioner där nätverket tror det är störst sannolikhet att hitta ett objekt. Sedan är det en detektor som verifierar om boxen är på ett objekt .

I denna uppsats går vi igenom den nuvarande litteraturen i arificiella neurala nätverk, objekt dektektering, förslags metoder och presenterar ett nytt förslag att generera förslag på regioner. Vi visar att genom att byta ut RPN med vår metod (MPN) ökar vi precisionen med 12% och reducerar tiden med 10%.

Nyckelord

Maskininlärning, Neurala nätverk, Objektdetektering, Tillämpad matematik, Matematisk statistik, RPN, Förslags nätverk

Acknowledgements

I want to thank my supervisor Heydar Maboudi and Sarvai AB for making this thesis possible, his guidance, providing me code and hardware and for the hours he put in. I also want to thank my supervisor at KTH Timo Koski.

Glossary

CNN	Convolutional Neural Network
R-CNN	Regions with Convolutional Neural Network features
FC	Fully Connected layer
YOLO	You Only Look Once
RPN	Region Proposal Network
MPN	Multiscale Proposal Network
ANN	Artificial Neural Network
FPN	Feature Pyramid Network
IoU	Intersection over Union
RoI	Region of Interest
FCN	Fully Convolutional Network
SS	Selective Search
EB	Edge Boxes
GPU	Graphics Processing Unit
CPU	Central Processing Unit
ReLU	Rectified Linear Unit
MLP	Multi Layer Perceptron
AP	Average Precision
TP	True Positives
FP	False Positives
FN	False Negatives

Contents

1 Introduction

A world where almost everybody has a camera leads to millions of pictures being captured everyday. This leads to a growing interest of automated processes concerning images. Cars monitor the roads and Facebook uses filters to remove unappropriated uploads, are examples on such processes. These processes use machine learning to extract features from image to do predictions. This field is called computer vision.

1.1 Problem background

In the past few years, the field of computer vision has been more focused on using variations of Convolutional Neural Networks (CNNs) for modeling and solving different problems. CNNs have proven to be a fantastic tool for capturing the large variations seen within images and are able to extract more accurate information from the images.

In the classic formulation of the object detection problem, researchers used to build object models of a fixed size [1, 2]. To find objects of different sizes this model was applied to the image scale pyramid with a sliding window technique. This combination allowed them to find multiple instances within the image regardless of their size. In the modern formulation of object detection [4, 5, 6, 7, 8] this process is replaced with a series of convolutional layers. The first layer convolves on the image and learns the network to recognize patters (lines and corners). These patterns is then passed through the layers and learns the network to recognize more complex features as we get deeper. The final layer, the proposal layer, learns to use these patterns to suggest object locations. These suggestions may not be very accurate but other parts of the network can use them to produce accurate localization of the objects.

In this thesis, our goal is to do a study for understanding the effect of this proposal network on the overall accuracy of the models. These effects can be studied by measuring the accuracy of the models, both at the proposal stage and at the final stage. Ideally, one would like to find a correlation between the two benchmarks. To achieve this, we will focus on the Faster R-CNN framework [6, 7] and build different variations on the proposal network used by this model. This makes it possible to compare the performance of newly built models with the original formulation.

1.2 Related work

There exists many methods to generate proposals. Early models were based on grouping super-pixels, for example selective search [9] was used with great success. But since these methods generated a lot of proposals (around 5000 per image) the networks were very slow to run and train. To improve the training speed methods such as objectness in windows [10] and edge-boxes [11], implementing a sliding window approach was used. These methods generate significantly less proposals and improved the learning time and accuracy of the detection models. To improve this even further researches created a neural network to mimic selective search but much faster. The network is called region proposal network (RPN) and is the current state-of-the-art. It uses sliding window which slides a small neural network over the convolutional feature map output by the last shared convolutional layer. More of the details is explained in chapter 3.

1.3 Goal

In this thesis we review the current literature on object detection and focuses on comparing different approaches of the proposal network. The goal of the project is to build a better way of handling the RPN in the Faster R-CNN framework with respect to running time and accuracy. The RPN is using pre-defined anchors and for each anchor a score of likelihood of being on an object or not is calculated. The boxes are refined so they cover more of the objects through bounding box regression. The performance on the RPN depends on which scales and aspect ratios is used. We are going to optimize the RPN and compare the results to our proposal network.

The idea behind our proposal network is to remove the use of anchors and let the network find proposals directly on the feature maps. To approach the multiscale problem, we split the network into 9 sections, where each section is trained to propose objects of a fixed size and aspect ratio.

2 Background ANN

This chapter contains the background needed for understanding the problem of object detection. It introduces machine learning terminology and artificial neural networks. Lastly it explains convolutional neural networks.

2.1 Introduction to Machine learning

The term machine learning comes from Arthur Samuel in 1959 [12]. Since then researches have been interested in having machines learning from data. Today machine learning can be classified into three categories, namely supervised learning, unsupervised learning and reinforcement learning [13].

In supervised learning, the goal is to map a set of input variables $x_1...x_n$ to a set of output variables $y_1...y_n$ and use this to predict the outputs for new data [16]. Learning is done using objective functions which depend on target values. The output could be a class label (classification problem) and a real number (regression problem).

In unsupervised learning the machine receives inputs $x_1...x_n$ without target outputs, nor reward from its environment. It seems hard to establish a model that can learn without outputs. But instead of mapping inputs with outputs unsupervised learning finds patterns in the given data. For example, clustering and dimensionality reduction are two examples of unsupervised learning [17].

The last one, reinforcement learning, is about learning what to do and how to map situations to actions. Given a reward signal the learner is not told which actions to take, but instead must discover which actions yield the most rewards by trying them. Sometimes the action may affect not only the immediate reward, but also the next reward, through that, all rewards. These two types are called trial-and-error search and delayed reward and are the most important features in reinforcement learning [18]. For example, alpha zero (Deepminds chess engine from Google) is using trial-and-error search to learn by itself to come up with the optimal strategy to win against the opponent by playing millions of games against itself.

2.2 Artificial neural networks (ANN)

ANN is a framework that can approximate any continuous function given the inputs and the outputs of the function [23]. There should be a mathematical function that given an image, outputs the location of the people in that image. This function is not known but we can approximate it. The ANN can learn to predict locations of people by considering thousands of example images containing the location of the people. By designing an accurate loss function, that compares the ANNs output with the desired output, each weight in the nodes gets updated accordingly. The nodes therefore learns to send an accurate signal to the next nodes and can then predict the output.

2.2.1 Perceptron

The simplest kind of a neural network is a single-layer perceptron that was invented by Frank Rosenblatt in 1957 [20]. It contains a single neuron and it is explained in the following way (see figure 1). Given a signal containing $x_1 ... x_n$ and a set of weights $w_1 ... w_n$ we want to map to a given output $z_1 ... z_n$. The signal and the weights are sent through a single neuron with the equation:

$$y(\boldsymbol{x}; \boldsymbol{w}, b) = \mathcal{G}(\sum_{i=1}^{n} w_i x_i + b) = \mathcal{G}(\boldsymbol{w}\boldsymbol{x}^T + b) \tag{1}$$

Here b is the bias and $\mathcal{G}$ is the activation function to add complexity in the network (more of this in section 2.2.3). In the original case the activation function is a simple step function given by:

$$\mathcal{G} = \begin{cases} 1 & \text{if } \boldsymbol{w}\boldsymbol{x}^T + b > 0 \\ 0 & \text{else} \end{cases} \tag{2}$$

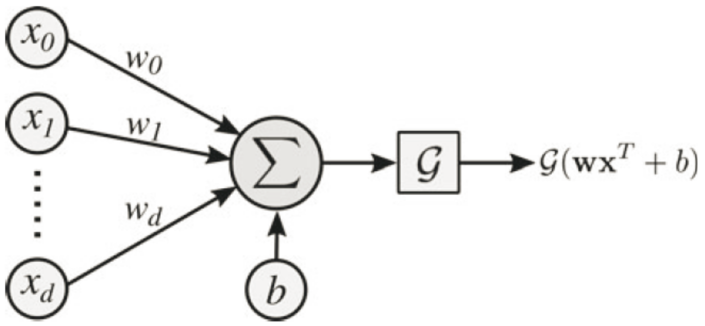


Figure 1: A single layer perceptron here $\mathcal{G}$ is referring to the activation function. [13].

The learning algorithm of a single-layer perceptron follows these steps

1. Define your training set containing $\boldsymbol{x}_1 ... \boldsymbol{x}_n$ and the desired output $z_1 ... z_n$
2. Initialize the weights to a random small number
3. For each $\boldsymbol{x}_j$ in our training set calculate the output $y(\boldsymbol{x}_j) = \mathcal{G}(\boldsymbol{w}\boldsymbol{x}_j^T + b)$ and update the weights accordingly

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + (z_j - y(\boldsymbol{x_j}))\boldsymbol{x}_j \tag{3}$$

4. Repeat 3 until the loss $\sum_{j=1}^{n}(z_j - y(\boldsymbol{x_j}))$ is minimized

2.2.2 Multilayer perceptron

Multilayer perceptron (MLP) is the most utilized model in neural network applications. It uses the so called back-propagation algorithm for learning [22]. In most of the cases the connections between the neurons do not form a cycle and are then called a feed-forward neural network. Figure 2 shows an example of such a network. Here $\mathcal{G}$ refers to the activation function, $x_0 ... x_d$ is the input signal, $\boldsymbol{W}^i$, b_i is a weight matrix and bias in the i^{th} layer, I the input layer, H_i hidden layers, Z the output layer, d_1, d_2 and d_z refers to the number of neurons in each layer. Using this, the network in figure 2 can be represented with the following equation [13].

$$y(\boldsymbol{x}; \boldsymbol{W^1}, \boldsymbol{W^2}, \boldsymbol{W^3}, b_1, b_2, b_3) = \mathcal{G}(\mathcal{G}(\mathcal{G}(\boldsymbol{x}\boldsymbol{W^1} + b_1)\boldsymbol{W^2} + b_2)\boldsymbol{W^3} + b_3) \tag{4}$$

A feed-forward network consists of an input layer I, hidden layers H_i and an output layer Z in which each layer contains neurons. Any layer between input layer and output layer is called hidden layers. For n number of layers equation (4) becomes:

$$\begin{cases} \boldsymbol{z^1} & = \mathcal{G}(\boldsymbol{x}\boldsymbol{W^1} + b_1) \\ \boldsymbol{z^2} & = \mathcal{G}(\boldsymbol{z^1}\boldsymbol{W^2} + b_2) \\ & \vdots \\ y(\boldsymbol{x}; \boldsymbol{W^1}, ..., \boldsymbol{W^n}, b_1, ..., b_n) & = \mathcal{G}(\boldsymbol{z^{n-1}}\boldsymbol{W^n} + b_n) \end{cases}$$

What makes this so special is that a feed-forward network with one layer and finite number of neurons can approximate any continuous function [23].

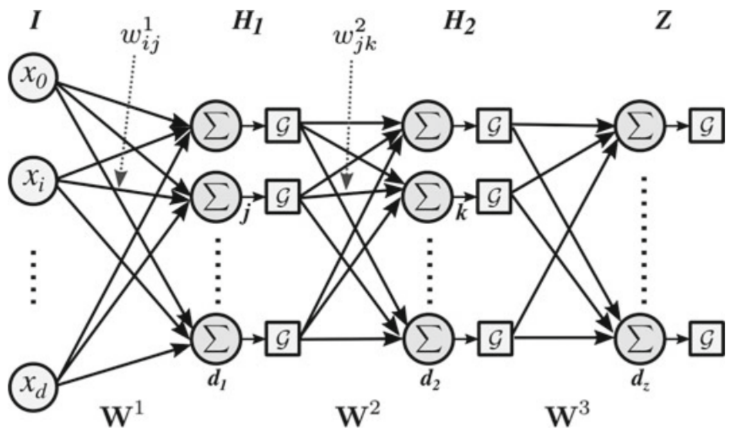


Figure 2: A multilayer perceptron. For simplicity the bias is not included in the figure, but do remember that every node still has a bias [13].

2.2.3 Activation functions

There exist different kinds of activation functions and the choice varies with the problem at hand. Here we are going to present the most common ones.

The linear activation function

$$\mathcal{G}(x) = x \tag{5}$$

Sigmoid activation function

$$\mathcal{G}(x) = \frac{1}{1 + exp(-x)} \tag{6}$$

Rectified Linear Units (ReLU)

$$\mathcal{G}(x) = max(0, x) \tag{7}$$

Noisy ReLU

$$\mathcal{G}(x) = max(0, x + N(0, \sigma(x))) \tag{8}$$

Here N is the normal distribution with mean 0 and $\sigma(x)$ is the variance function.

Tanh Activation Function

$$\mathcal{G}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{9}$$

2.3 Training ANNs

A feed-forward network is used to find the optimal weights for the given problem, to map the input signal to the output signal. Updating the weights can be a difficult task. A simple method to update the weights for a neural network with only one neuron using a sigmoid activation function (f) is the stochastic gradient descent. By minimizing the quadratic loss $(y(\boldsymbol{x}_i; \boldsymbol{w}, b) - z_i)^2$ where $y(\boldsymbol{x}_i; \boldsymbol{w}, b) = f(\boldsymbol{w}^T\boldsymbol{x}_i + b)$ we want to update the weights accordingly.

$$w_k = w_k - \alpha \Delta w_k \qquad \text{for all } k \tag{10}$$

$$b = b - \alpha \Delta b \tag{11}$$

Here α is the learning rate (usually a small number 0.05). The gradient is a good descent direction for a function and can be applied in our case.

$$\begin{aligned} \Delta w_k &= \frac{\partial}{\partial w_k}(f(\boldsymbol{w}^T\boldsymbol{x}_i + b) - z_i)^2 \\ &= 2(f(\boldsymbol{w}^T\boldsymbol{x}_i + b) - y_i \frac{\partial}{\partial w_k}(f(\boldsymbol{w}^T\boldsymbol{x}_i + b))^2 \end{aligned} \tag{12}$$

Using the chain rule and the fact that $\frac{df}{dx} = [1 - f(x)]f(x)$.

$$\begin{aligned} \frac{\partial}{\partial w_k}(f(\boldsymbol{w}^T\boldsymbol{x}_i + b))^2 &= \frac{\partial f(\boldsymbol{w}^T\boldsymbol{x}_i + b)}{\partial(\boldsymbol{w}^T\boldsymbol{x}_i + b)} \frac{\partial(\boldsymbol{w}^T\boldsymbol{x}_i + b)}{\partial w_k} \\ &= [1 - f(\boldsymbol{w}^T\boldsymbol{x}_i + b)]f(\boldsymbol{w}^T\boldsymbol{x}_i + b)x_i^k \end{aligned} \tag{13}$$

Inserting this into equation (12), we get:

$$\Delta w_k = 2[f(\boldsymbol{w}^T \boldsymbol{x}_i + b) - y_i][1 - f(\boldsymbol{w}^T \boldsymbol{x}_i + b)]f(\boldsymbol{w}^T \boldsymbol{x}_i + b)x_i^k \tag{14}$$

Similarly for the bias:

$$\Delta b = 2[f(\boldsymbol{w}^T \boldsymbol{x}_i + b) - y_i][1 - f(\boldsymbol{w}^T \boldsymbol{x}_i + b)]f(\boldsymbol{w}^T \boldsymbol{x}_i + b) \tag{15}$$

Now we have defined the stochastic gradient update algorithm in this specific case [25].

2.3.1 Back propagation

In reality, we want to be able to calculate the gradient descent for more than a single node and for more layers. For doing this the back propagation algorithm is used. For deriving the algorithm we are using a quadratic loss function and a sigmoid activation. Other functions can be derived in a similar way.

$$L = \frac{1}{2}\sum_{j=1}^{m}(y_j - z_j)^2 \tag{16}$$

Here $y_j = f(v_j)$, $v_j = b + \sum_{k \in K_j} w_{kj}x_k)$, $f(x)$ is a sigmoid activation function, K_j is the set of nodes from the k^{th} layer which feed node j. Be aware that this function only works if j is an output node since we only know z_j in the output. Then the weight change becomes:

$$\Delta w_{kj} = -\alpha \frac{\partial L}{\partial w_{kj}} \tag{17}$$

where α is the learning rate. Expanding the partial derivative, we get:

$$\frac{\partial L}{\partial w_{kj}} = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial v_j}\frac{\partial v_j}{\partial w_{kj}} \tag{18}$$

And we know

$$\frac{\partial v_j}{\partial w_{kj}} = x_k \tag{19}$$

We define an error term for simplicity.

$$\delta_j = -\frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial v_j} \tag{20}$$

Using the fact that $y_j = f(v_j)$ we get from deriving the sigmoid function

$$\frac{\partial y_j}{\partial v_j} = y_j(1 - y_j) \tag{21}$$

Then for the first term in equation (18) if j is the output layer.

$$\frac{\partial L}{\partial y_j} = \frac{\partial}{\partial y_j}\frac{1}{2}\sum_{j=1}^{m}(z_j - y_j)^2 = -(z_j - y_j) \tag{22}$$

If j is a hidden layer the equation gets a little trickier.

$$\frac{\partial L}{\partial y_j} = \sum_{i \in I_j} \frac{\partial L}{\partial y_i} \frac{\partial y_i}{\partial v_i} \frac{\partial v_i}{\partial x_j} \tag{23}$$

Recall equation (20) and

$$\frac{\partial v_i}{\partial x_j} = w_{ji} \tag{24}$$

thus

$$\frac{\partial L}{\partial y_j} = -\alpha \sum_{i \in I_j} \delta_i w_{ji} \tag{25}$$

and we gain the gradient for the hidden layer case:

$$\frac{\partial L}{\partial w_{kj}} = -\alpha \sum_{i \in I_j} \delta_i w_{ji} y_j (1 - y_j) x_k \tag{26}$$

and for the output layer

$$\frac{\partial L}{\partial w_{kj}} = -(z_j - y_j) y_j (1 - y_j) x_k \tag{27}$$

Each node in the equation will calculate its own error term [26]. Lets explain the steps of the algorithm

1. Perform a feed forward pass to compute y_j
2. Compute δ_i for the output layer O
3. Perform a backward pass for each layer $i = O - 1, O - 2, ..., 2$ and calculate δ_i
4. Set the partial derivatives as equation (26) and (27) and update the weights [25].

2.3.2 Loss function

The most common loss function is the quadratic loss. Here I presents some other common ones.

- The mean squared error also called the quadratic cost or L2 loss is the most common one used in machine learning for regression problems.

$$L = \frac{1}{2} \sum_{j=1}^{n} (y_j - d_j)^2 \tag{28}$$

The gradient of this lost function can be written as

$$\nabla L = \begin{bmatrix} y_1 - d_1 \\ y_2 - d_2 \\ \vdots \\ y_n - d_n \end{bmatrix} \tag{29}$$

- Cross-entropy loss
$$L = -\sum_j [d_j \ln y_j + (1-d_j)\ln(1-y_j)] \tag{30}$$
The gradient of this lost function can be written as
$$\nabla L = \begin{bmatrix} \frac{(y_1-d_1)}{(1-y_1)y_1} \\ \vdots \\ \frac{(y_n-d_n)}{(1-y_n)y_n} \end{bmatrix} \tag{31}$$

- L1 loss, also called absolute error, is similar to the quadratic loss except it calculates the absolute value instead of the square.
$$L = \frac{1}{2}\sum_{j=1}^{n} |y_j - d_j| \tag{32}$$
The gradient of this lost function can be written as
$$\nabla L = \begin{bmatrix} \frac{y_1-d_1}{|y_1-d_1|} \\ \vdots \\ \frac{y_n-d_n}{|y_n-d_n|} \end{bmatrix} \tag{33}$$

- Smooth L1 error is a smooth version of the absolute error. It uses a squared term if the squared error falls below 1 and uses the L1 loss otherwise. It is less sensitive to outliers than the Mean Squared Error and it can also prevent exploding gradients.
$$L = \sum_{j=1}^{n} smooth_{L1}(y_j - d_j) \tag{34}$$
With
$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{35}$$

- Softmax is usually used in the end of an ANN to classify which object is seen in the picture. It takes a vector of scores and reduces it to a vector that sums to one containing values between zero and one. Hence, it gives us a probability vector of the classes.
$$L_j = -log(e^{y_j} / \sum_j e^{y_j}) \tag{36}$$
$$L = \frac{1}{N}\sum_{j=1}^{N} L_j \tag{37}$$

2.3.3 Preventing overfitting

There are many variables and methods that can be used to shorten the learning time and decrease the risk of the model overfitting to the given data. In most of the cases one has to choose the model for a given problem. Here follows a survey of some methods and variables that are important to understand.

- The learning rate α decides how much of the gradient the weights will change. If the learning rate is a high, there is a risk for oscillation around the minimum. If α is to low it will take a longer time for the model to learn. There are different methods for choosing the learning rate. Some use a learning rate that varies depending on how stable the gradient is.
- The number of hidden layers depends on the problem. More than 2 hidden layers are needed to learn complex representations such as object detection.
- The number of neurons affect how well the model will fit to the data. Too many will increase chances of overfitting and too few will result in a model that does not fit at all, which is called underfitting.
- Batch size defines the number of samples that are going to be propagated through the network. If you have 1000 samples, to decrease the training time one can choose to train the network on 10 samples at a time. Then the batch size is 10.
- The number of epochs decides how many times the weights are going to be updated. Too many epochs results in the network overfitting to the data and too few results in underfitting.
- Dropout is a method that is used to decrease the chance of the network overfitting to the given sample. By randomly canceling out some weights (setting the weight to 0) while training, the neurons are less likely to cooperate.
- L_p-regularisation idea is to punish large weights as they tend to result in overfitting. By adding an extra term to the loss function the new loss becomes

$$L_p = L + \frac{\lambda}{2} w^T w \tag{38}$$

- Weight sharing is an idea to reduce the number of parameters in a system. By having identical weights for different units within each layer.

2.4 Computer vision

The goal of computer vision is to model the human visual system. It aims to extract meaningful information from a photo or a video. By doing so it aims to surpass humans. In the case of object detection computers has already surpassed humans given a training set. But a human gains a lot more by looking at a picture than where and what the object is. It can see poses, expressions and what is actually going on in the picture.

2.4.1 Using ANN for object detection

The task of detecting multiple objects in a picture is a classical problem in computer vision. We cannot use a standard fully connected feed-forward ANN for learning features and classifying data to solve even the simplest tasks. A picture of good quality has a resolution of 1920*1080 pixels and thus needs at least equally many weights as pixels resulting in a very big network. Another problem is that the network is not translation invariant meaning that an object in the top left corner will be subjected to other parameters compared to the object in the bottom right. We would need an insane amount of data to train such a network. To solve this problem convolutional neural networks are used instead. It reduces the number of parameters resulting in a more efficient neural network [13]. It uses the convolutional operation to extract important features from the image. This operation is defined as:

$$f(t) * g(t) \overset{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{39}$$

and in the discrete case where we have an image f and a filter matrix g:

$$f[x,y] * g[x,y] = \sum_{n}\sum_{m} f(n,m)g(x-n,y-m) \tag{40}$$

This convolution operation has also some interesting features including linear time invariant and linear shift invariant, which makes it great for using on pictures. [14]

2.5 Convolutional neural networks (CNN)

Using the convolution operation creates opportunities for the network to learn their own filters, that previously had to be hand engineered, to extract useful features to solve the task. This approach has revolutionized the computer vision field. In the next sections we will introduce the most common layers used in CNNs. An example of a CNN is illustrated in figure 3.

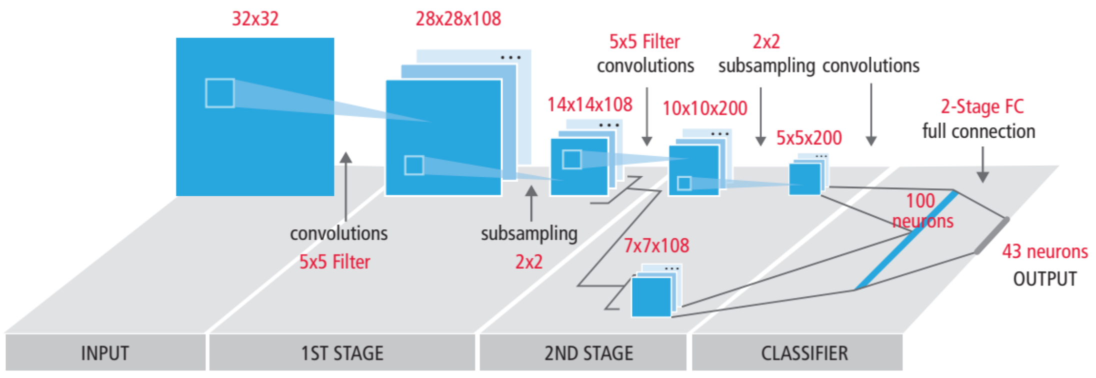


Figure 3: A block diagram of a convolution neural network [29].

2.5.1 Convolution layer

The convolution layer (l) takes as input $m^{(l-1)}$ feature maps ($Y^{(l-1)}$) from the previous layer or the full image if $l = 1$ and convolves new feature maps, here denoted Y_i^l for the i^{th} feature map.

$$Y_i^l = f(B_i^l + \sum_{j=1}^{m^{(l-1)}} K_{i,j}^l * Y_j^{l-1}) \tag{41}$$

Here B_i^l is a bias matrix, f is the activation function and $K_{i,j}$ is the filter containing learnable weights [27].

2.5.2 Pooling layer

The pooling layer is referred to the downsampling or subsampling layer. The most common one is the max-pooling layer. It takes a filter of size m^2 (normally m=2) and convolves on the feature map of size n^2 and takes the maximum value of every subregion the filter convolves around giving an output of size $\frac{n^2}{m^2}$ resulting in a smaller feature map.

2.5.3 Residual building block

Gradient vanishing is a common problem of a CNN with many layers (as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient infinitely small). To make sure that the network is deep enough (has enough layers) a shortcut is introduced where the signal can jump over one or more layers. This makes it possible for the signal to skip layers that do nothing without affecting the back-propagation algorithm [43].

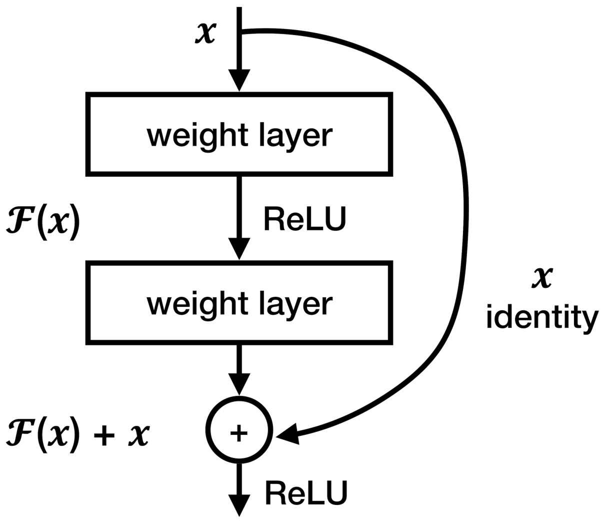


Figure 4: Shows a residual building block. Making it possible for the signal x to skip one or several layers. [43].

2.6 Training CNN on an image dataset

The data needs to be preprocessed, before training can begin. A technique called mean-variance normalization is the most common one. With this technique, the mean image is subtracted from every image and divided by the variance which results in a normalized sample.

2.6.1 Initialization

It is important to know that putting all weights initially to zero will not wok. All weights will be updated equally throughout the back propagation and will not be able to learn. A better way of initializing the weights is to randomly generate them from a normal distribution with mean zero and a small variance. This is called random initialization and is the most common one used [13].

2.6.2 Back-propagation algorithm in CNN

To better understand the back-propagation in CNN, it is recommended to read the example derivation by Zhifei Zhang [28]. Here I will only present the algorithm.

1. Initialize the weights randomly
2. Present the input signal $z^1_{x,y}$ to the model
3. Perform a forward pass for each $l = 2, 3, ..., O$ compute $z^l_{x,y} = w^l * f(z^{(l-1)}_{x,y}) + b^l_{x,y}$ and the corresponding activation $a^l_{x,y} = f(z^l_{x,y})$
4. Compute the output error $\delta^O = \nabla_a L \cdot f'(z^O_{x,y})$
5. Back-propagate the error for each $l = O - 1, O - 2, ..., 2$ compute $\delta^l_{x,y} = \delta^{l+1} * ROT180(w^{l+1}_{x,y})f(z^l_{x,y})$
6. Calculate the gradient of the loss function $\frac{\partial L}{\partial w^l_{x,y}} = \delta^l_{x,y} * f(ROT180(z^{l-1}_{x,y}))$ and update the weights accordingly.
7. Redo until the loss is minimized

3 Convolutional object detection

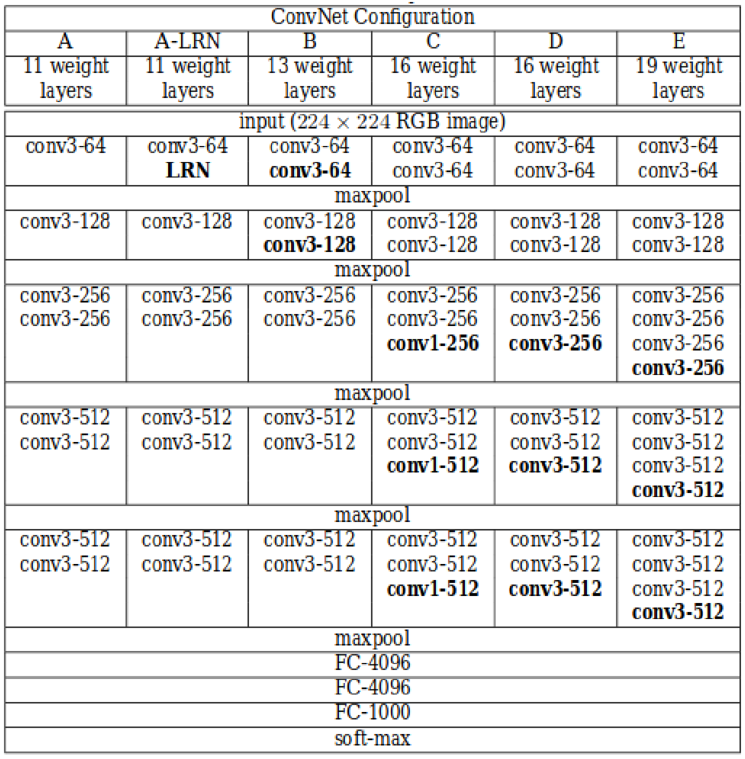
An object detector consist of three parts, feature extractor, region proposal and classifier. In this chapter, I will first cover the most common ways constructing the feature extractors and region proposals. Then I will cover the main techniques of modern object detection and see how they have evolved. I will discuss the differences and how to improve them.

3.1 Feature extractors

The feature extractors have become more complex since the graphic cards have become better. The residual building block (as explained in 2.3.4) allowing the network to become even more complex. Here are the three most common ones used.

3.1.1 VGG

VGG net was created in 2014 by Karen Simonyan and Andrew Zisserman [42]. It is a maximum 19 layers network that strictly uses 3x3 filters with a stride length of 1 and 2x2 maxpooling layers with a stride length of 2. Figure 5 shows how the layers in the VGG network are constructed. The most common one used is the VGG-16 shown in the D column.



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 5: Shows the VGG network structure [42].

3.1.2 ResNet

ResNet from 2015 is a 152 layer network [43]. Since the residual building block allows the network to jump over layers, it results in an ultra deep network without overfitting issues.

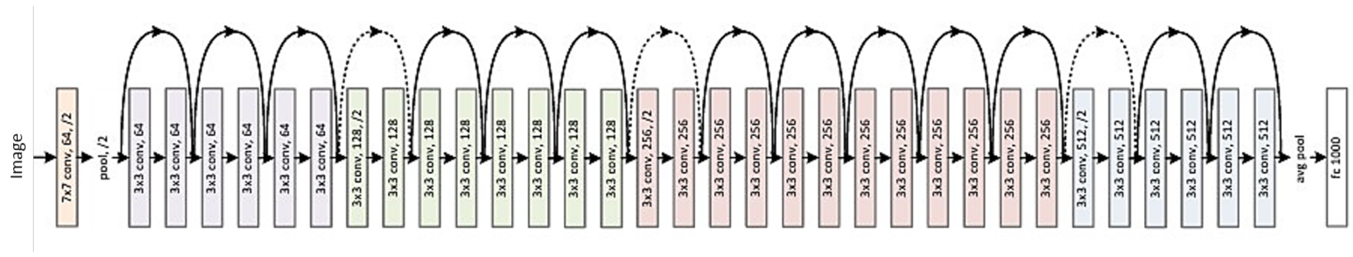


Figure 6: Shows the ResNet network structure [43].

3.2 Region proposal algorithms

The region proposal role is to suggest a boxes (RoIs) to the classifier and regressor to check the occurrence of objects. At first, the algorithm was hard coded to look for important features in the picture. Then research suggested that the best way is to just let the NN learn to find proposals by itself.

3.2.1 Sliding window

A sliding window is a naive approach to object detection. By using a window of a fixed size slide through an image in different scales (image pyramid), it tries to answer if the box contains an object or not. If this is the case, the box is generated and sent as proposal to the network. The main problem with this approach is that the window is of a fixed size and thus needs a lot of images in different scales making it very time consuming. It also analyzes more false candidates then positives making it class imbalance.

3.2.2 Selective search

Selective search [9] uses a Hierarchical Grouping algorithm to identify objects of varied scales, colors, textures and enclosures. To get a set of starting regions the writer uses Felzenszwalb and Huttenlocher method that segments the image into regions [39]. The grouping algorithm is shown in Algorithm 1.

```
Input: (colour Image)
Output: Set of object location hypotheses L
Obtain initial regions $R = r_1, ..., r_n$ using [39]
Initialize similarity set $S = \emptyset$
foreach Neighboring region pair $(r_i, r_j)$ do
    Calculate similarity $s(r_i, r_j)$
    $S = S \cup s(r_i, r_j)$
end
while $S = \emptyset$ do
    Get highest similarity $s(r_i, r_j) = max(S)$
    Merge corresponding regions $r_t = r_i \cup r_j$
    Remove similarities regarding $r_i : S = S \setminus s(r_i, r_*)$
    Remove similarities regarding $r_j : S = S \setminus s(r_*, r_j)$
    Calculate similarity set $S_t$ between $r_t$ and its neighbours
    $S = S \cup S_t$
    $R = R \cup r_t$
end
Extract object location boxes L from all regions in R
```

Algorithm 1: Hierarchical Grouping Algorithm [9]

3.2.3 Edge-boxes

The authors of edge boxes [11] recognized that the number of edge contours wholly enclosed by a bounding box correlates with the likelihood that the box contains an object. they use a filter that draws edges of a figure and groups the pixels. They perform a sliding window on the resulting future map calculating an object proposal score. The score is computed by summing the edge strength of edge groups that lie completely within the box and subtracting the strength of edge groups that are part of a contour that cross the box boundary. Regions with high scores are then further refined.

3.2.4 Region Proposal Network (RPN)

RPN [6] compared to the previous methods let the CNN learn to generate feature maps to get good predictions. This approach, saves time and improves the detection rate.

The method works by sliding a small window of size n^2 (usually $n = 3$) over the feature map and feed it to a small network. Each window is mapped to a lower-dimensional feature. For each window a set of k anchors (usually $k = 9$) are generated which are located at the center of the sliding window but with different aspect ratios ($1 : 1$, $1 : 2$ and $2 : 1$) and scales (128^2, 256^2 and 512^2). For each of these anchors a value p^* is calculated.

$$p^* = \begin{cases} 1 & \text{if } IoU > 0.7 \\ -1 & \text{if } IoU < 0.3 \\ 0 & \text{otherwise} \end{cases} \tag{42}$$

Here IoU refers to the intersection over union between a predicted bounding box (P_b) and the location of the object given by a ground truth bounding box (G_t).

$$IoU(P_b, G_t) = \frac{P_b \cap G_t}{P_b \cup G_t} \tag{43}$$

This is then fed into two sibling layers, a box regression layer (reg) and a box classification level (cls). At each sliding window they simultaneously predict multiple region proposals. So the reg layer has 4k outputs denoting the coordinates (x, y, w, h) of the k boxes. And the cls layer is a two-class soft max layer and has 2k outputs that estimates the probability of object or not object for each proposal. A demonstration of the RPN is shown in figure 7.

The RPN consist of two losses as seen in equation below.

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{44}$$

Here i is the index of the anchor, p_i the predicted probability of anchor i being an object, t_i is a vector represented 4 parameterized coordinates of the predicted bonding box, L_{cls} is the log loss over two classes (object vs not object) and L_{reg} is

$$L_{reg}(t_i, t_i^*) = R(t_i - t_i^*) \tag{45}$$

and R is the robust smooth L_1 loss function

Figure 8 shows a comparison on how these different region proposal methods compares on the Pascal VOC 2007 [45] test set. Here SS is the selective search algorithm, EB is the edge-boxes algorithm, RPN ZF stands for RPN with a Zeiler and Fergus [41] CNN and RPN VGG with a Simonyan and Zisserman [42] CNN. The recall is the amount of objects detected with a bigger detection score than the threshold (IoU) divided by number of objects. As seen in the figure, RPN is giving a good result with far less proposals per image than EB and SS resulting in a faster network.

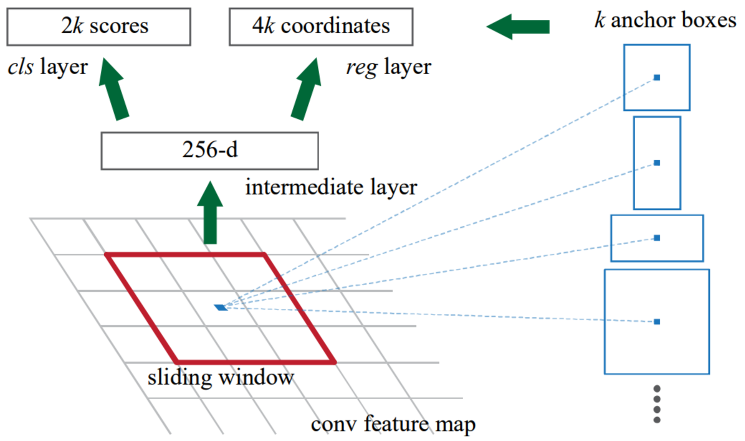


Figure 7: Shows how the RPN generates proposals from a feature map [6].

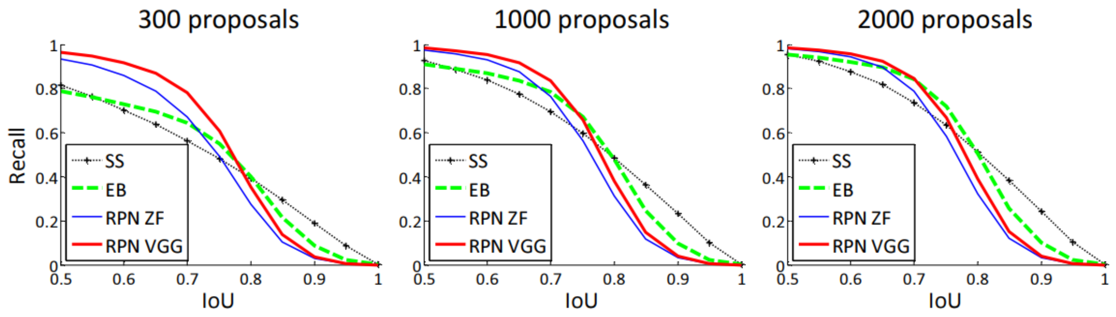


Figure 8: A comparison on the detection rate versus intersection over union threshold on the Pascal VOC 2007 test set. [6, 45].

3.3 Previous object detection methods

Image feature extraction is the key to object detection. Better features results in a wide array of computer vision tasks to be possible. With Scale-invariant feature transform (SIFT) [30] and Histogram of oriented gradients (HOG) [2] being the last revolution of machine learning techniques prior to the implementation of CNN. Better understanding of GPU programming and faster GPUs results in researchers being able to train new and more advanced models. This results in new improved models being released on a regular basis.

3.3.1 Regions with Convolutional Neural Network Features (R-CNN)

R-CNN from 2014 takes as input an image and uses selective search to generate a lot of object like boxes called Regions of Interests (RoI). With these regions the pictures are trained on a CNN to extract features of the boxes. In the last stage the object is classified and the bounding boxes are being tightened around the object. [4] This is illustrated in figure 9.

The drawback of this model is that every picture generates around 2000 proposals and every proposal has to be forward passed in the CNN. This results in a very time consuming training process. It also has three different stages that have to be trained separately, resulting in a pipeline hard to train. To improve the model the researcher Ross Girschick designed Fast R-CNN (2015).

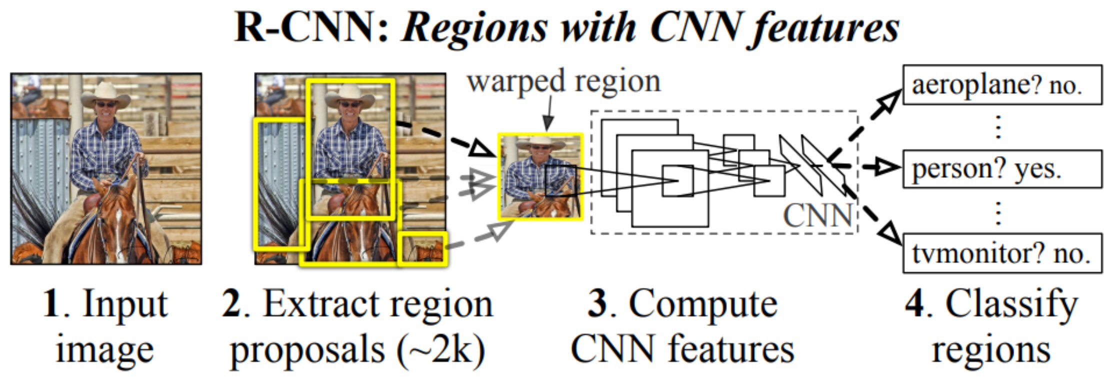


Figure 9: An illustration on the different stages of a R-CNN model [4]. From the image selective search is used to generate proposals to the CNN. From the feature maps a classifier is used to identify the objects.

3.3.2 Fast R-CNN

Fast R-CNN implements the CNN first and then shares that computation across the 2000 proposals. This increases the speed of the learning. To make this possible a max pooling layer called RoI Pooling (RoIPool) that shares the CNN for an image across its subregions is used by converting the features inside any valid RoI into a small feature map [5]. This results in a single system and not three separate ones. As image 10 shows, Fast R-CNN compared to R-CNN calculates the CNN first and then use the RoIs to classify the objects.

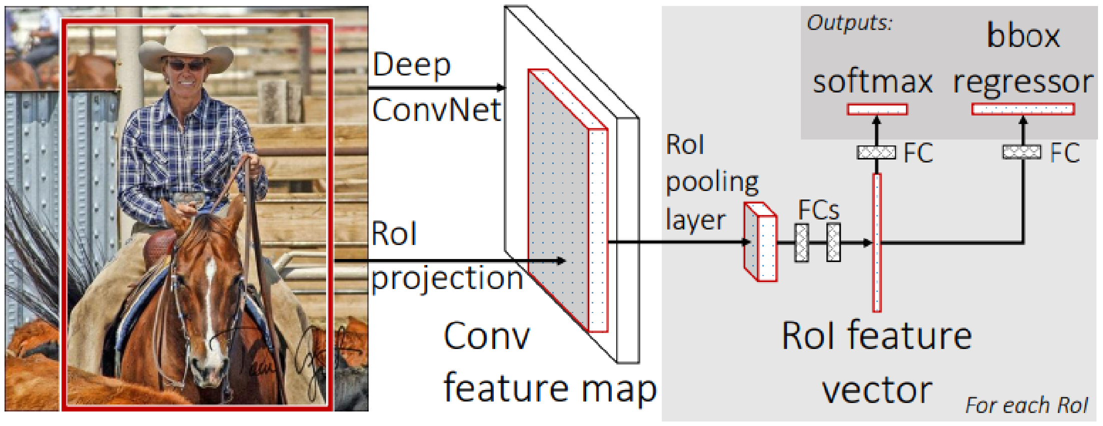


Figure 10: An illustration of the different stages of a Fast R-CNN model [5]. From the feature map selective search is used to generate proposals to the classification stage.

3.4 Modern two stage detectors

A two stage detector (region based detector) contains a region proposal network that suggests RoI to a classifier that decides what object it is and adjusts the region accordingly.

3.4.1 Faster R-CNN

In 2015, researchers came up with the idea to generate proposals from future maps to reduce the number of proposals [6]. Adding almost cost free proposals compared to the selective search method by using a Region Proposal Network (RPN) that tells the Fast R-CNN detector where to look. Figure 11 shows how the feature maps are used to generate proposals to the classifier. From the proposals (see part 3.2.4) the classifier identifies the class of the object and uses a bounding box regression to improve the proposal boxes.

3.4.2 Feature pyramid network (FPN)

It is hard for detectors to detect objects in different scales, especially small objects. It is possible to re-scale the images and feed it to the network, but that is very time consuming. FPN from 2017 is a feature extractor for detectors like Faster R- CNN. It generates multiple feature maps (multi-scale feature maps) with better information than the regular feature pyramid [35]. The top-down pathway makes lower dimension features able to use information from later features to make predictions. This makes the network able to remember more information that may be forgotten in a later stage.

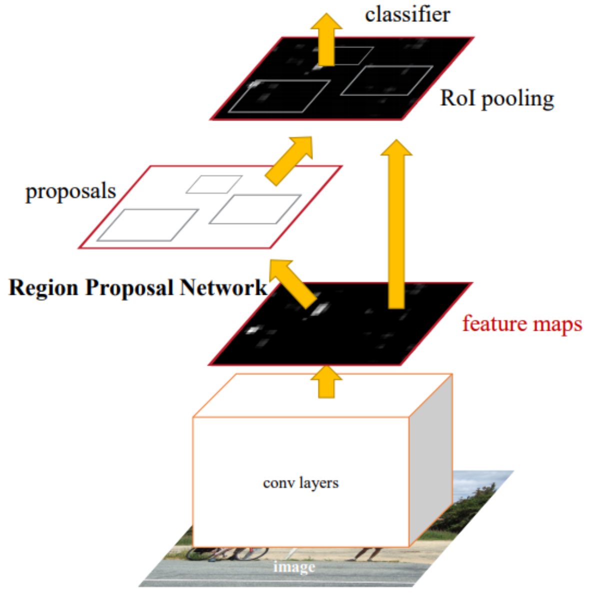


Figure 11: An illustration on the different stages of a Faster R-CNN model [6]. The RPN gets a feature map as input and uses it to suggest proposals. From the proposals and the feature maps a classifier identifies the objects and a bounding box regression improves the boxes.

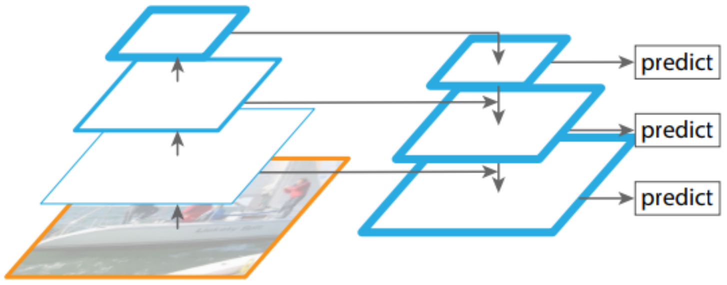


Figure 12: Shows how the features are being extracted from a picture using FPN [35]. The key point of the FPN is to combine the information from lower and higher features to make predictions.

3.5 Modern single shot detectors

The problem with the region based detectors is the speed. Engineers have been trying to reduce the work done for each RoI. But the question is if we really need a separate RPN? It is not better to find boxes and classify objects in a single step? That is what a single shot detector does. Until now they have been worse detectors than state-of-the-art two-stage detectors but, they are much faster. Next we present the two newest single shot detectors that are also comparable to the best region based detectors.

3.5.1 YOLOv3

Humans glance at an image and instantly knows where and what the objects are. This is the idea behind You Only Look Once (YOLO) [32]. YOLOv3 [34] from 2018 is an upgraded version from YOLO and YOLOv2 [33]. It uses a single neural network to the full image. By dividing the image into regions and predict bounding boxes and probabilities for each region. The architecture is very simple, it is just one big convolutional neural network. As the figure 13 shows YOLO works by dividing up the picture in an SxS grid. Each of the cells predicts B bonding boxes and a confidence score for each bonding box. The confidence score (CS) is defined as

$$CS = Pr(object)IoU(G_t, P_b) \tag{46}$$

where IoU is the intersection over union between predicted bonding box (P_b) and the ground truth box (G_t).

$$IoU(P_b, G_t) = \frac{P_b \cap G_t}{P_b \cup G_t} \tag{47}$$

Figure 14 shows the architecture of the convolutional network with 53 layers called Darknet-53 and is used for training YOLOv3.

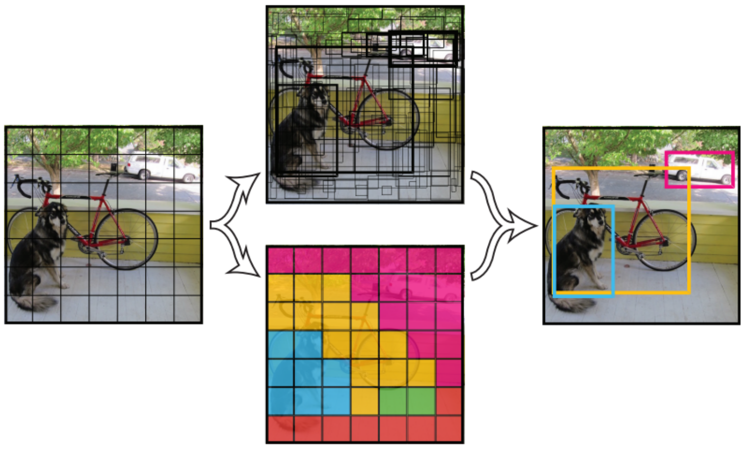
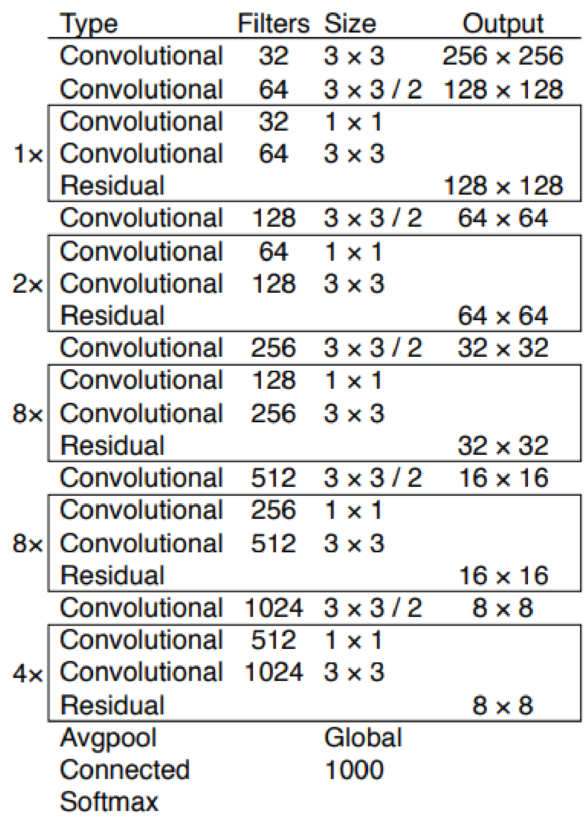


Figure 13: YOLO divides the picture into an even grid and simultaneously predict boxes, confidence in those boxes and class probabilities [32].



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
	Convolutional	64	1 × 1	
2×	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
	Convolutional	128	1 × 1	
8×	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	
8×	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
	Convolutional	512	1 × 1	
4×	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 14: The architecture of the YOLOv3 network [34]. The first part of the network is a feature network called Darknet-53. From the features an fully connected layer is used to generate boxes and calculate a score. The loss function of the scores is a standard softmax.

3.5.2 Focal Loss (RetinaNet)

The authors of RetinaNet [38] realized that one stage detectors have one big problem and that is class imbalance. The detectors analyze a lot of candidate regions (up to 10^5) and only a few contains an object. This results in slow training and an overwhelming amount of negatives compared to the positives. To tackle this problem they design a loss function called Focal Loss. Using

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{else} \end{cases} \tag{48}$$

where $y \in \{\pm 1\}$ is the ground-truth class and $p \in [0, 1]$ is the model's estimated probability for the class with label $y = 1$. From the cross entropy loss for binary classification they define the focal loss as

$$FL(p_t) = -\alpha(1 - p_t)^\gamma log(p_t \tag{49}$$

where $\gamma \in [0, 5]$ is a constant and α is a scaling factor. The authors claims $\gamma = 2$ and $\alpha = 0.25$ yields the best results, and that extending the focal loss to the multi-class case is straightforward and works well.

RetinaNet is an unified network composed of a FPN on top of a feedforward convolutional network called ResNet and two task-specific subnetworks. As figure 15 shows that one sub-network identifies the class and the other one identifies where the object is. Using the focal loss the authors claims that this simple model eliminates the accuracy gap between one-stage detectors and state-of-the art two-stage detector Faster R-CNN with FPN.

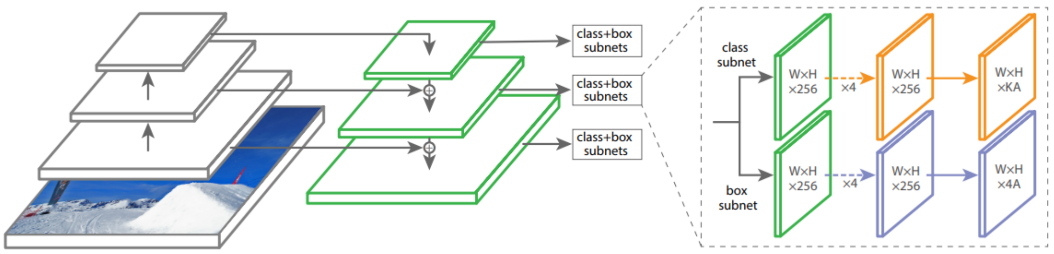
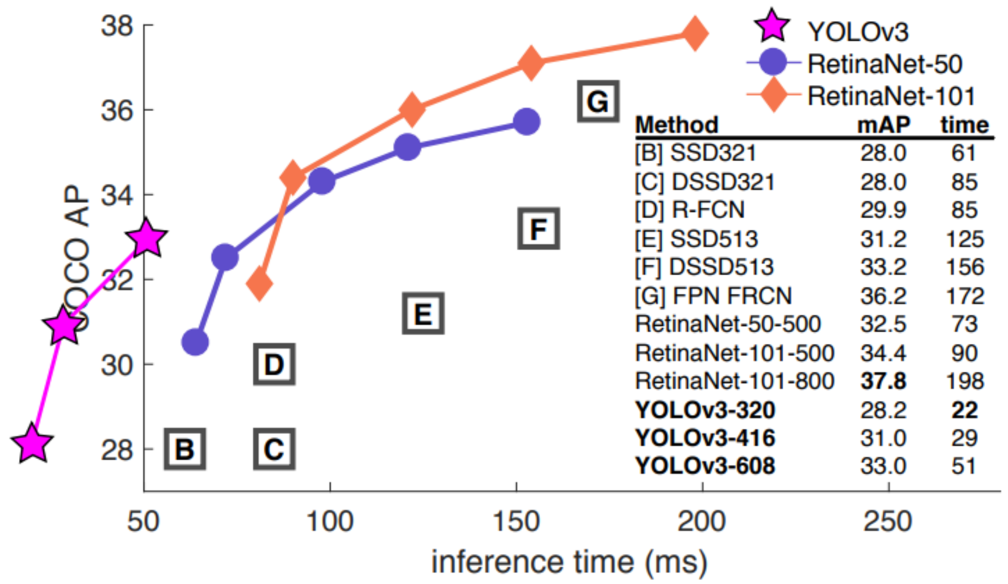


Figure 15: The architecture of RetinaNet [38]. It uses a FPN to extract features. From the features it uses an part that identifies the classes and one part that identifies the best location of the boxes.

3.6 Comparing the methods

To answer which of these methods are the best you have to know what you are going to use it for. YOLOv3 is the fastest and still scores well and can thus be used in real time detection. RetinaNet and Faster RCNN with FPN is the best if time is not a problem. To improve the detection rate even further it is important to analyze the model and find where the detection fails and improve that.



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Figure 16: Comparing the models on COCO dataset. YOLOv3 is the fastest but RetinaNet and Faster RCN with a FPN (FPN FRCN) scores the highest mean average precision (mAP) [34, 38].

4 Method

The goal of the project is to build a better way of handling the RPN part in the Faster R-CNN. The RPN is using pre-defined anchors and for each anchor we give them a score of likelihood of being on an object or not. We refine the boxes so they cover more of the object through bounding box regression. The performance on the RPN depends on which scales and aspect ratios we use. So we are going to optimize the RPN and compare the results to our proposal network.

Our proposal network works similar to YOLO which instead of using anchors it directly looks for the objects. From the feature map we train a network to do a grid search to find objects and give them a score. In this way we expect a faster and more accurate network.

To approach the multiscale problem, we split the network into 9 sections, where each section is trained to propose objects of a fixed size and aspect ratio.

4.1 Multiscale proposal network

The multiscale proposal network (MPN) is our approach to generate proposals. It takes the place of the RPN accordingly to figure 11. It starts by a 2D convolution on the feature map and each location is responsible for generating 9 boxes (one for each scale) called RoIs. For each RoI a confidence score is calculated that tells us how certain it is that the predicted bounding box actually encloses some object.

The training GT boxes are all different in aspect ratio and size. For increasing the proposal rate the GT boxes are labeled from 1-9, depending on the size and aspect ratio. The label is '1' if the size is smaller than 32^2 and the aspect ratio is smaller than 0.8, and gets the label '9' if the size is bigger than 96^2 and has a ratio of bigger than 2.0. Each RoI gets compared to the target GT box with the same label. This makes the network handling different sizes better and increases the performance.

4.1.1 Loss function

Our network loss function consist of two parts, the MPN loss and the detection loss. The MPN loss function consist of two parts, one for the location and one for the score. The loss of the location consist of a smooth L_1 loss on the RoIs and the target GT from the same label. The loss of the score is a cross entropy softmax between the target labels and the score. The target label is one if the proposed box has an IoU bigger than 0.3 with a GT box and 0 otherwise.

In the detection part of the network we train a parameter delta, δ, that adjusts the high scoring boxes to fit the object better. Again we use a smooth L_1 loss function between the generated deltas and the target deltas. We also recalculate the confidence score in the same manners as before. But now the target label is one if the adjusted box (proposal box plus delta) has an IoU bigger than 0.5 and 0 otherwise.

4.1.2 Non-maximum suppression

The MPN generates many overlapping boxes. To reduce number of boxes non-maximum suppression (NMS) is used. If two boxes are overlapping each other with an IoU bigger than 0.8 the box with the lowest classification scores gets eliminated. An example of this is shown in figure 17.

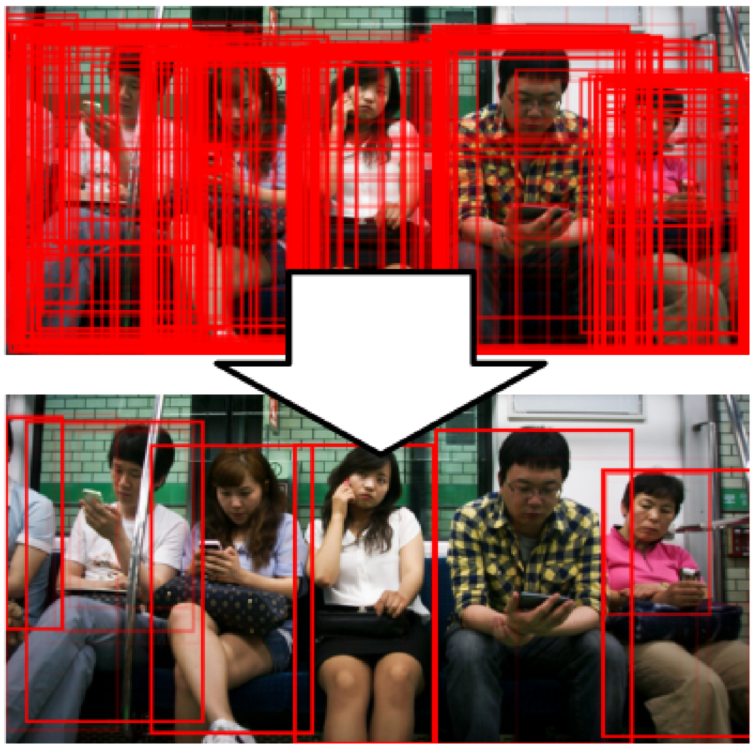


Figure 17: An illustration for how the NMS works [46].

4.2 Data

The dataset used for learning the CNNs in this project is the large-scale object detection, segmentation, and captioning dataset COCO [44]. The dataset contains photos of 91 object types (classes) and 2.5 million labeled instances in 328 000 images. In this project we only use the human class and the 2014 version of COCO. An example of a picture we feed the network for training is shown in figure 18. It is essential to have a big dataset to be able to learn the features necessary for detecting objects. Another popular dataset is PASCAL Visual Object Classes (VOC) which is a smaller scale dataset that can be used if the computer power is limited.

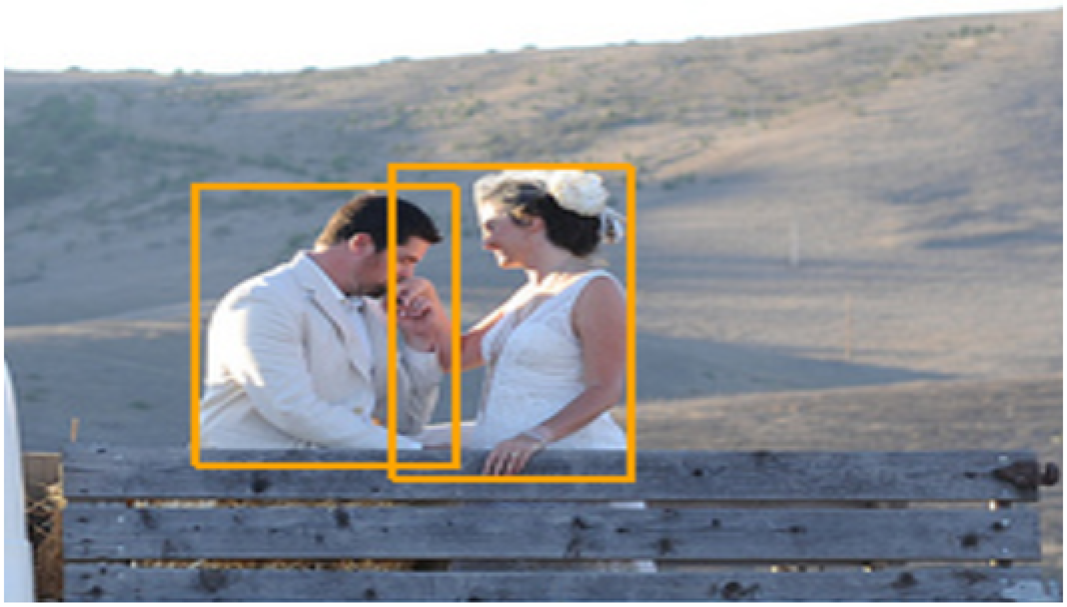


Figure 18: An example picture with GT boxes, from COCO2014

4.3 Pre-training

We initialize the parameters of the feature network (VGG16) from a pretrained imagenet model. Since our model just consist of one class it can be argued that using a pretrained model does not affect the overall results. After testing we noticed that using pre-training makes the learning faster and thus reduce the number of iterations needed for training.

4.4 Evaluation metrics

To evaluate how the different models performs, some benchmarks are being used.

- The IoU introduced in chapter 3 is a metrics that says how much a prediction covers the actual object.
- The recall is a measurement on how many objects are actually detected. It is given by:

$$Recall = \frac{TP}{TP + FP} \tag{50}$$

  Where TP is the number of True Positives and FP is the number of False Positives.

- Precision says how accurate a prediction is i.e. the probability that the prediction is correct.

$$Precision = \frac{TP}{TP + FN} \tag{51}$$

  Here FN is the number of False Negatives.

4.5 Experimental setup

The programming language used is Python together with the open source machine-learning package Tensorflow. Tensorflow requires CUDA which is a parallel computing platform developed by NVIDA to make training faster with GPU acceleration. CUDA is then supplemented with cuDNN which is a deep neural network library.

The hardware used in the learning, testing and evaluation of the models is the following:

- **GPU**: NVIDIA GTX1080Ti
- **RAM**: 32GB
- **Processor**: Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz
- **OS**: Ubuntu 16.04

4.6 Parameters

In the training we choose to use a dropout probability of 0.5, the RELU activation function and a L_2 regularizer with a penalty term of 0.05. We trained one million iterations and updated the weights using gradient descent with learning rate 0.001, decay step 500 000 and decay rate 0.1.

In the post processing of the detector output a NMS with threshold 0.3 was used. This is to reduce boxes predicting the same object.

4.6.1 RPN parameters

To optimize the parameters for the anchors in the RPN we analyzed different parameters on the dataset. Since we only use the person class on COCO2014 we analyzed how many boxes we could find for different aspect ratios and scales. The standard value from the original paper is: scales= $[64, 128, 256]$ and aspects= $[0.5, 1, 2]$. Using an overlap threshold bigger than 0.3 the model finds 61% of the GT boxes in the dataset. Using clustering on the dataset we found that using more scales is more efficient since we are not capturing the small and the big boxes as well. So what worked best is scales= $[16, 32, 64, 128, 256]$ and since the people are standing the aspects= $[1, 2]$ is working as well. This is shown in table 1.

The RPN performed better if the NMS threshold was lowered to 0.6 instead of 0.8. This makes less boxes of the same quality being generated and therefore increases the precision.

Scales	*Aspects*	*Av*	*P (%)*	*S (%)*	*M (%)*	*L (%)*	*GT (%)*
64	1.0	2500	0.80	0.00	88.0	19.0	35.7
64, 128, 256	0.5, 1.0, 2.0	22500	2.09	0.00	89.2	93.7	60.9
32, 64, 128	0.5, 1.0, 2.0	22500	1.07	52.5	99.8	74.6	75.6
16, 32, 64, 128, 256	1.0, 2.0	25000	2.00	75.0	99.7	99.0	91.2
16, 32, 64, 128, 256	1.0, 2.0, 4.0	37500	1.93	75.0	99.9	99.1	91.3
16, 32, 64, 128, 256	0.25, 0.5, 1.0, 2.0, 4.0	62500	1.47	76.7	99.9	99.2	91.9

Table 1: Shows how well the anchors captures the dataset for different aspects and scales. Av is how many anchors there are per image, P is the percentage of boxes covering an object with an IoU bigger than 0.3, S the percentage of small GT objects found ($< 32^2$), M the percentage of medium GT objects found, L the percentage of large boxes found ($> 96^2$) and GT the amount of total number of boxes found. In the dataset there are 1/3 small boxes, 1/3 medium boxes and 1/3 large boxes.

5 Results

The results of the network are split into two parts. In the first part, the output from the proposal networks is presented. In the second part, the detection from the whole network is analyzed.

We test our MPN and compare it to the RPN with two different anchor configurations. The first configuration, RPN, uses scales=$64, 128, 256$ and aspects=$0.5, 1, 2$. The second configuration, RPN extended, uses scales=$16, 32, 64, 128, 256$ and aspects=$1, 2$

5.1 Proposal network

The recall gives a percentage of how well the network captures the dataset. A recall score of one means that all the GT boxes, therefore all the persons, have at least one RoI overlapping more than the IoU threshold. Figure 19 shows that the RPN scores a little bit better. The recall at IoU=0.5 for the RPN is 0.1 higher than the MPN. This is mainly because RPN is handling more proposals and therefore find more objects. This is also shown in table 2.

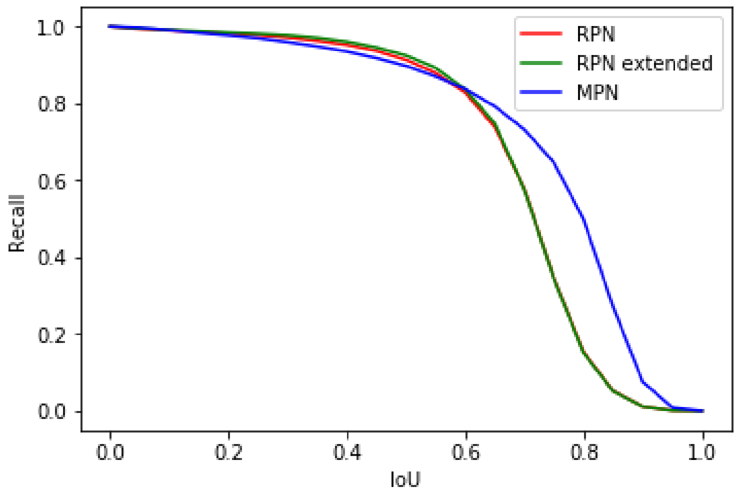


Figure 19: The recall of the proposal boxes with a confidence score higher than 0.8. On the 2014 COCO validation dataset.

Proposal network	*Proposals/image*	*Positives/image*	*Recall at 0.5*	*Anchors/image*
RPN	575	126	0.91	22500
RPN extended	550	142	0.93	25000
MPN	433	428	0.90	

Table 2: The overall results of the proposal networks. As seen out of the 22500 anchors on the RPN, after adding deltas and using non maximum suppression (NMS) 575 proposals is left. Out of these, 126 have a score higher than 0.8.

5.2 Detector

The recall of the detector (see figure 20) drops. The reason behind this is that the positives boxes should contain a person, so it has to be more precise. In figure 24 some of the false negatives is shown. In some of the boxes not even a human can see that it contains a person. So it is understandable that the networks do not cover everything. Because the dataset contains bad boxes and unlabeled persons.

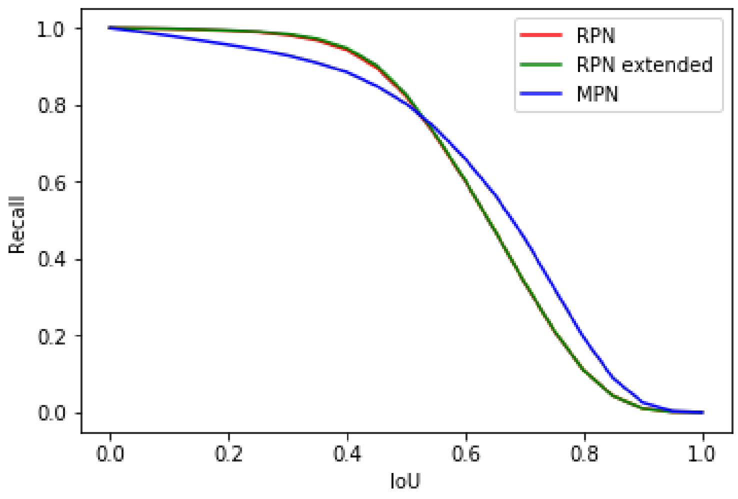


Figure 20: The recall of the detections with a confidence score higher than 0.8. On the 2014 COCO validation dataset.

The precision gives a value on how accurate the predictions are. A precision value of one means that all the positives RoI's intersect with an object. So the overall performance of an object detector is best represented with a precision-recall curve. In figure 21 the precision-recall curve is presented. Together with the average precision, which is the area under the curve. It is seen that our proposal network is more accurate than the RPN.

Proposal network	*Detection AP*	*Detection recall at 0.5*	*Training time*	*Running time*
RPN	0.7992	0.8230	4 days, 17h	30ms
RPN extended	0.7879	0.8275	4 days, 13h	33ms
MPN	0.9022	0.8032	4 days, 8h	25ms

Table 3: The overall results of the detectors. The RPN seems to give more FP resulting in a worse AP score than the MPN. The time is measured on a NVIDIA GTX1080ti. The running time is how long time it takes to generate the detections per image.

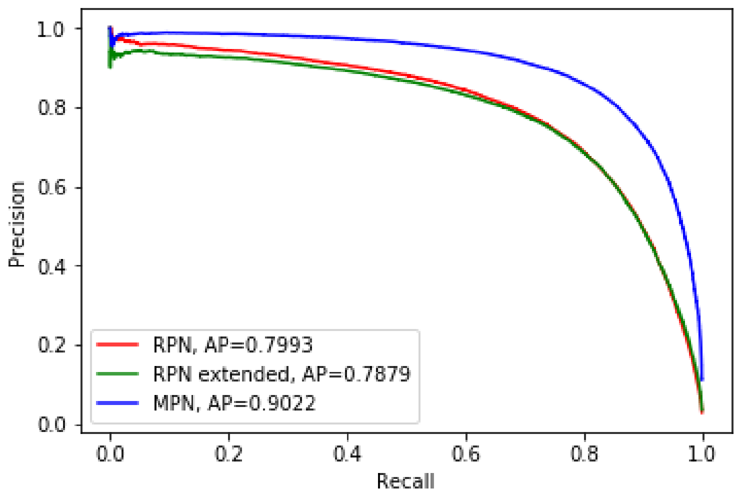


Figure 21: Precision recall of the detectors

6 Discussion

The main issue of this study has been to implement Faster R-CNN and exchange the RPN to a proposal network that directly tries to find boxes from the feature maps. As benchmark we tested Faster R-CNN with RPN and compared it to our MPN. As seen in the results, the RPN has a higher recall (see table 2). This comes with a cost of also having a lot of false positives, making it less precise than our model. Another reason for MPN performing better is because we are overtuning the RPNs which results in less negatives and therefore eliminates many of the high scoring negatives that were needed for properly training the detector. In figure 22 some chosen examples are shown. Our method detects boxes of different scales and aspects. Each output box has a softmax score in [0,1]. A score threshold of 0.8 is used to display these images. For this study, the MPN is faster and is more precise. To be able to see if MPN can truly compare to the state-of-the-art object detector FPN and ResNet needs to be implemented and trained on the full COCO dataset. This needs more processing time to converge. With our current setup that would require either more GPUs or training for a much longer time. Right now the training time for MPN running one million iterations on a NVIDIA GTX1080i is 4 days and 8h.

Figure 23 presents the top scoring false positives from our network. As seen there, some of them are human-like (statutes and dolls) or are miss-labeled humans. There also exist objects that have nothing in common with humans (crows and airplanes). The reason that there are objects identified as humans is because the feature network consists of many layers, but early layer features may have been forgotten. A solution to this would be to implement a way for the network to re-use earlier feature maps. One way is to use the FPN (see 3.4.2). Another way is to store the feature maps and training the network to decide which one is best to use.

Figure 24 contains some of the false negatives. As seen, the blurry and dark boxes are almost impossible for a human to see a person in and some of them just contain small parts of a human (fingers, knees and feet). So it is understandable that those boxes are wrongly classified since these pictures are rare. If we add more unclear pictures to balance the training set we can improve the results. If tested on clearer, more normal pictures the detector scores even better than what the results show.

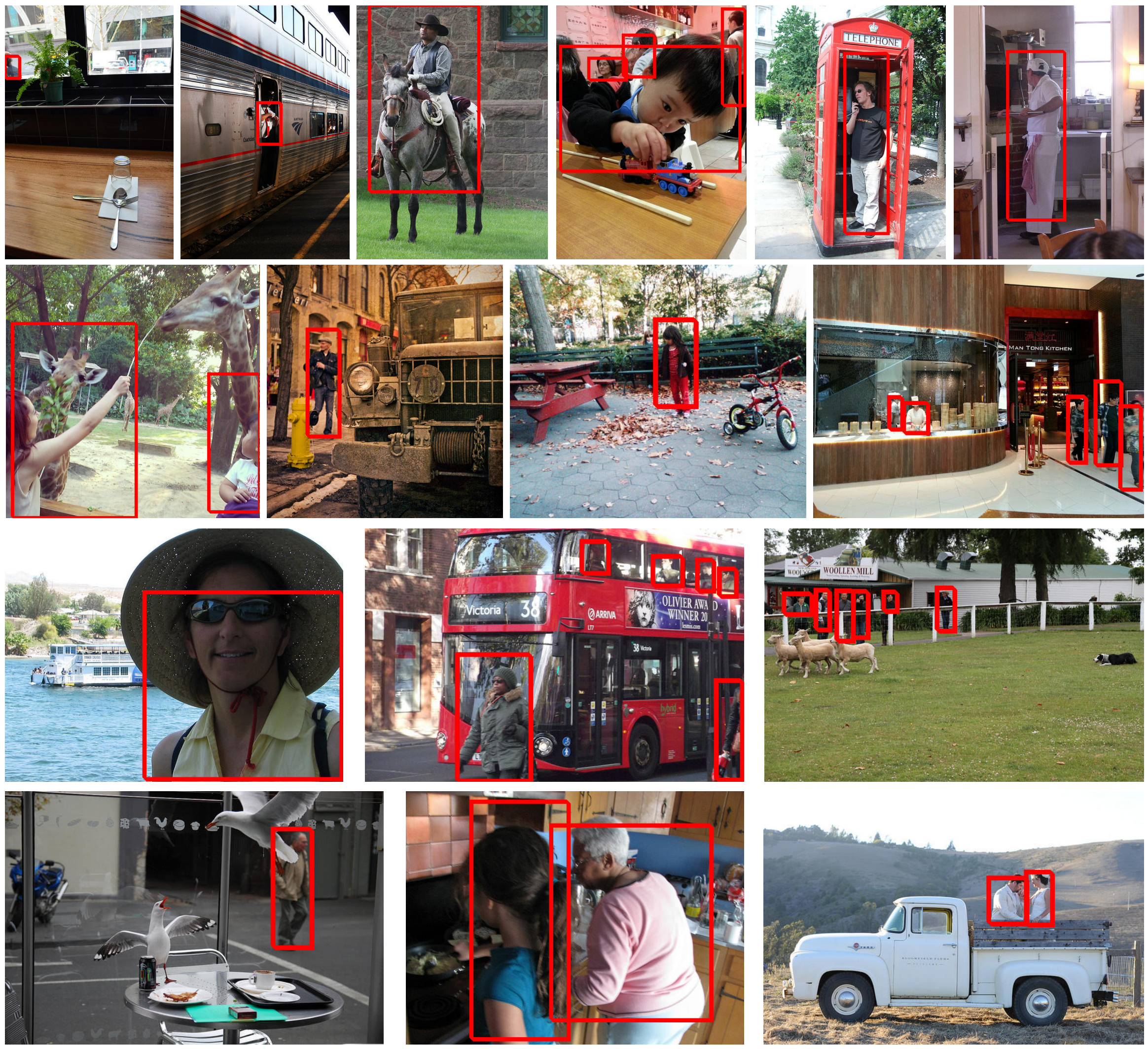


Figure 22: Selected examples of object detection results on the COCO 2014 validation set using the MPN in the Faster R-CNN system.

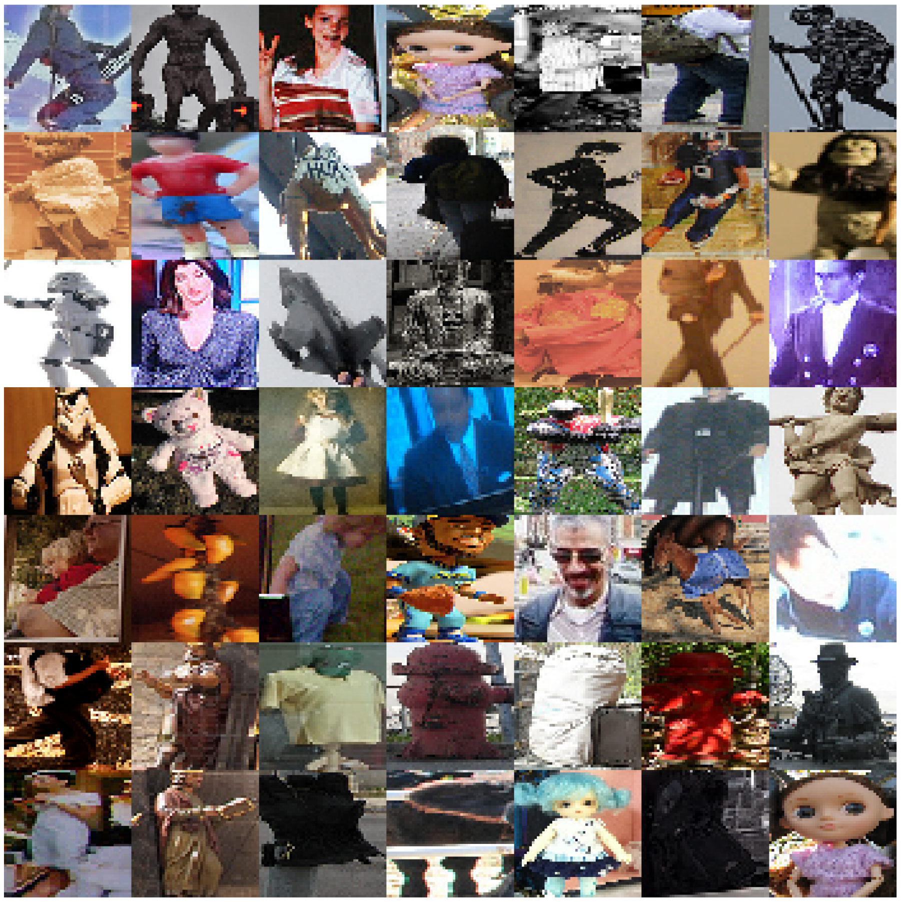


Figure 23: The 49 highest scoring false positives from the detector using our MPN. Some of them is wrong label humans, human like dolls and statutes. The most surprising is the aircraft and the bird.

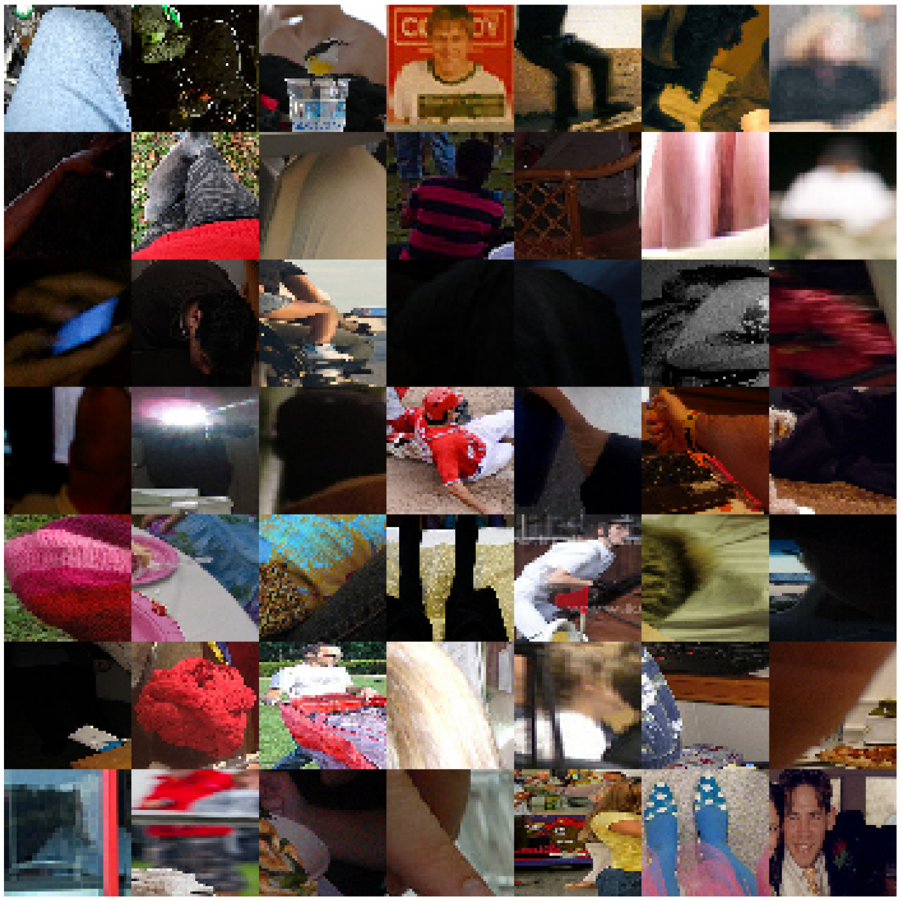


Figure 24: The best overlapping false negatives from the detector using our MPN. Dark pictures and pictures containing human parts leads to miss classifications.

7 Conclusions

In this study of the proposal networks in object detection we have looked at CNNs and how it has revolutionized the field. Instead of hard-coding the program to tell where the object detector should look, CNNs lets the program learn to find the objects by itself and create its own filters. This method makes object detection faster, more precise and easier to code.

Previous models that generate proposals used hand engineered machine learning models to make these region proposals. The latest big improvement is to use the RPN as a proposal generator. The RPN is better since it lets a neural network create faster and more precise filters to suggest regions. One of the limitations of the RPN is that it uses pre-defined anchors. To improve the proposal generation even further we suggest a model that lets the CNN directly look for objects, give it a box and give it a score. In addition we split the network into 9 separate parts to do suggestions for different sizes and shapes of objects. We call it the Multiscale Proposal Network (MPN). This network has an average precision of 0.1 higher than the RPN, with a drawback that the recall is slightly worse than the RPN. The network is also delivering the detections from the pictures 10% faster. Which means our network is both faster and more precise than the RPN resulting in a better choice to use in the field of object detection.

7.1 Future studies

In this study of the proposal network we only used one object class. What would be interesting to see is how our network performs with several classes. And if our method is compatible with the state of the art models.

Each part of the object detector can be researched on. For example letting the network choose different feature maps, letting the network do the NMS, testing different loss functions and parameters can all be studied for big extent. It is much left to explore and many ideas left to test in this growing field. Every year the computer power gets faster and python packages such as tensorflow and pytorch improves and become more user friendly, resulting in more people being able to start develop in the field.

8 References

[1] Viola, P., & Jones, M. (2001). Robust real-time face detection (p. 747). Presented at the Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on.

[2] Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection (pp. 886–893). Presented at the CVPR (1).

[3] E.H. Andelson and C.H. Anderson and J.R. Bergen and P.J. Burt and J.M. Ogden. "Pyramid methods in image processing". 1984.

[4] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation (pp. 580–587). Presented at the CVPR '14: Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, IEEE Computer Society.

[5] Girshick, R. (2015). Fast R-CNN. 2015 IEEE International Conference on Computer Vision (ICCV), 1440–1448. http://doi.org/10.1109/ICCV.2015.169

[6] Ren, S., He, K., Girshick, R., & Sun, J. (2015, June). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv.org.

[7] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., et al. (2017). Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. Cvpr, 3296–3297. http://doi.org/10.1109/CVPR.2017.351

[8] Stewart, R., & Andriluka, M. (2015, June 16). End-to-end people detection in crowded scenes.

[9] Uijlings, J.R.R., van de Sande, K.E.A., Gevers, T. et al. Int J Comput Vis (2013) Selective Search for Object Recognition

[10] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari (2012) Measuring the Objectness of Image Windows

[11] Zitnick C.L., Dollár P. (2014) Edge Boxes: Locating Object Proposals from Edges. In: Fleet D., Pajdla T., Schiele B., Tuytelaars T. (eds) Computer Vision - ECCV 2014. ECCV 2014. Lecture Notes in Computer Science, vol 8693. Springer, Cham

[12] R. Kohavi and F. Provost, "Glossary of terms" Machine Learning, vol. 30, no. 2-3, pp. 271-274, 1998.

[13] Habibi Aghdam, Hamed and Jahani Heravi, Elnaz (2017) Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification

[14] Pattanayak, Santanu (2017) Pro Deep Learning with TensorFlow: A Mathematical Approach to Advanced Artificial Intelligence in Python

[15] Ketkar, Nikhil (2017) Deep Learning with Python

[16] Cunningham P., Cord M., Delany S.J. (2008) Supervised Learning. In: Cord M., Cunningham P. (eds) Machine Learning Techniques for Multimedia. Cognitive Technologies. Springer, Berlin, Heidelberg

[17] Ghahramani Z. (2004) Unsupervised Learning. In: Bousquet O., von Luxburg U., Rätsch G. (eds) Advanced Lectures on Machine Learning. Lecture Notes in Computer Science, vol 3176. Springer, Berlin, Heidelberg

[18] Richard S. Sutton and Andrew G. Barto (2007) Reinforcement Learning: An Introduction Second Edition, in progress MIT Press, Cambridge, MA

[19] David Kriesel (2007) A Brief Introduction to Neural Networks

[20] Rosenblatt, Frank (1957), The Perceptron–a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.

[21] Marvin Minsky and Seymour Papert, 1972 (2nd edition with corrections, first edition 1969) Perceptrons: An Introduction to Computational Geometry, The MIT Press, Cambridge MA, ISBN 0-262-63022-2.

[22] Ramchoun, H., M. A. J. Idrissi, Y. Ghanou, and M. Ettaouil (2016) Multilayer Perceptron: Architecture Optimization and Training

[23] Cybenko, G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2 (4), 303-314

[24] T. Huang (1996) Computer Vision: Evolution and Promise

[25] Q.V. Le (2015) A Tutorial on Deep Learning, Lecture Notes

[26] JG Makin (2006) Backpropagation

[27] David Stutz (August 30, 2014) Seminar Report, Understanding Convolutional Neural Networks

[28] Zhifei Zhang (2016) Derivation of Backpropagation in Convolutional Neural Network (CNN), University of Tennessee, Knoxvill, TN, October 18, 2016

[29] Samer Hijazi, Rishi Kumar, and Chris Rowen, IP Group, Cadence (2015) Using Convolutional Neural Networks for Image Recognition

[30] D. Lowe. (2004) Distinctive image features from scale-invariant keypoints. IJCV

[31] Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick (2017) Mask R-CNN. arXiv:1703.06870

[32] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi (2015) You Only Look Once: Unified, Real-Time Object Detection. arXiv:1506.02640

[33] Joseph Redmon, Ali Farhadi (2016) YOLO9000: Better, Faster, Stronger. arXiv:1612.08242

[34] Joseph Redmon, Ali Farhadi (2018) YOLOv3: An Incremental Improvement. University of Washington. arXiv:1804.02767

[35] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, Serge Belongie (2017) Feature Pyramid Networks for Object Detection. arXiv:1612.03144

[36] Jonathan Hui, a deeplearning blog explaining and comparing all the models for object detection. https://medium.com/@jonathan_hui

[37] Jifeng Dai, Yi Li, Kaiming He, Jian Sun (2016) R-FCN: Object Detection via Region-based Fully Convolutional Networks. arXiv:1605.06409

[38] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár (2017) Focal Loss for Dense Object Detection. arXiv:1708.02002

[39] P. F. Felzenszwalb and D. P. Huttenlocher. (2004) Efficient GraphBased Image Segmentation. IJCV, 59:167–181,

[40] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. How good are detection proposals, really?. Proceedings of the British Machine Vision Conference. BMVA Press, September 2014.

[41] M. D. Zeiler and R. Fergus (2014) Visualizing and understanding convolutional neural networks, in European Conference on Computer Vision (ECCV)

[42] K. Simonyan and A. Zisserman (2015) Very deep convolutional networks for large-scale image recognition, in International Conference on Learning Representations (ICLR)

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2015) Deep Residual Learning for Image Recognition. arXiv:1512.03385

[44] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollár (2014) Microsoft COCO: Common Objects in Context arXiv:1405.0312

[45] Everingham, M. and Van Gool, L. and Williams, C. K. I. and Winn, J. and Zisserman, A., The PASCAL Visual Object Classes Challenge 2007 http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html

[46] Jan Hosang, Rodrigo Benenson and Bernt Schiele. Learning non-maximum suppression, arXiv:1705.02950

9 Appendix

9.1 Additional networks

9.1.1 Zeiler and Fergus network

The Zeiler and Fergus (ZF) feature network from 2013 [41] is a seven layer network.

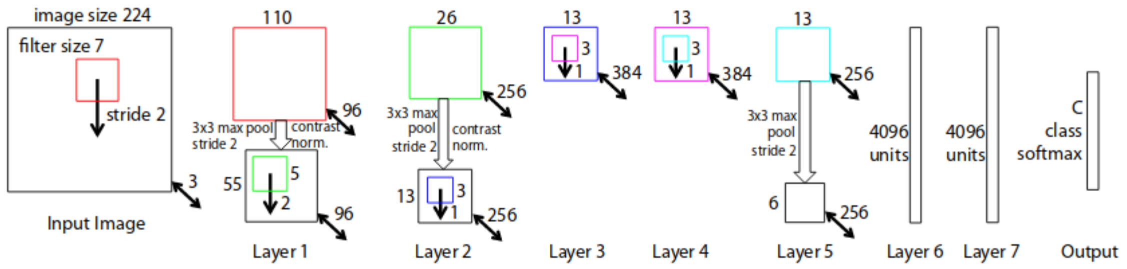


Figure 25: Shows the ZF network structure [41].

9.1.2 Mask R-CNN

Mask R-CNN (since 2017) is an extension of Faster R-CNN and builds on the following question: Is it possible from Faster R-CNN to carry out pixel level segmentation? This is what mask RCNN does. By adding a branch to Faster R-CNN that outputs a binary mask around the object [31].

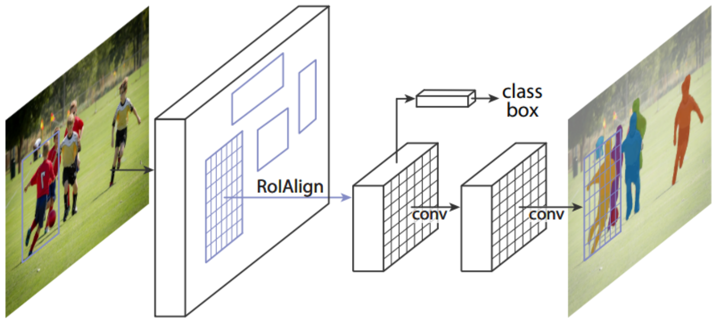


Figure 26: Shows the different stages of Mask R-CNN [31].

9.1.3 Region-based Fully Convolutional Networks (R-FCN)

R-FCN uses position sensitive score maps to address translation-invariance and translation-variance resulting in a much faster network than Faster R-CNN that uses a costly per-region network hundreds of times [37]. Figure 27 shows how the R-FCN classifies an object.

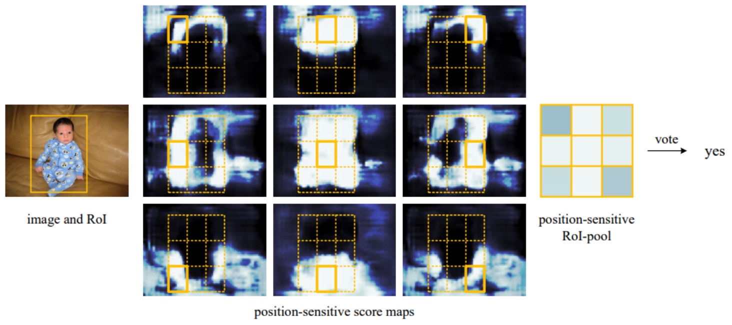


Figure 27: Shows how the features are being extracted from a picture using R-FCN [37].

9.2 Additional equations

$$L_i = -log(e^{y_i} / \sum_j e^{d_j}) \tag{52}$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i \tag{53}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{54}$$

$$Recall = \frac{TP}{TP + FN} \tag{55}$$

$$Precision = \frac{TP}{TP + FP} \tag{56}$$

$$IoU(A, B) = \frac{A \cap B}{A \cup B} \tag{57}$$

Mean Average Precision (mAP)

$$mAP = \frac{1}{|classes|} \sum_{c \in classes} \frac{TP(c)}{TP(c) + FP(c)} \tag{58}$$

Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{59}$$

9.3 Additional pictures

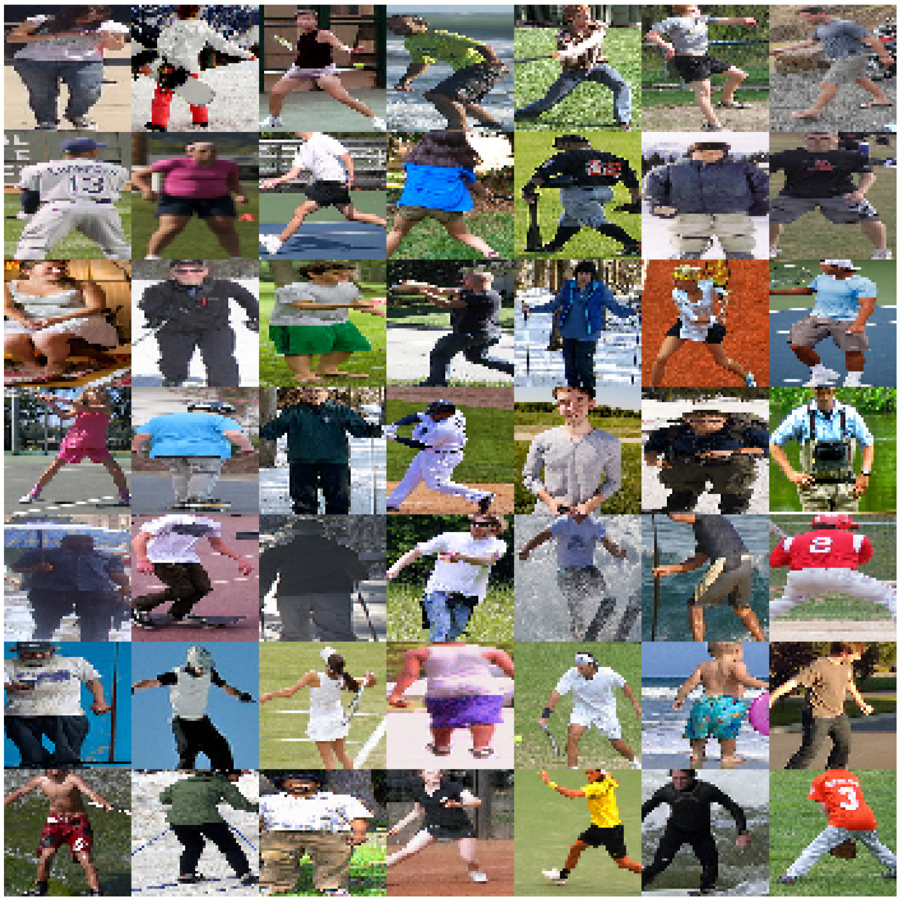


Figure 28: The best overlapping true positives from the detector using our MPN.