technische universität
dortmund

Object Detection and Segmentation using Region-based Deep Learning Architectures

Master Thesis

Muhammad Waleed Zafar

December 14, 2018

Supervisors:
Prof. Dr.-Ing. Gernot A. Fink
Fernando Moya Rueda, M.Sc.

Fakultät für Informatik - LS XII
Technische Universität Dortmund
http://patrec.cs.uni-dortmund.de

CONTENTS

1 INTRODUCTION

Computer systems need to understand a visual scene in order to perform various tasks. Humans, equipped with senses, can effectively and subconsciously interpret visual information to make informed decisions. However, for computer systems this means crossing the semantic gap between the pixel level information and human's perception of visual information. Computer vision bridges this gap.

Extracting higher level semantic information from images is one the most fundamental and challenging problems in computer vision. Image recognition, object detection, and instance segmentation are the broad categorizations of computer vision. Recognition also referred to as classification problem, is the process of identifying and validating objects in an image and classifying them into certain classes. Detection, however, is the process of identifying and labeling multiple relevant objects in a single image or video frame, together with a rough estimation of the location and the size of that object. Instance segmentation is the combination of classical object detection tasks with semantic segmentation, where the goal is to classify each pixel into a fixed set of categories without differentiating object instances.

Prior to the application of deep neural networks for object detection and segmentation tasks, classic object detection frameworks were mainly based on extracting feature descriptors such as Histograms of Orientated Gradients (HOG) [DT05], Viola-Jones algorithm [VJS05], and Scale-Invariant Feature Transform (SIFT) [Low99]. These classic object detectors relied heavily on hand-coded features.

With significant advances in deep learning methods, fast and accurate object detection systems are rising in demand. Over the past few years, considerable efforts has been invested in the augmentation of Convolutional Neural Networks (CNNs) to propose regions which encapsulate objects of interest in a single image. Such networks are called region-based deep neural networks. The work presented in this thesis is concerned with the analysis of performance of state-of-the-art region-based deep neural networks. These networks are comprised of three basic modules: region proposal generation, classification, and bounding-box regression. Initial region-based models for object detection such as Region-based Convolutional Neural Network (R-CNN) [GDDM14] and Fast Region-based Convolutional Neural Network (Fast R-CNN) [Gir15] deployed external algorithms for the task of region proposal generation. Selective Search [UVDSGS13], Category-Independent Object Proposals [EH10], Constrained Parametric Min-Cuts (CPMC) [CS11], Multi-Scale Combinatorial Grouping [APTB+14], Edge Box Detection [ZD14] etc. are few examples of such external algorithms. Once, these algorithms propose such regions, multi-layer Convolutional Neural Networks (CNNs) are employed to compute highly discriminative features in each region, and then classifiers are trained to label the relevant objects in those regions.

Even though the initial region-based object detectors were intuitive, they had a bottleneck in terms of running time. With the increase of data volume and computational resources, faster and robust detection techniques became a requirement. Hence, Faster Region-based Convolutional Neural Network (Faster R-CNN) [RHGS15] and Mask Region-based Convolutional Neural Network (Mask R-CNN) [HGDG17] were developed. These models do not require any external algorithm for the region proposal generation task. Instead, they utilize convolutional layers of the network for region proposal generation, reducing its marginal cost and making the detection even faster.

The objective of this thesis is to analyze the performance of state-of-the-art region-based deep neural architectures for object detection and instance segmentation using different backbone architectures and other notable modifications. The work presented in this thesis is also concerned with finding a speed-accuracy trade-off for said object detection and segmentation frameworks in order to detect and segment objects with higher detection rate and accuracy.

1.1 STRUCTURE

Chapter 1 provides a brief introduction to the theoretical background of machine learning and explains the fundamental concepts associated with deep learning for object detection and segmentation. In chapter 2, a brief history of classical object detection methods is presented along with the modern history of object detection and segmentation. The third chapter explains the related work that combines Convolutional Neural Networks (CNNs) with region proposal generators. The methods adopted in the thesis are presented in the fourth chapter. Chapter 5 is dedicated to the experimental evaluation along with the analysis of results. Finally, chapter 6 provides the conclusion and possible future work.

2 FUNDAMENTALS

In this work, region-based deep learning architectures are explored. The theoretical background necessary for understanding the fundamental concepts and terminologies associated with deep learning for object detection and instance segmentation will be provided in this chapter. First, the basics of machine learning will be discussed in section 2.1 followed by deep learning in section 2.2. section 2.3 discusses the history, development and operation of Convolutional Neural Networks (CNNs). Finally, the fundamentals of region-based deep neural networks will be discussed in section 2.8.

2.1 BASICS OF MACHINE LEARNING

The term "machine learning" was first used by Arthur Samuel in 1959. He defined it as "a field of study that gives the ability to the computer to learn without being explicitly programmed". By this definition, machine learning can be thought of as a plethora of algorithms that 'learn' to recognize patterns in a set of training data without being explicitly programmed to do so. Generally, the learning can be of two types: **supervised** and **unsupervised**. The key difference between both types is that supervised learning requires the labeled data to train the algorithms, while unsupervised learning does not require the data with historical labels. Goodfellow et al. [GBCB16] define unsupervised learning as a technique of implicitly or explicitly learning the probability distribution $p(x)$ by observing several examples of a random vector x.

ARTIFICIAL NEURAL NETWORKS (ANNS)

Artificial Neural Networks (ANNs) are the class of machine learning algorithms, which are highly inspired by the operations of the biological nervous system. An ANN is comprised of a large number of small inter-connected units (known as neurons). An artificial neuron (Figure 2.1.1), first introduced by Warren McCulloch and Walter Pitts in 1943 [MP43], is the basic unit of an ANN. The neuron is activated if the sum of the binary inputs is greater than threshold θ. For threshold $\theta > 0$, the activation function f is defined as

$$f(x_1, ..., x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i \geqslant \theta \ . \\ 0 & \text{otherwise} \end{cases} \tag{2.1.1}$$

These neurons are generally segmented into multiple layers. An input layer takes information as a multidimensional vector and distributes it to a series of hidden layers. The information is then transformed and disseminated (parameterized by the trainable parameters i.e. weights and biases) across the hidden layers. Finally, the last layer outputs some abstract features of the input information. An ANN optimizes these trainable features through a differentiable loss function, with the aim of improving the likeness of the target labels and the predictions generated by the ANN.

The basic structure of an ANN is shown in Figure 2.1.2.

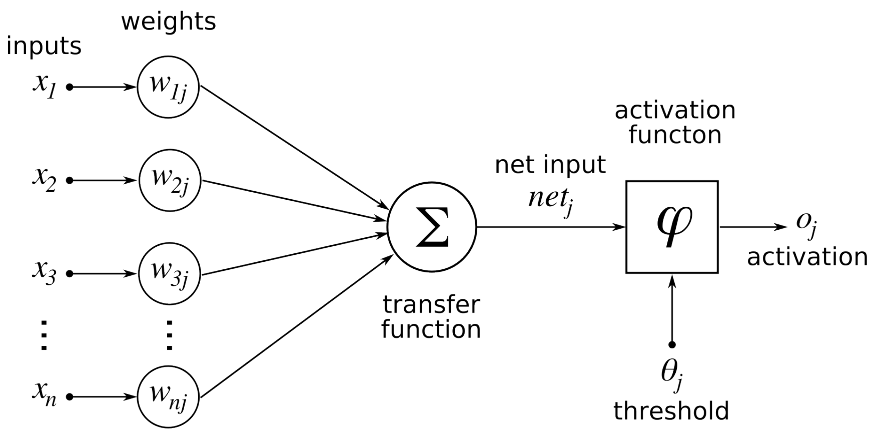


Figure 2.1.1: A McCulloch-Pitts neuron [Cas]

2.2 DEEP LEARNING

Deep learning can be considered as a part of a broader family of machine learning, which allows the computer systems to transform simpler concepts into more abstract and complex concepts. [GBCB16]. Deep learning models, more colloquially known as deep neural networks, deploy multiple hidden layers to exploit the unknown structure in the input distribution to discover composite representations.

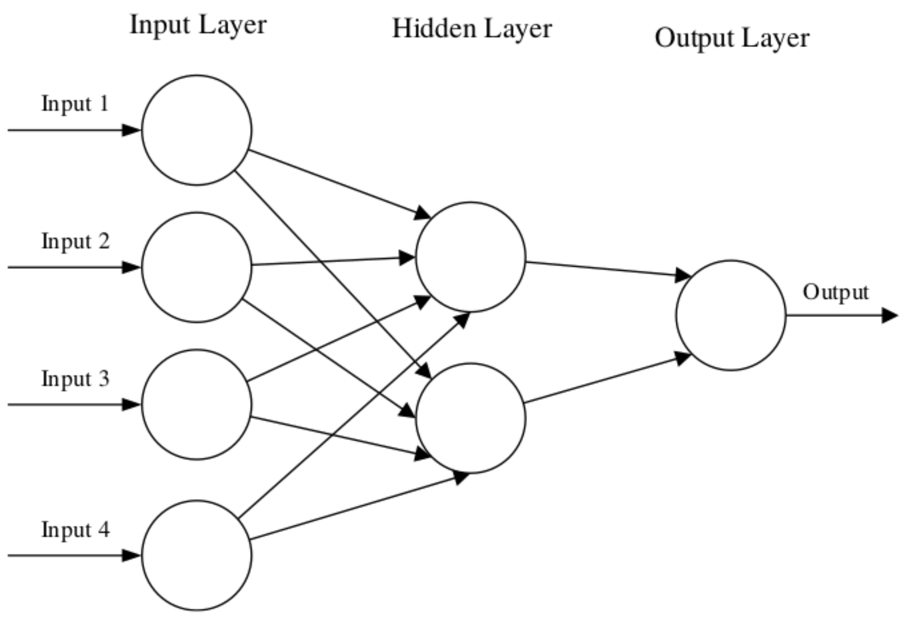


Figure 2.1.2: A three-layered Artificial Neural Network (ANN), comprised of an input layer, a hidden layer, and an output layer [ON15]

Multi-layer deep neural networks have been around since the 1980s, but in recent years, the advancements in powerful computation and availability of larger datasets have increased the popularity of deep neural networks. With the arrival of graphics processing units, the training of deep neural networks with greater efficiency has become a reality. In contrast to the classical pattern recognition systems, the deep learning minimizes the need of hand-crafted machine learning solutions by a large amount.

2.3 CONVOLUTIONAL NEURAL NETWORKS (CNNS)

Convolutional Neural Networks (CNNs) are the most impressive form of Artificial Neural Network (ANNs). CNNs are the key component of deep learning, which are primarily used to solve the difficult image-driven pattern recognition tasks [ON15].

The basic idea of CNN was inspired by a feature of the animal visual cortex called receptive field [Fuk88]. Receptive fields act as detectors that are sensitive to certain types of stimulus, for example, edges. In image processing, the same kind of visible effects can be produced by convolution filtering [Fuk88].

A typical CNN is constructed by the repetition of three basic types of layers, namely convolutional layers, pooling layers, and fully connected layers. A deep neural network stacks a large number of these layers to perform pattern recognition and detection related tasks. A simplified CNN architecture for the MNIST [Den12] handwritten digit image classification is illustrated in Figure 2.3.1.

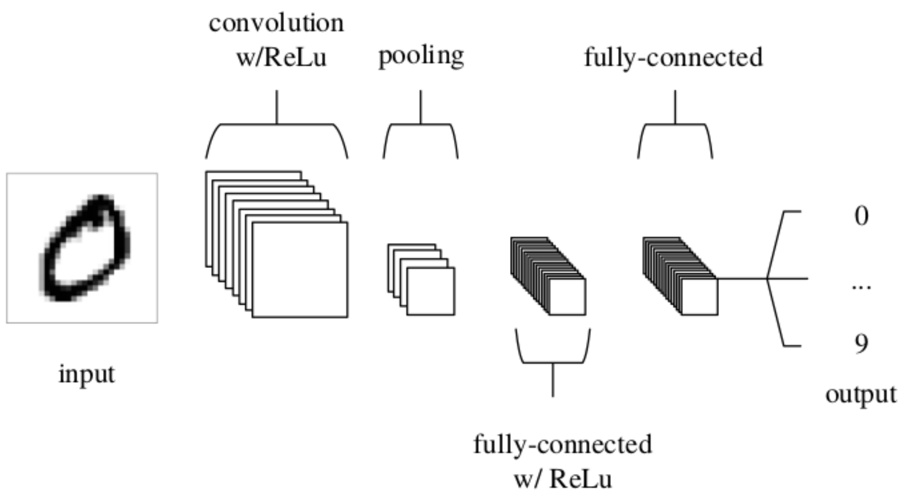


Figure 2.3.1: An CNN architecture, comprised of five layers [ON15]

Even though, the CNNs have attracted significant amount of attention in recent years, their history extends back to the 1980s. The first supervised learning algorithm that used gradient descent (subsection 2.5.2) [RHW86] was created by Rumelhart et al. in 1986, which was later used by LeCun et al. for the handwritten digits recognition [LBD+89]. However, this algorithm suffered from multiple performance issues. One of the issues was concerned with poor in-built invariance as it failed to handle the variability in handwriting samples when subjected to translations or distortions. This led to the creation of a new model with stronger shift invariance which responds to the hierarchies of local features. This new network was called the Convolutional Neural Network (CNN)[LBBH98].

CNNs were abandoned rightly after due to the lack of computing power and replaced by Support Vector Machines (SVMs). However, the development of powerful GPUs in last decade has reawakened the hope in the perceptive abilities of CNNs.

Now, the CNNs are being used as the default approach to solve many computer vision problems [GDDM14].

2.4 OPERATIONS OF A CNN

A CNN is comprised of multiple layers of operations, each with their own function. The CNN takes an image as input and feeds it to the first layer. Feature information is propagated through the hidden layers. For each layer, the activation functions perform an element-wise activation to the output produced by the previous layer. The output of the final layer is then compared against the targeted output. This process is known as *forward pass*.

2.4.1 *Convolutional Layer*

The convolutional layer plays an important role in how CNNs operate. This layer's parameters focus on small $n \times n$ windows called kernels, filter matrices or simply filters. Given input data, the convolutional layer convolves each filter or kernel across the spatial dimensionality of the input to produce a 2D feature map (or an activation map after computing the activation function). These activation maps can be visualized in Figure 2.4.1

Classically, the convolution operation is defined as an integral that expresses the amount of overlap of one function g as it is shifted over another function f, given as

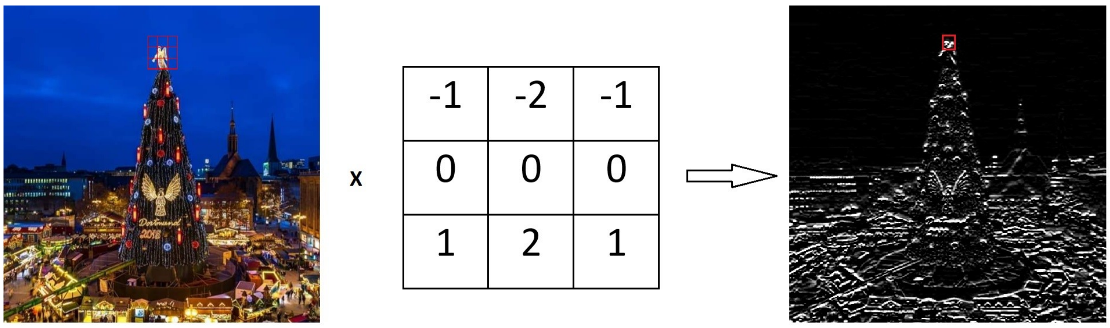
$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau, \tag{2.4.1}$$

where f and g are the arbitrary continuous functions in domain t. In case of an image, the convolution operation is defined as a discrete function. The discrete convolution operation between an image f and a filter matrix g is defined as

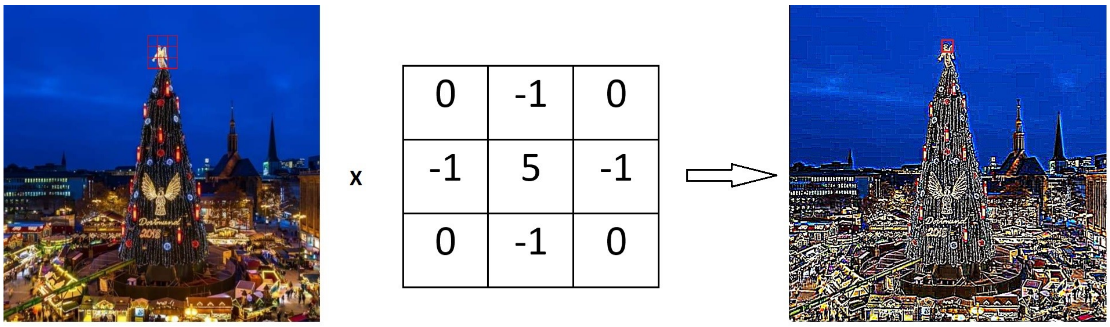
$$f[n] * g[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]. \tag{2.4.2}$$

The feature map is the output matrix h (Equation 2.4.2) obtained by aligning the filter successively with every sub-image of f (with same dimensions as g) centered on coordinates x, y.

The convolutional layer of a neural network is a combination of multiple convolutional filters, whose matrix values are treated as neuron parameters. Successive repetition of the convolutional layers along with some other types of layers (such as pooling layer) results in a Convolutional Neural Network (CNN) [ON15].



(a) Horizontal edges detection



(b) Image sharpening

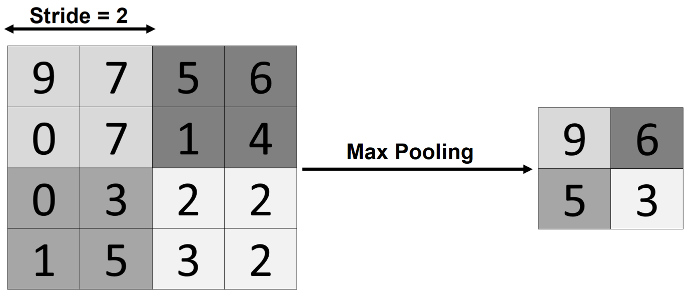
Figure 2.4.1: Feature maps after convolution filtering

.

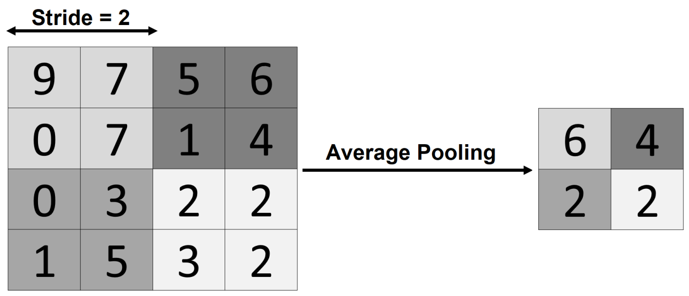
2.4.2 *Pooling Layer*

To add spatial invariance and to make the network more manageable for image processing, it is useful to reduce the activation map size. Hence, the pooling operation of a neural network is utilized to gradually reduce dimensionality of the activation map. Pooling layer, when applied, reduces the number of parameters and the computational complexity of the model [ON15].

Since the deeper layers of a network require less information about exact spatial locations of features and more filter matrices to recognize multiple high-level patterns[GBCB16], it is beneficial to reduce the dimensions of the data volume to optimize computation time. Introduction of a pooling layer after a convolutional layer is one way to achieve the aforementioned goal. The most commonly used method of pooling is max-pooling. Max-pooling outputs the maximum value within a rectangular neighborhood of the activation map[GBCB16]. Another type of pooling is average pooling that gives the average value of the rectangular neighborhood of the activation map. The difference between both types with stride 2 is illustrated in Figure 2.4.2.



(a) The maximum value from a 2×2 local neighbourhood is taken as the output of a max-pooling layer.



(b) The average value from a 2×2 local neighbourhood is taken as the output of an avg-pooling layer.

Figure 2.4.2: Max-pooling vs average pooling with stride 2

2.4.3 *Fully-Connected Layer*

Fully-connected layers are the final layers of a CNN that contain a number of neurons, all connected to each other between two adjacent layers[ON15]. A fully-connected layer generally acts as a classifier at the end of the CNN and is analogous to a traditional ANN, as shown in Figure 2.1.2.

2.4.4 *Activation Function*

The activation function ϕ is applied to the output of a convolution layer to limit the output of each neuron and to introduce non-linearities to the linear activations generated by a convolutional layer. Few examples of some of the popular activation functions used in neural networks for classification and detection tasks are given as follows:

2.4.4.1 *Rectified Linear Unit (ReLU)*

Rectified Linear Unit (ReLU) is one of the most commonly used activation function in deep learning, which is computed as

$$f(x) = max(0, x). \tag{2.4.3}$$

ReLU activation functions are quite popular for the creation of a non-linear network as it is easier to differentiate for back-propagation. Even though the ReLU function is not differentiable at zero unlike sigmoidal activation function, which has smooth derivatives, it still converges faster than sigmoid and hyperbolic tangent function.

2.4.4.2 *Leaky Rectified Linear Unit (Leaky ReLU)*

Leaky ReLU activation function is similar to the ReLU function with one key difference: the leaky ReLU allows a small positive gradient when the unit is inactive. A leaky ReLU is computed as

$$f(x) = max(x, ax). \tag{2.4.4}$$

2.4.4.3 *Sigmoid Function*

The Sigmoid function was the most widely-used activation function before ReLU, which takes real values as input and outputs the values in the range [0, 1]. The sigmoid function is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.4.5}$$

The larger negative input values tend to be closer to 0 while the larger positive inputs are closer to 1. Sigmoid function is rarely used now as it suffers from the gradient saturation problems and slower computation. Also, it is not zero centered.

2.4.4.4 *Hyperbolic Tangent Function*

Hyperbolic Tangent function or tanh is similar to sigmoid function with a difference of output, ranging in the interval $[-1, 1]$ instead of $[0, 1]$. It is defined as

$$\tanh(x) = \frac{2}{1 + e^{-2x}}. \tag{2.4.6}$$

2.4.4.5 *Heaviside Step Function*

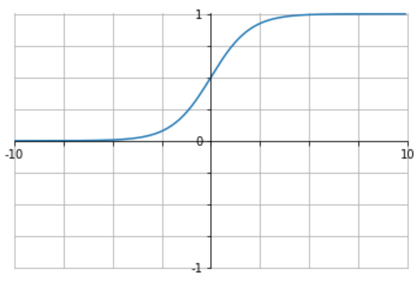
Heaviside step function is a discontinuous activation function which gives the output of 1 for $x > 0$ and 0 otherwise. This activation function, however, can not be used in deep neural networks. Since deep neural networks use gradient descent with back-propagation for the training and the back-propagation requires a differentiable activation function, Heaviside Step function, being non-differentiable at $x = 0$, can not be used.

2.5 TRAINING OF A CNN

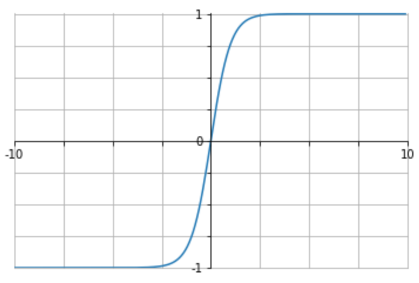
The training of a CNN for an image processing task means the calculation of a loss by comparing the output predictions with the ground-truths (targeted output). During training, the backward pass is computed. The backward pass can simply be described as the sensitivity of loss with respect to changes in the network parameters. The main objective of training a CNN is to minimize the loss function across the training dataset by modifying the network's trainable parameters, such as weights and biases, using the backward passes.

2.5.1 *Loss Function*

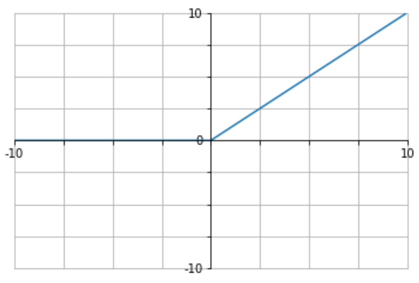
Loss functions are the differentiable functions used to guide the training process of a neural network. For the training of a CNN for an object detection task, networks are typically optimized for classification and regression elements. Mathematically, the loss function maps the network output y and the targeted output $\hat{y}$ on to a real value representing the penalty for the inaccuracy of predictions. The most widely-used loss functions for classification problem are Total Mean Squared Error (TMSE or L_2 loss), Cross Entropy Loss (CEL) and Softmax Loss. For a regression problem, loss functions may vary from network to network.



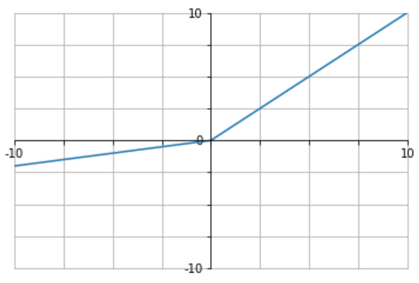
(a) Sigmoid function



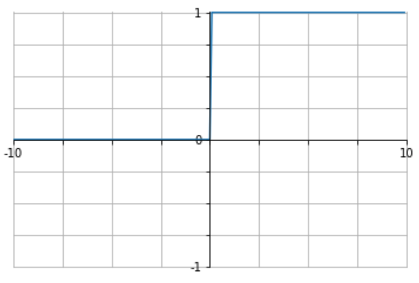
(b) Tanh function



(c) Relu function



(d) Leaky Relu function



(e) Heaviside step function

Figure 2.4.3: Different activation functions and their graphs

2.5.1.1 *Total Mean Squared Error (TMSE)*

The Total Mean Squared Error (TMSE) for the network output y and the targeted output $\hat{y}$ is defined as

$$\mathrm{TMSE} = \frac{1}{n \cdot l} \sum_{i=1}^{n} ||y_i - \hat{y_i}||^2, \tag{2.5.1}$$

where n and l represent the number of training samples and the number of output neurons respectively. The term $||y_i - \hat{y_i}||$ is the Euclidean distance between y and $\hat{y}$.

2.5.1.2 *Cross Entropy Loss (CEL)*

The cross entropy loss for the network output y and the targeted output $\hat{y}$ is defined as

$$\mathrm{CEL} = -\frac{1}{N} \sum_{i=1}^{N} [\hat{y_i} \log y_i + (1 - \hat{y_i}) \log(1 - y_i)], \tag{2.5.2}$$

where l is the number of output neurons.

2.5.1.3 *Softmax Loss*

Softmax loss is one of the most commonly used loss function in CNNs. It takes N-dimensional vector of real values and transforms it into the vector of real values in the range (0,1) such that the sum is equal to 1. Given an input matrix $X_{n,k}$, the Softmax loss is defined as

$$\sigma(X_{n,k}) = \frac{e^{(X_{n,k})}}{\sum_{j=1}^{K} e^{(X_{n,j})}}, \tag{2.5.3}$$

and the loss is calculated as

$$L = -\frac{1}{N} \sum_{n=1}^{N} \log(\frac{e^{(X_{n,k})}}{\sum_{K}^{j=1} e^{(X_{n,j})}}), \tag{2.5.4}$$

where N is the batch size and K is the number of identities (i.e. class labels).

A detailed description of the loss functions used in this thesis is given in subsection 4.2.3 and subsection 4.3.2.

2.5.2 *Learning with gradient descent*

As explained earlier in section 2.5, the main objective of training a CNN is to minimize the error between the targeted output and the network output by updating its parameters during the backward pass. This is done by calculating the analytic gradient of the loss function with respect to the network outputs, and driving the error gradient backward throughout the network's layers until the first layer is reached. Hence, this process is called "back-propagation of error with gradient descent"[RHW86].

Intuitively, the goal of the training algorithm is to minimize the loss function L of the network's parameters i.e. weights w and biases b ($L(v) = L(w, b) \approx 0$). In order to find the minimum of a loss function L, the analytic gradient ∇L is calculated. ∇L indicates the extent and direction of change in the loss function resulted by changing convolution parameters (weights w and biases b) by one unit. Gradient descent uses this information to alter the parameters in such a way that the error 'descends' to its minima.

The gradient descent update rule is defined as

$$v_i \leftarrow v_i - \eta \frac{\partial L}{\partial v_i} \tag{2.5.5}$$

or

$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i} \quad , \quad b_i \leftarrow b_i - \eta \frac{\partial L}{\partial b_i}, \tag{2.5.6}$$

where η is the learning rate. It is initialized by trial and error and it gradually decreases over training iterations. The terms $\frac{\partial C}{\partial w_i}$ and $\frac{\partial C}{\partial w_i}$ determine the rate of change of loss function with respect to any weight w and bias b in the network.

Despite its popularity, gradient descent converges very slow at a minimum loss setting. It often does not achieve convergence at all. To overcome this problem, *momentum* [Rud16] is introduced. Momentum helps accelerating gradients vectors in the right directions, which leads to faster convergence.

It does so by adding a fraction γ of the update vector of the previous step to the current update vector:

$$\theta'_i = \gamma\theta_i - \eta \frac{\partial L}{\partial \theta_i} \quad , \quad v_i \leftarrow v_i - \theta'_i \ , \tag{2.5.7}$$

where γ<1. Another problem of gradient descent is that the gradients for shallow layers are typically much smaller than the gradients of deeper layers, and the learning rates should be able to accommodate for this problem. There are several optimization techniques to tackle such problem. *AdaGrad* [DHS11] is one of such techniques that includes an adaptive learning rate, given as

$$\Delta v_i = -\eta \frac{1}{\sqrt{r + \delta}} \odot \frac{\partial L}{\partial v_i} \quad , \quad v_i \leftarrow v_i + \Delta v_i \ , \tag{2.5.8}$$

where $\odot$ represents an element-wise multiplication. δ is a constant that avoids the division by zero, whereas r is the sum of squares of past gradients. It is given as

$$r \leftarrow r + (\frac{\partial L}{\partial v_i} \odot \frac{\partial L}{\partial v_i}). \tag{2.5.9}$$

RMSprop [TH12], an extension of AdaGrad, introduces another parameter ρ that tackles the problem of AdaGrad's monotonically decreasing learning rate. The update rule for AdaGrad with RMSprop becomes

$$r \leftarrow \rho r + (1-\rho)(\frac{\partial L}{\partial v_i} \odot \frac{\partial L}{\partial v_i}), \tag{2.5.10}$$

where r is now the weighted sum of squared gradients.

Adaptive Moment Estimation, or *Adam* [KB14], is another adaptive learning rate optimization algorithm. Adam optimizer is one of most commonly used optimizer in modern deep learning applications. It is a combination of Momentum and RMSprop.

The update rule for Adam optimizer follows

$$s \leftarrow \rho_1 s + (1-\rho_1)\frac{\partial L}{\partial v_i} \quad , \quad \hat{s} \leftarrow \frac{s}{1-\rho_1^t} \ , \tag{2.5.11}$$

$$r \leftarrow \rho_2 r + (1-\rho_2)(\frac{\partial L}{\partial v_i} \odot \frac{\partial L}{\partial v_i}) \quad , \quad \hat{r} \leftarrow \frac{r}{1-\rho_2^t} \ , \tag{2.5.12}$$

$$\Delta v_i = -\eta \frac{\hat{s}}{\sqrt{\hat{r}}+\delta} \odot \frac{\partial L}{\partial v_i} \quad , \quad v_i \leftarrow v_i + \Delta v_i \ , \tag{2.5.13}$$

where s and $\hat{s}$ represent the estimate and its bias correction respectively for the first momentum, while r and $\hat{r}$ represent the same for the second momentum. ρ_1 and ρ_2 are the decay rates which are usually set to be 0.9 and 0.999 respectively. Empirically shown, Adam works robustly to the choices of hyperparameters and compares favorably to other adaptive learning algorithms [GBCB16].

2.6 OBJECT DETECTION

Object detection in images is one of the most fundamental and challenging problems in computer vision. In addition to identifying and validating multiple objects in an image and classifying them into certain classes, object detection also deals with the localization problem, together with a rough estimation of their sizes. By this definition, one may conclude that object detection is concerned with two sub-problems: classification and regression. The regression problem (also referred to as bounding-box regression) deals with the linear regression of bounding-boxes encapsulating objects of interest in an image. An ideal bounding-box is an axis-parallel rectangle that contains all parts of an object. Each bounding-box in an image is associated with a confidence score that estimates the probability of an object lying within it.

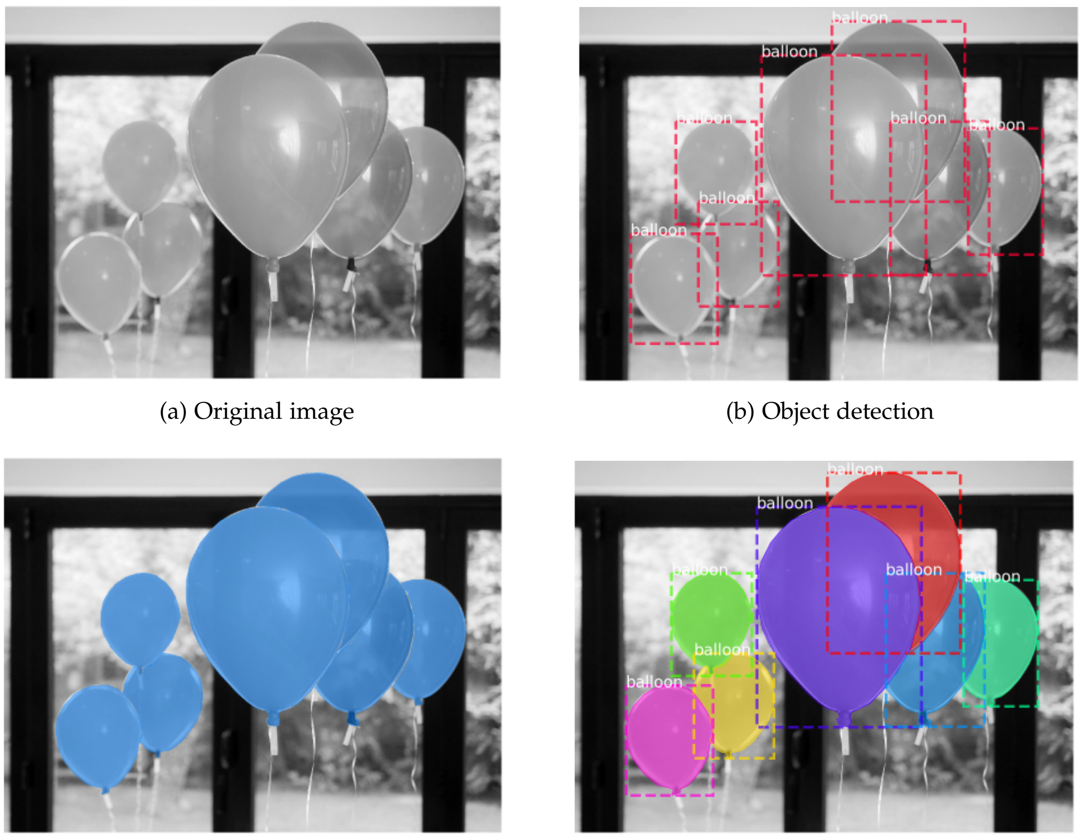
In this work, 'region-based' deep neural networks are specifically explored for object detection and instance segmentation. Typically, there are three basic steps to a region-based object detection framework. The first step is the regions of interest (RoI) or region proposals generation (see subsection 2.8.1). An algorithm or a model such as RPN (see subsection 4.2.1) generates rectangular bounding-boxes in an image which is the object localization component of an object detector. After region proposals generation, visual features are extracted for each of the bounding-boxes in the next step. These visual features decide whether a bounding-box contains an object or not. In the final step, overlapping boxes are combined into a single bounding-box using an external algorithm such as NMS (see subsection 2.8.3). The classifying component of the object detector then classifies object within the bounding-box into a certain class. The region-based object detection and segmentation frameworks are discussed in greater details in chapter 3 and chapter 4.

2.7 INSTANCE SEGMENTATION

Intuitively, segmentation is the understanding of an image on the pixel level, where the goal is to assign the class labels to each pixel in the image.

Unlike semantic segmentation, where each pixel in the image is catogarized, instance segmentation does not label every pixel in the image. Instead, the goal of instance segmentation is to detect specific objects in an image and create a mask around the object of interest. Instance segmentation combines classical object detection tasks (classification of multiple objects and localization using bounding-boxes) with semantic segmentation where the goal is to classify each pixel into a fixed set of categories without differentiating object instances.

Figure 2.7.1 illustrates the difference between semantic and instance segmentation. Semantic segmentation partitions the image into semantically meaningful parts and classifies each part into one of the pre-determined classes (i.e. balloon, 2.7.1c). Instance segmentation, however, segments the image for all individual objects, regardless of their association to the same class(2.7.1d).



(a) Original image

(b) Object detection

(c) Semantic segmentation

(d) Instance segmentation

Figure 2.7.1: Object detection vs semantic segmentation vs instance segmentation [Abd]

2.8 IMPORTANT OBJECT DETECTION AND SEGMENTATION CONCEPTS

In this section, a brief introduction to some of the important concepts used in object detection is provided.

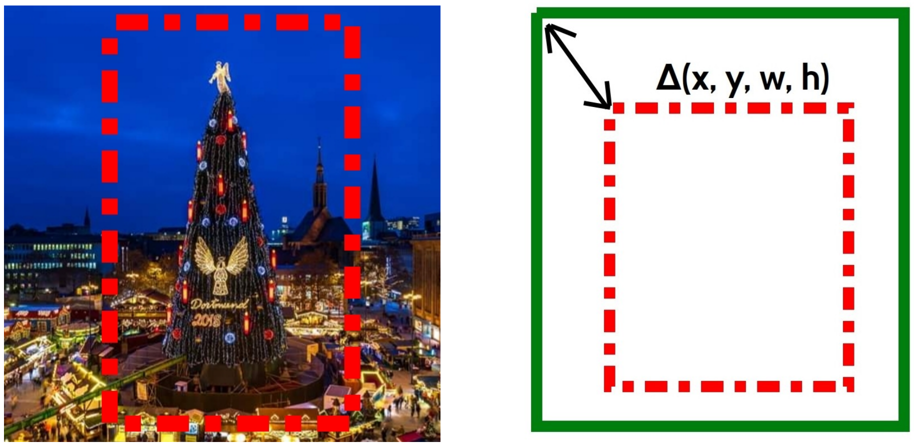
2.8.1 *Region of Interest (RoI)*

A region of interest, or more colloquially known as region proposal or bounding-box proposal, is a rectangular region in an input image that potentially contains an object. These proposals can be generated by some external algorithms such as Selective Search [UVDSGS13], Edge Box detection [ZD14] or by a Region Proposal Network

(RPN) [RHGS15] (see subsection 4.2.1).

A bounding-box is represented as a 4×1 vector containing its center location, width, and height (x, y, w, h). Each bounding-box in an image is accompanied by an objectness or confidence score of how likely the box contains an object.

The difference between two bounding-boxes is usually measured by the L2 distance of their vector representations (see Figure 2.8.1).



(a) Bounding-box encapsulating an object of interest

(b) Offset between two bounding-boxes

Figure 2.8.1: RoI in an image

2.8.2 *Intersection over Union (IoU)*

Intersection over union (IoU) is a measure based on Jaccard Index [H+89] that measures the similarity between predicted bounding-box B_p and the ground truth B_{gt}. It simply determines whether a detection is valid (True Positive) or not (False Positive) and is given by the overlapping area between the predicted bounding-box and the ground truth bounding-box divided by the area of union between them (Figure 2.8.2).

$$\mathrm{IoU} = \frac{\mathrm{area}(B_p \cap B_{gt})}{\mathrm{area}(B_p \cup B_{gt})}. \tag{2.8.1}$$

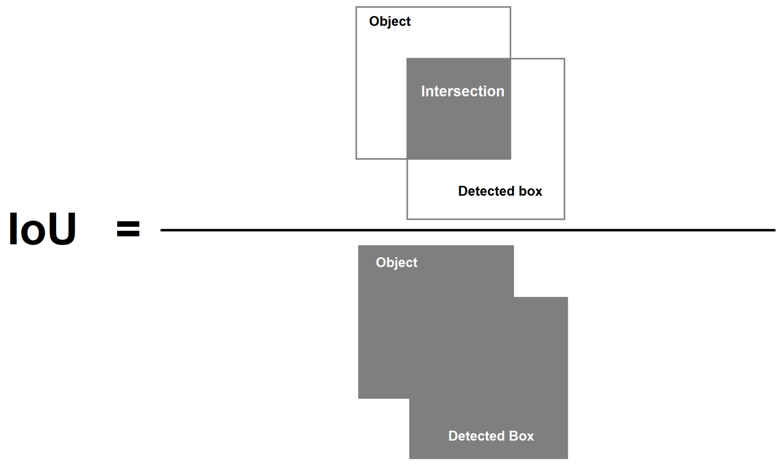


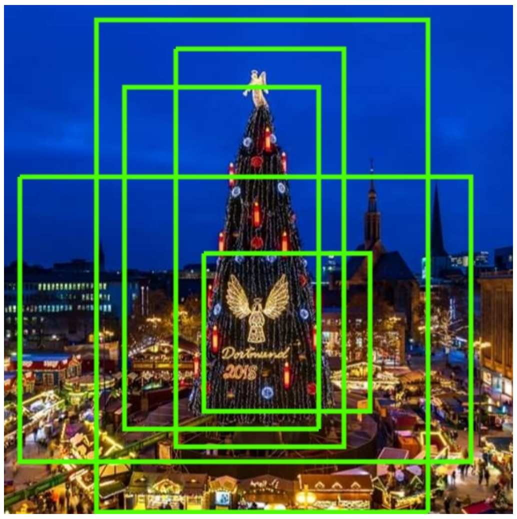
Figure 2.8.2: Area of overlap over area of union

2.8.3 *Non-Maxium Suppression (NMS)*

Non-Maxium Suppression (NMS) is a greedy algorithm used in most modern object detectors to merge overlapping bounding boxes (or region proposals). It sorts detections by their object confidence scores, takes the highest scoring detection and removes lower-scoring detections which have an IOU greater than a pre-defined threshold (Figure 2.8.3).

2.8.4 *Bounding-box regression (bounding-box refinement)*

Most of the modern object detectors utilize bounding-box regressors which are trained to look at an input region and predict the offset Δ(x, y, w, h) between the input region box and the ground truth box. If there is one regressor for each object class, it is called class-specific regression, otherwise, it is called class-agnostic (one regressor for all classes). A bounding-box regressor is often accompanied by a bounding-box classifier (confidence scorer) to estimate the confidence of object existence in the box. The classifier can also be class-specific or class-agnostic.


(a) Before NMS

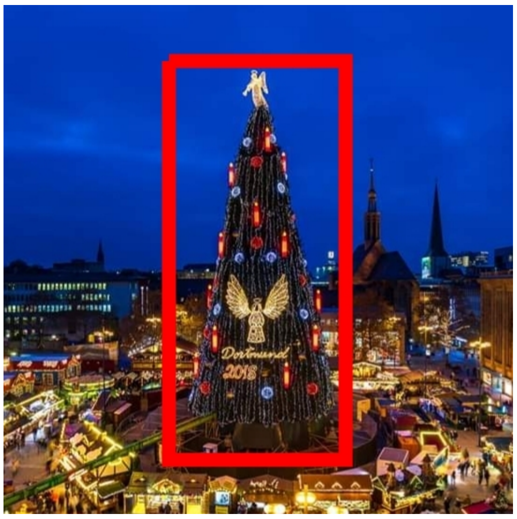

(b) After NMS

Figure 2.8.3: After applying NMS, only the bounding-boxes with highest confidence score remain.

3 RELATED WORK

This chapter provides a brief history of classical object detection methods that utilize Convolutional Neural Networks (CNNs) as well as their modern history. Furthermore, the methods that combine CNNs with region proposal generators will be briefly discussed. section 3.2 and section 3.3 discuss the fundamentals of region-based object detection and segmentation frameworks, which have inspired the development of the approaches presented in this work.

3.1 CLASSICAL METHODS

One of the fundamental tasks of computer vision is to enable computers to have the human-like interpretation of complex visual information. Such an interpretation can occur at several levels within an image, such as object detection and segmentation. Object detection is the task of identifying the presence of any instances of a given object class within an image whereas segmentation is the task of labeling each image pixel with the object class.

Object detection and segmentation are the most rapidly advancing areas of computer vision. Now, a computer can easily surpass an average human performance in object detection and segmentation, all thanks to the advanced computing technologies and large publicly available datasets. Even though, the popularity of aforementioned computer vision applications has increased drastically in the last decade, its history extends back to the early 1960s.

Prior to the application of deep neural networks for object detection and segmentation tasks, classic object detection frameworks were mainly based on extracting feature descriptors, such as Histograms of Orientated Gradients (HOG) [DT05] or scale-invariant feature transform (SIFT) [Low99] with subsequent classification by a linear classifier, such as SVM. In this section, these classic methods will be briefly described.

3.1.1 *Histogram of Oriented Gradients (HOG)*

Histogram of Oriented Gradients (HOG) is a dense feature descriptor for images, developed by Navneet Dalal et al. [DT05]. The basic idea is that the shape of structures can be characterized by the distribution of local intensity gradients or edge directions. HOG method captures these shapes in a region by extracting information about said gradients.
In practice, HOG first divides an image into a number of overlapping spatial regions (or blocks). These blocks are also referred to as histogram of oriented gradients (HOG) descriptors. Each block is then further divided into smaller $n \times n$ cells, where each cell contains a fixed number of gradient orientation bins over n number of pixels in it. Each pixel in the cell gives a score for a gradient orientation bin proportional to the gradient magnitude at that pixel (Figure 3.1.2). Once the intensity gradients are computed for each pixel in the cell, the gradient orientations are quantized into several bins.
In order to extract features and to classify regions from sub-images at multiple scales and aspect ratios, HOG utilizes a 'sliding window' (subsection 3.1.3) approach. Tiling these sliding windows with a dense overlapping grid of blocks or HOG descriptors and using the combined feature vector in a linear classifier such as support vector machines (SVMs) results in final output (object/non-object classification).

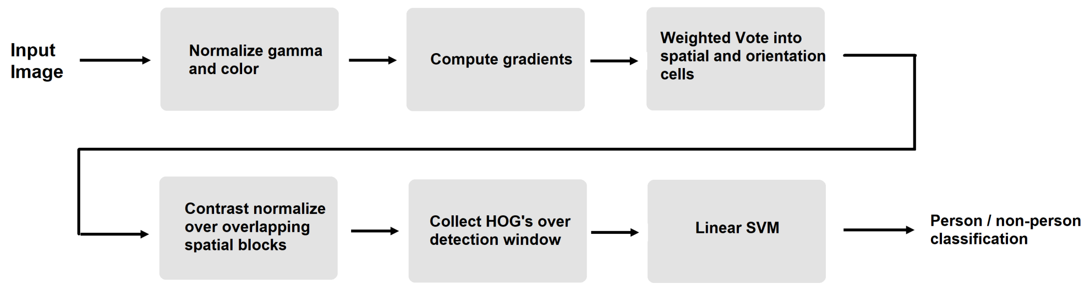
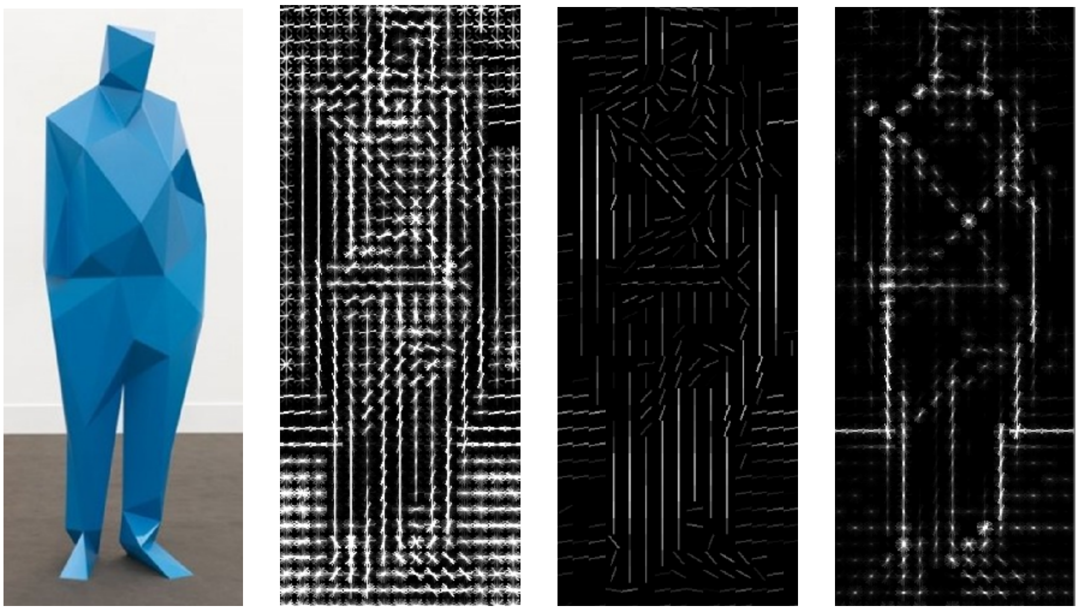

Input Image
Normalize gamma and color
Compute gradients
Weighted Vote into spatial and orientation cells
Contrast normalize over overlapping spatial blocks
Collect HOG's over detection window
Linear SVM
Person / non-person classification

Figure 3.1.1: The detector window is tiled with a grid of overlapping blocks in which HOG feature vectors are extracted. The combined vectors are fed to a linear SVM for object/non-object classification. [DT05].

As great HOG was at producing state-of-the-art results in the early 2000s, it lacked robustness with regard to occlusion and deformations. The accuracy and detection speed of HOG method were also prohibitive for many applications.



(a) Original image (b) HOG features (c) Stable orientations of HOG features (d) Cells with high continuity values.

Figure 3.1.2: Each pixel in an image is replaced with a gradient with a score proportional to the gradient magnitude at that pixel [Giv].

3.1.2 *Scale Invariant Feature Transform (SIFT)*

Scale Invariant Feature Transform (SIFT), originally developed by David Lowe in 1999 [Low99], is an image descriptor, which has been extensively used for the person/face localization and recognition. The SIFT descriptor has been proven to be very useful in practice for image matching and object recognition under real-world conditions because of its invariance to translations, rotations, and scaling transformations in the image domain. The operation of a SIFT descriptor can be divided into the following steps. The first step is the extraction of key-points or interest points from labeled grey-level images. These key-points are located in 2D space where the signal's variation exceeds some threshold. SIFT extracts these key-points by creating a scale space. This scale space is obtained by constructing a set of Gaussian-blurred images. Once the key-points are obtained, a feature vector is computed by finding the histograms of gradient directions in a local neighborhood around each key-point. This creation of feature vector like descriptor is the second step and a unique aspect of SIFT. In the third step, bad key-points such as edges and low contrast regions are eliminated and an array of orientation histograms is computed for the remaining key-points. Any further calculations are done relative to this orientation histogram which effectively cancels out the effect of orientation, making it rotation invariant. Finally, with the scale and rotation invariance in place, a Hough transform is performed to identify the clusters from a specific object. Next, the probability of a particular feature vector representing an object in the image is computed. The verification is performed as a least-squares solution, applied to the parameters obtained from the affine transformation[Low99].

3.1.3 *OverFeat (Sliding Window Approach)*

Early attempts at object detection using CNN relied heavily on sliding window processing. Such methods have proved to be very successful in several domains, including face, pedestrian, and text detection. One such network, namely OverFeat [SEZ$^+$13], won the ImageNet Large Scale Visual Recognition Challenge 2013 (ILSVRC 2013) [RL13]). Modern region-based object detectors owe their success to Overfeet which is considered to be the pioneer model of integrating the image classification and regression tasks into one Convolutional Neural Network. The main idea behind it is to perform classification at different locations (or regions) on multiple scales of an image in a sliding window fashion (i.e. by sliding an $n \times n$ window on the feature map obtained by last convolutional layer of the CNN), and to perform bounding-box regression (explained in subsection 2.8.4) to envelop an object more tightly.

This method first trains a CNN model for the image classification task. Then, it replaces the top classifier layers with a class-specific regression network and trains it to refine the boundaries of bounding-boxes at each spatial location. Finally, the resulting class scores and regressed bounding-boxes are aggregated using a greedy merging algorithm

Despite the success of such methods, many drawbacks remained. One of the major drawbacks was the redundancy of the processed windows (only partially containing an object), which increased the running time by a considerable amount. However, the idea of object localization (combination of image classification and regression) in Overfeat has been extended in the modern object detection task (where the idea is to localize the objects on multiple regions of images).

3.2 REGION-BASED CONVOLUTION NEURAL NETWORK (R-CNN)

In 2014, the very first region-based object detector was developed by Girshick et. al, namely Region-based Convolution Neural Network (R-CNN) [GDDM14].

R-CNN model consists of three modules. The first module generates region proposals by scanning the input image for possible objects using an external algorithm. There are various methods for the region proposals generation such as objectness [ADF12], Selective Search [UVDSGS13], Category-Independent Object Proposals [EH10], Constrained Parametric Min-Cuts (CPMC) [CS11], Multi-Scale Combinatorial Grouping [APTB+14] etc. The most commonly used region proposal generation method is Selective Search algorithm, which bottoms up nearly 2000 category independent RoIs or region proposals. These region proposals are then warped to match the input of a large multi-layered Convolutional Neural Network (CNN), which is the second module of R-CNN. This large CNN extracts a fixed-length feature vector from each region in an image.

The third module is a set of class-specific linear support vector machines (SVMs). In this module, feature vectors obtained from the CNN are fed to the said classifier and the set of classified region proposals are reduced using Non- Maximum Suppression (NMS). The feature vector is also fed to a linear regressor, leaving the expected bounding-boxes of objects in the input image. Three modules of an R-CNN are shown in Figure 3.2.1.

Although, the R-CNN was intuitive, it had few drawbacks. One of the major drawbacks was the fixed input size of region proposals, surpassed by warping image regions (regardless of its size, location or aspect) to the pre-defined dimensions before processing by the CNN. Another major drawbacks of R-CNN was its high computational cost; a result of passing warped sub-images individually through the CNN (with around 2,000 regions being generated at test time).

3.3 FAST REGION-BASED CONVOLUTION NEURAL NETWORK (FAST R-CNN)

Fast-R-CNN [Gir15] is the first successor of R-CNN, retaining most of the core notions of R-CNN and introducing few refinements. In Fast R-CNN, the main CNN with multiple convolutional layers is applied to the image for feature detection before proposing regions. That means it does not require several CNNs over several overlapping regions. Also the SVM is replaced with a Softmax layer, thus extending the neural network for predictions instead of creating a new model (Figure 3.3.1). Another refinement was the Region of Interest (RoI) pooling layer (see subsection 4.3.1).

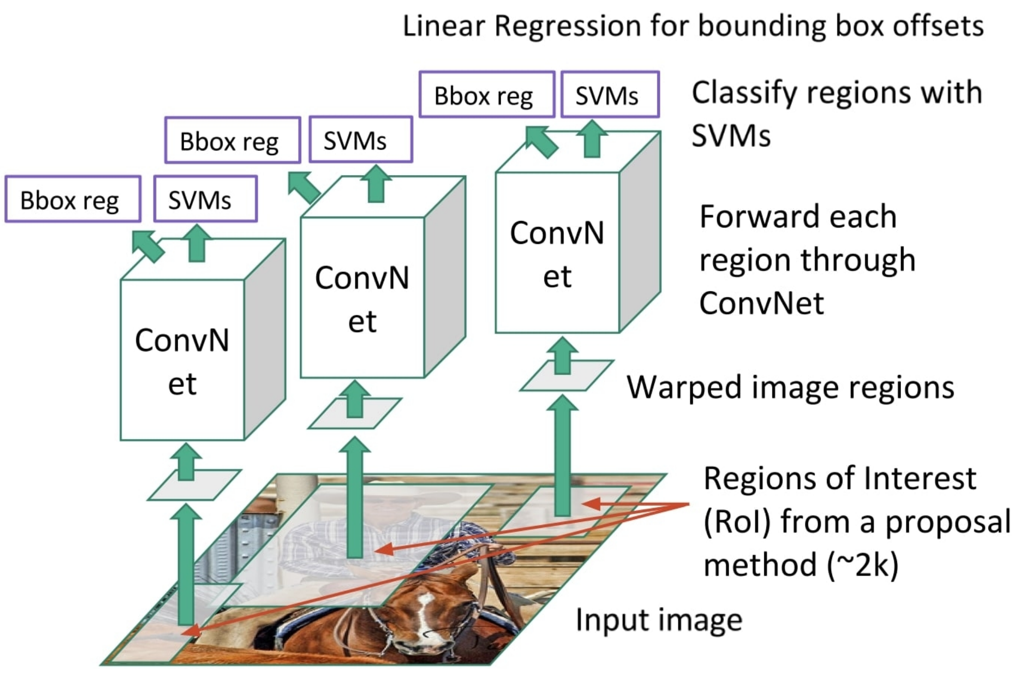

Linear Regression for bounding box offsets
Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Classify regions with SVMs
ConvNet
ConvNet
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

Figure 3.2.1: RCNN takes an input image, extracts around 2000 bottom-up region proposals, computes features for each proposal using a large CNN and then classifies each region using class-specific linear SVMs [GDDM14].

Fast R-CNN is almost 10× more computationally efficient and accurate than R-CNN as the region proposals are generated from the feature map instead of the original image. However, region proposals were still detected with the slow selective search method.

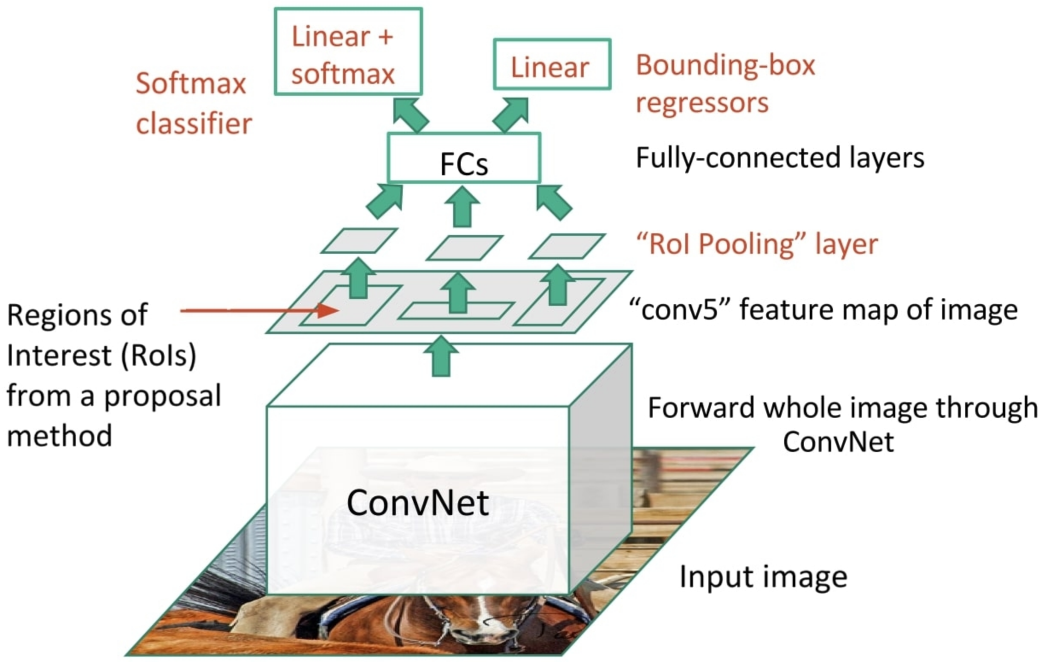


Figure 3.3.1: An input image and multiple regions of interest (RoIs) are input into a CNN. Each RoI is first pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: Softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss [Gir15].

4 METHODS

Previous chapters explained the fundamental concepts necessary to understand the object detection and the instance segmentation tasks along with their classic and modern history. This chapter provides the infrastructures and setup used in this thesis based on the related work. In section 4.1, a brief introduction to the evaluation matrices used in this work will be given. As explained earlier, this thesis is divided into two modules: object detection and instance segmentation. The methods and approaches used for the object detection task as well as the instance segmentation task will be described in greater detail along with their loss functions and training schemes in section 4.2 and section 4.3.

4.1 EVALUATION METRICS

In this section, the concepts required to understand the evaluation matrices used in this work will be briefly explained. The most useful and commonly used evaluation metric for the task of object detection is called mean Average Precision (mAP). This evaluation metric is used by most of the modern object detection and instance segmentation frameworks to compare their performances. In order to understand mAP, it is necessary to first have an intuition of sensitivity and specificity, which are the statistical measures of the performance of a binary classification test.

Sensitivity is also referred to as the true positive rate or the recall. It measures the proportion of actual positives that are correctly identified. For object detection task, declaration of true positives (TP) and false positives (FP) depends on a pre-defined IoU threshold (see subsection 2.8.2). The IoU threshold is usually set to 50%, 75% or 95%. The true positives (TP) and false positives (FP) are hypothesized as follows:

- True Positive (TP): The proportion of actual positives that are correctly identified with an IOU $\geqslant$ some pre-defined threshold. In the case of object detection, a true positive (TP) is a correctly-detected object in an image.

- False Positive (FP): The proportion of actual negatives incorrectly identified as positives with an IOU $<$ some pre-defined threshold. This can be thought of as a false detection i.e. if an object is detected in an image when there is none.

Specificity is also called the true negative rate. The true negatives (TN) and false negatives (FN) are defined as follows:

- True Negative (TN): The proportion of actual negatives that are correctly identified i.e. for object detection true negatives (TN)) are the cases where non-object regions are correctly identified as non-object regions
- False Negative (FN): A case, where a detector fails to detect an object.

4.1.1 *Precision*

Precision [EVGW+10] is the ability of a model to identify only the relevant objects, i.e. the percentage of correct positive predictions. It is given by

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} = \frac{\mathrm{TP}}{\mathrm{all\ detections}}. \tag{4.1.1}$$

4.1.2 *Recall*

Recall [EVGW+10] is the ability of a model to find all the relevant cases. It represents the proportion of the actual positives that are correctly identified. It is given by:

$$\mathrm{Recall} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} = \frac{\mathrm{TP}}{\mathrm{all\ positive\ cases}}. \tag{4.1.2}$$

4.1.3 *Precision-Recall curve*

The Precision-Recall curve is one of the most common metrics used to evaluate the performance of an object detection network.

An object detector of a particular class is considered good if its precision remains high as recall increases. A good object detector has the ability to identify all the relevant objects (zero false positives = high precision) and to find all the ground truth objects (zero false negatives = high recall).

While training an object detector, the number of detected objects increases with time. Consequently, the number of false positives also increases which results in the precision drop. This is the reason the precision-recall curve usually starts with the high precision values and gradually decreases as the object detector tries to retrieve all ground truth objects.

Figure 4.1.1 shows an illustration of a precision-recall curve. The recall value increases as the number of detected object increases. However, the precision value would drop and rise in a zig-zag fashion.

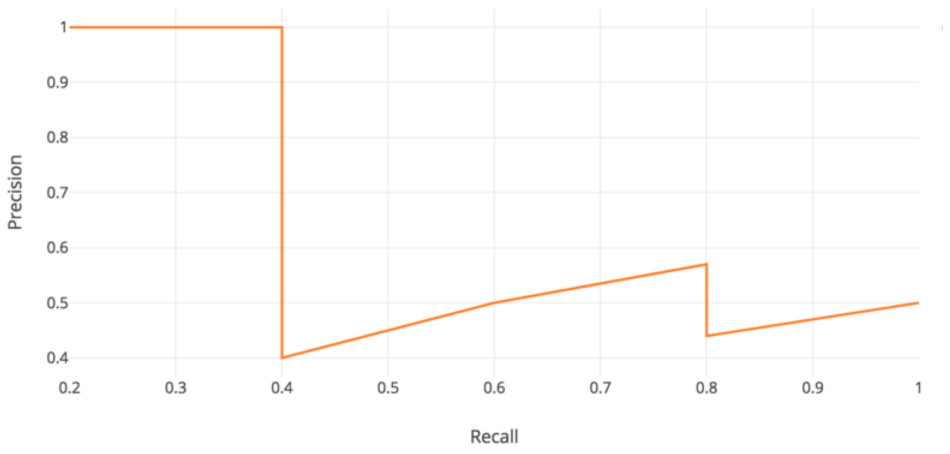


Figure 4.1.1: An example of a Precision-Recall curve [Hui]

4.1.4 *Average Precision (AP)*

The Average Precision (AP) is calculated by first smoothing out the zig-zag pattern of the precision-recall curve and then calculating the area under it. The average precision can also be used to compare the performances of different detectors.
In practice, a graph is plotted with recall $\hat{r}$ value in the range (0,1) and the precision value is replaced with the maximum precision for any recall $\geqslant \hat{r}$.

For example, in the PASCAL Visual Object Classes (VOC) challenge [EVGW+10] , the AP (average precision) is computed by averaging the precision at a set of eleven equally spaced recall levels i.e. $0, 0.1, 0.2, ..., 1$ (see (Figure 4.1.2)).
Mathematically, it can be expressed as

$$AP = \frac{1}{11} \sum_{r \epsilon (0.0...1.0)} AP_r \,, \tag{4.1.3}$$

$$= \frac{1}{11} \sum_{r \epsilon (0.0...1.0)} P_{interp(r)} \,, \tag{4.1.4}$$

with

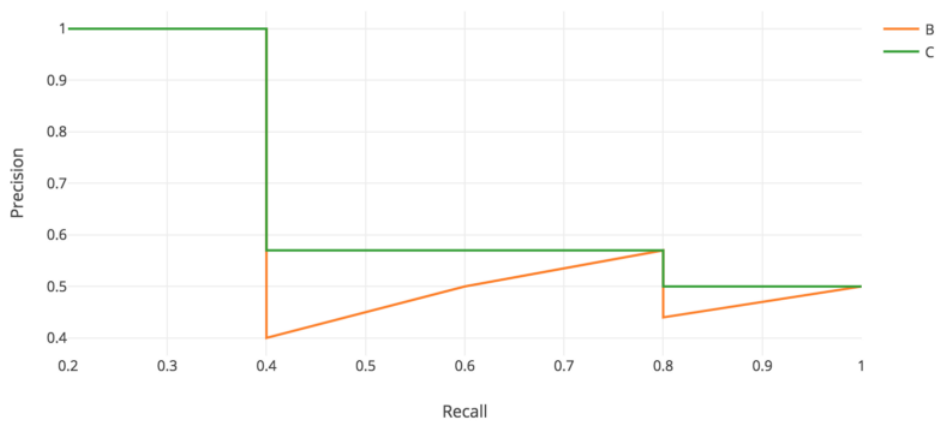


Figure 4.1.2: The highest precision value (the green curve) at the recall values (0, 0.1, 0.2, . . . , 0.9 and 1.0) [Hui]

$$P_{\mathrm{interp}(r)} = \max_{\hat{r} \geqslant r} p(\hat{r}) \;, \tag{4.1.5}$$

where $P_{\mathrm{interp}(r)}$ is an interpolated precision that takes the maximum precision over all recalls greater than r. $p(\hat{r})$ is the measured precision at recall $\hat{r}$.

4.1.5 *Mean Average Precision (mAP)*

Mean Average Precision (mAP) is simply the average of AP over all classes. It is often referred to as AP instead.

The next section explains the Faster R-CNN algorithm which was used for the first task of this thesis i.e. object detection task.

4.2 FASTER REGION-BASED CONVOLUTION NEURAL NETWORK (FASTER R-CNN)

Faster Region-based Convolution Neural Network (Faster R-CNN), developed by Shaoqing Ren et al. in 2016 [RHGS15], is the successor of Fast R-CNN [Gir15].

Despite the success achieved by Fast R-CNN, it still relies on an external algorithm to propose region proposals. The external region proposal generation methods such as Selective Search [UVDSGS13] algorithm typically rely on inexpensive features, which is a bottleneck in terms of running time. Selective Search greedily merges super pixels based on engineered low-level features and it takes 2 seconds per image in a CPU implementation. Edge Boxes [ZD14], another external region proposer showed a better trade-off between proposal quality and speed, at 0.2 seconds per image. Nevertheless, it still consumes as much running time as the detection network [RHGS15].

Faster R-CNN, in contrast, exploits the feature extractor output for classification and regression and uses this information to generate region proposals which leads to an effective solution, namely Region Proposal Networks (RPNs). RPN shares convolutional layers with the object detection networks, making proposal computation nearly cost-free (e.g. 10 ms per image).

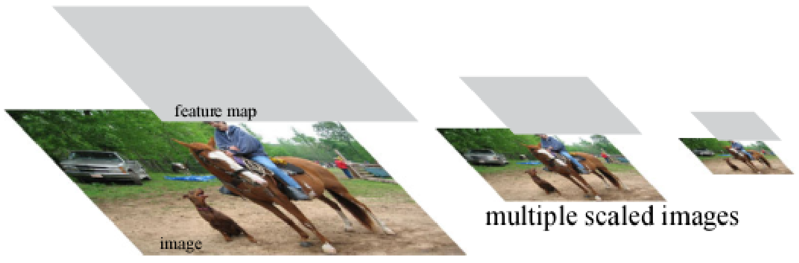
4.2.1 *Region Proposal Network (RPN)*

An RPN is a Fully Convolutional Network (FCN) [LSD15] that simultaneously predicts object bounds and objectness scores at each position of an image. It is trained end-to-end specifically for the task of generating high quality region proposals.

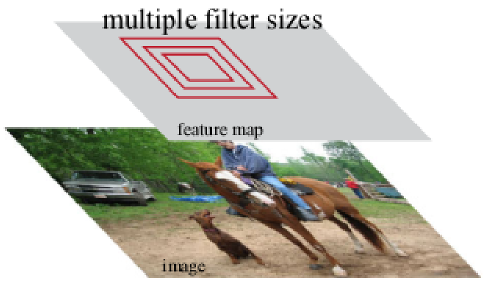
RPNs can efficiently bottom-up region proposals with a wide range of scales and aspect ratios by using "anchor" boxes.

The prevalent networks such as Spatial Pyramid Pooling [RHZS14], Fast R-CNN [Gir15] and OverFeat [SEZ$^+$13] etc used the pyramids of images (4.2.1a) or pyramids of filters (4.2.1b), while RPN uses the pyramid of regression references scheme (4.2.1c), which avoids enumerating images or filters of multiple scales or aspect ratios, enabling Faster R-CNN to perform better when trained and tested using single-scale images.
Faster R-CNN can be considered as a unification of a region proposer (RPN) and an object detector (Fast R-CNN). A training scheme that alternates between fine-tuning for the region proposal task and then fine-tuning for object detection, while keeping the proposals fixed results in a unified network with convolutional features that are shared between both tasks (Figure 4.2.2).

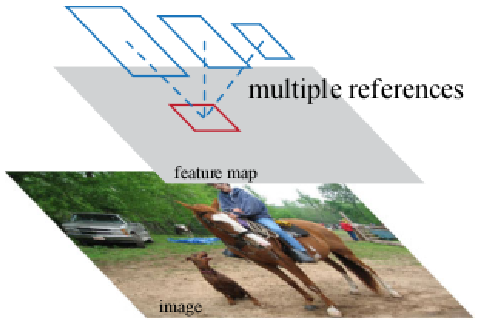
An RPN shares computation with the Fast R-CNN detector by utilizing all the convolutional layers of the core CNN. This could be thought of as a Fully Convolutional Network (FCN) [LSD15]. An RPN takes an image as input and outputs a set of rectangular object proposals, each associated with an objectness score. It does so by taking an $n \times n$ spatial window of feature map from the last shared convolutional layer and by sliding a small network over it (as illustrated in Figure 4.2.3). Each sliding window is then mapped to a lower-dimensional feature map, which is further fed



(a) Pyramids of images and feature maps



(b) Pyramids of filters with multiple scales/sizes



(c) Pyramids of reference boxes in the regression functions

Figure 4.2.1: Different schemes for addressing multiple scales and aspect ratios [RHGS15]

into two sibling fully-connected layers: a bounding-box regression layer (reg) and a bounding-box classification layer (cls).

4.2.2 *Anchors*

For each sliding-window location, k number of region proposals are predicted simultaneously (see Figure 4.2.3). These k number of proposals are referred to as anchors. For three aspect ratios 1:1, 1:2 and 2:1 (as per original implementation) each sliding window yields a total of $k = 9$ anchors.

The $\mathtt{reg}$ layer has $4k$ outputs (its center location and its height and width) and the $\mathtt{cls}$ layer has $2k$ scores (probability of a proposal containing an object or not). For a convolutional feature map of a size $W \times H$, there are $(W \times H) \times k$ anchors in total.

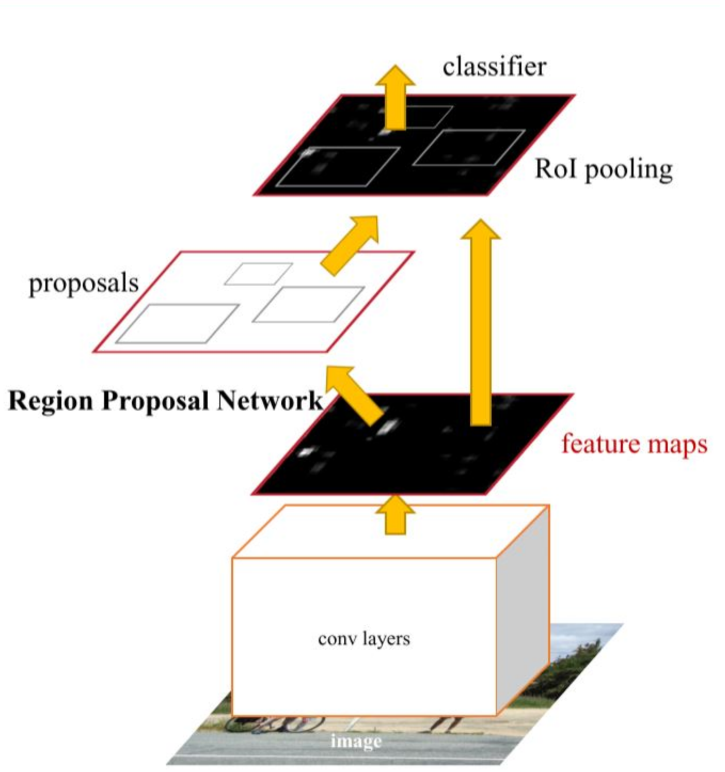


Figure 4.2.2: Faster R-CNN represented as a single unified network for object detection [RHGS15]

4.2.3 *Loss Function*

Each anchor is assigned a binary class label (of being an object or not). An anchor is assigned a positive label when it has either the highest IoU overlap with a ground-truth box, or an IoU overlap higher than 0.7 with any ground-truth box. It is also possible that a single ground-truth box may assign positive labels to multiple anchors. The first condition is adopted in some rare cases where the second condition fails to find any positive sample.

Likewise, when an anchor has an IoU ratio lower than 0.3 for all ground-truth boxes, it is assigned a negative label.

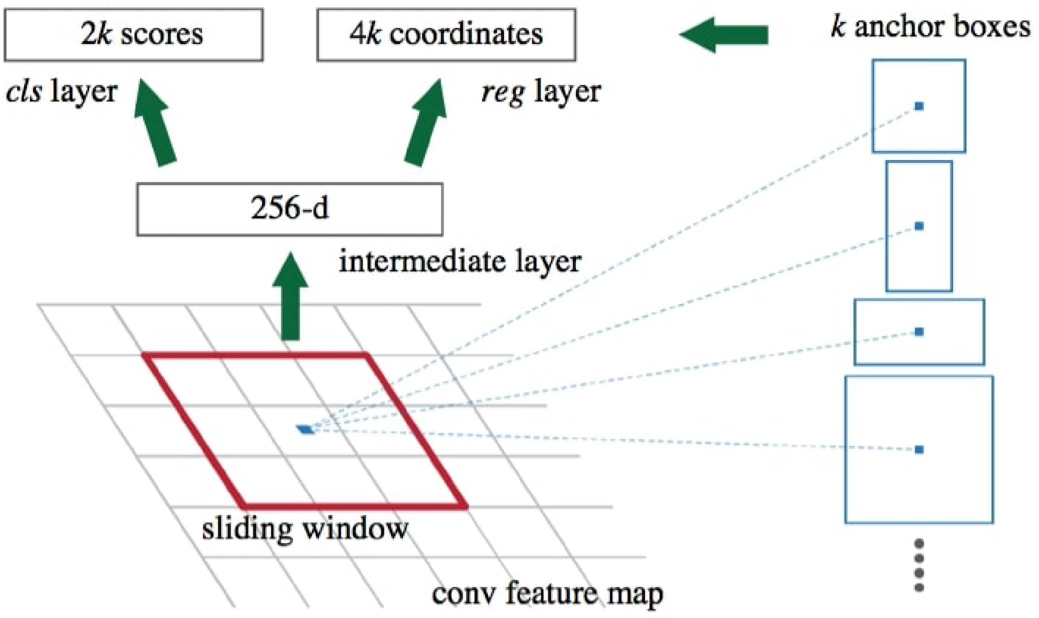


Figure 4.2.3: Working of an RPN [RHGS15]

With these definitions, the loss function for an image is defined as

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_{i^*}) + \lambda \frac{1}{N_{reg}} \sum_i p_{i^*} L_{reg}(t_i, t_{i^*}). \qquad (4.2.1)$$

Here, i is the index of an anchor in a mini-batch. p_i is the predicted probability of anchor i being an object whereas p^* is the ground-truth label which is 1 for a positive anchor, and 0 for the negative one. t_i is a vector representing the 4 parameterized coordinates of the predicted bounding-box, and t_i^* represents coordinates of the ground-truth box associated with a positive anchor.

L_{cls} is the log loss over two classes (object vs not object) for classification whereas L_{reg} represents the regression loss. The term $p_i * L_{reg}$ indicates that the regression loss is activated only for positive anchors and is disabled otherwise.

The classification loss is normalized by N_{cls}, which is the mini-batch size, and the regression loss by the number of anchor locations N_{reg}. The regression loss is also weighted by a balancing parameter λ. For bounding-box regression, the 4 coordinates are calculated as

$$t_x = {}^{(x-x_a)}/_{w_a} \quad , \quad t_y = {}^{(y-y_a)}/_{h_a} \ , \tag{4.2.2}$$
$$t_w = \log({}^{w}/_{w_a}) \quad , \quad t_h = \log({}^{h}/_{h_a}) \ , \tag{4.2.3}$$
$$t_x^* = {}^{(x^*-x_a)}/_{w_a} \quad , \quad t_y^* = {}^{(y^*-y_a)}/_{h_a} \ , \tag{4.2.4}$$
$$t_w^* = \log({}^{w^*}/_{w_a}) \quad , \quad t_h^* = \log({}^{h^*}/_{h_a}) \ , \tag{4.2.5}$$

where x, y, w, and h indicate the anchor box's center coordinates and its width and height. Variables x, x_a, and x^* are for the predicted box, anchor box, and ground-truth box respectively (likewise for y, w, h) [RHGS15].

4.2.4 *Training Procedure*

In this section, the training procedure for Faster R-CNN as a unified network is explained, where computation is shared between the RPN and the object detector e.g. Fast R-CNN.

Both RPN and the object detector (Fast R-CNN) can be trained independently. To take advantage of shared computation, a method is required that allows sharing convolutional layers between two networks. A Faster R-CNN network can be trained in following ways:

4.2.4.1 *Alternate Training*

In this iterative approach, RPN is first trained to propose region proposals, which are then used to train the detector. The tuned object detection network is then used to initialize RPN again.

4.2.4.2 *Approximate Joint Training*

This approach merges RPN and Fast R-CNN networks into one unified network as shown in Figure 4.2.2. During training of the detector, the forward pass generates region proposals which are treated as fixed, pre-computed proposals and the backward propagated signals from both the RPN loss and the Fast R-CNN loss are combined for the shared layers. This approach, however, ignores the derivative w.r.t. the proposal boxes' coordinates that are also network's responses.

4.2.4.3 *Non-Approximate Joint Training*

Since the bounding-boxes that are predicted by as RPN are also functions of the input and the above approach ignores the gradients w.r.t. the bounding-boxes' coordinates, an extra layer is required. This layer is called ***RoIPool*** layer (will be explained later in subsection 4.3.1) and it is differentiable w.r.t. the bounding-boxes' coordinates. The rest training procedure is same as the above approach.

4.2.4.4 *4-Step Alternating Training*

The work presented in this thesis uses a 4-Step alternating training approach to learn shared features via alternating optimization. In the first step, the RPN is initialized with ImageNet pre-trained weights and fine-tuned end-to-end for the region proposal generation task. In the second step, a separate detection network (also initialized with the ImageNet pre-trained weights) is trained using the proposals generated by step 1. The two networks do not share convolutional layers until this point. In the third step, the detection network initializes the RPN, but the shared convolutional layers are fixed and only the layers unique to RPN are fine-tuned. Finally, keeping the shared convolutional layers fixed, the unique layers of the detector (Fast R-CNN) are fine-tuned. Hence, both networks share the same convolutional layers resulting into a unified network [RHGS15].

The second module of this thesis is called instance segmentation task, which combines classical object detection tasks (classification of multiple objects and localization using bounding-boxes) with semantic segmentation where the goal is to classify each pixel into a fixed set of categories without differentiating object instances. The method used for instance segmentation task is called Mask R-CNN which is explained as follows:

4.3 MASK REGION-BASED CONVOLUTION NEURAL NETWORK (MASK R-CNN)

Mask Region-based Convolution Neural Network (Mask R-CNN) [HGDG17] can be considered as an extension of the Faster R-CNN. Mask R-CNN adds a small overhead to the Faster R-CNN by adding an extra branch for predicting an object mask in parallel with the existing branches for classification and bounding-box regression (Figure 4.3.1).

Mask R-CNN follows the same two-stage training procedure as Faster R-CNN. The first stage (region proposal generation via RPN) is identical to Faster R-CNN, however, in the second stage, Mask R-CNN outputs a binary mask for each RoI along with class prediction and bounding-box regression. The masks are produced by using up-sampling and deconvolutional layers to resize the output feature map to its original image dimensions.

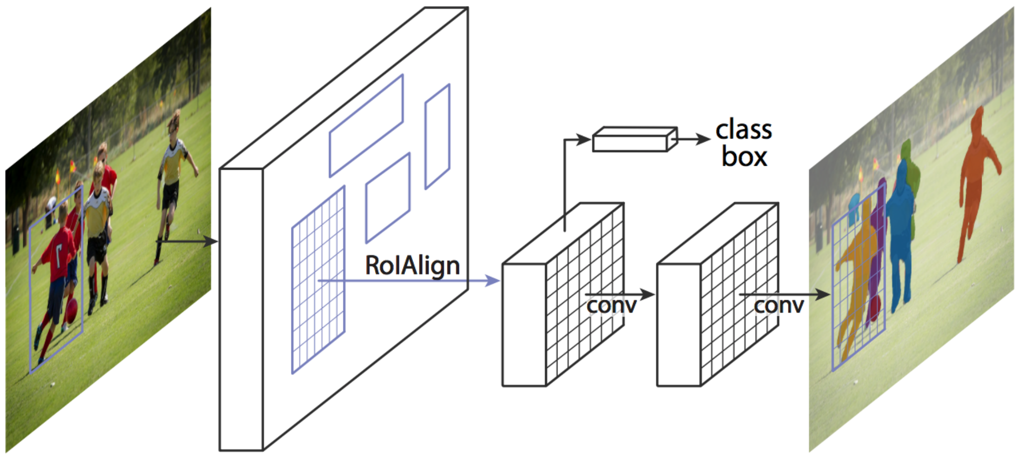


Figure 4.3.1: Mask R-CNN for instance segmentation[HGDG17]

.

Mask R-CNN uses the ***RoIAlign*** (subsection 4.3.1) layer instead of ***RoIPool*** layer. The difference between two layers (i.e. RoI pooling approaches) is explained in the next section. Since the Faster R-CNN was not designed for pixel-to-pixel alignment between the network inputs and outputs, it performs coarse spatial quantization for feature extraction which is fine for just classification task. However, the misalignment caused by ***RoIPool*** creates an offset for object's binary mask. To fix the misalignment, Mask R-CNN uses the ***RoIAlign*** which is quantization-free.

4.3.1 *RoIPool vs RoIAlign*

The prevalent networks such as Fast R-CNN and Faster R-CNN used RoI pooling method for the purpose of speeding up training/testing process. Fast R-CNN applied the ROI pooling method to allow only one forward/backward pass for multiple RoIs in one input image.

The ***RoIPool*** takes two inputs: a fixed-size feature map obtained after several convolutions and max-pooling operations and an $N \times 4$ matrix representing a list of RoIs, where N is a number of RoIs. This matrix includes the coordinates of the region along with its height and width (x,y,h,w).

Multiple RoIs in an image are of different sizes. For each RoI of size $h \times w$, the ***RoIPool*** takes a section of the corresponding input feature map and scales it to size

$H \times W$. This scaling is done by max-pooling. (The max-pool kernel is [h/H], [w/W] respectively). RoI pooling allows the network to forward the image just once for different scaled RoIs.

Incorporating ***RoIPool*** layer in Mask R-CNN has a downside as it involves two steps of coordinates quantization: from the original image to feature map and from feature map to RoI feature. Those quantizations cause a huge loss of location precision. Therefore Mask R-CNN replaces the ***RoIPool*** layer with the ***RoIAlign*** layer as the RoI pooling method. ***RoIAlign*** removes those two quantizations and manipulate coordinates on continuous domain, which increase the location accuracy greatly.

Figure 4.3.2 shows that the ***RoIAlign*** computes the exact values of the input features at four regularly sampled locations using bilinear interpolation in each RoI bin, and aggregate the result (using max or average). No quantization is performed on any coordinates involved in the RoI, its bins, or the sampling points.

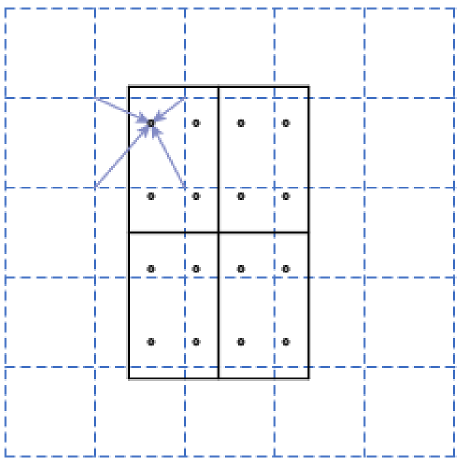


Figure 4.3.2: A feature map with 5×5 bins is mapped to an RoI of 2×2 bins (the dots represent the 4 sampling points in each bin) [HGDG17]

.

4.3.2 *Loss Function*

The multi-task loss for each sampled RoI is defined as

$$L = L_{cls} + L_{reg} + L_{mask}. \qquad (4.3.1)$$

The classification loss L_{cls} and the regression loss L_{reg} in Equation 4.3.1 are identical to those defined in Equation 4.2.1 for Faster R-CNN. The mask loss L_{mask} is the average binary cross-entropy loss, only including k-th mask if the RoI is associated with the ground truth class k. It has the dimension of Km^2 for each RoI (K binary masks of resolution $m \times m$, one for each of the K classes). Since the model is trying to learn a mask for each class, there is no competition among classes for generating masks.
The Mask R-CNN adopts the same 4-step training scheme as Faster R-CNN for training.

The next chapter discusses the implementation details and the experimental setup for the methods described above. Additionally, the modifications made to these networks will also be characterized, giving motivation for these changes. The datasets used for the training and testing will be described along with the evaluation metrics used to quantify the results.

5 EXPERIMENTS

This chapter provides further description of the experimental setup and implementation details of the proposed methods for object detection and segmentation along with problems encountered during experiments and the conclusions drawn from them. The parameters of the proposed experimental setup will also be justified in this chapter. Additionally, the modifications made to the networks will also be characterized, giving motivation for these changes in section 5.4. The section 5.1 gives a description of the datasets on which the experiments have been conducted. Finally, different methods will be compared and evaluated using the selected threshold values and results from the presented methods will be discussed.

5.1 DATASETS

The performance of deep neural networks in object detection relies on a large amount of training data along with the computing power. Collecting and annotating a dataset of sufficient size is a hectic and time taking process. Therefore, most research in object detection and segmentation is performed on publicly available datasets. There are several such datasets that serve the purpose. The standard benchmark datasets (PASCAL VOC [EVGW+10] and MS COCO [LMB+14]) were selected for the work in this thesis as they provide diversity and support multiple object categories.

5.1.1 *PASCAL Visual Object Classes (PASCAL VOC)*

The Pascal Visual Object Classes (VOC) [EVGW+10] is a publicly available dataset of images together with ground truth annotation and standardized evaluation software. It supports five types of challenges: classification, detection, segmentation, action classification, and person layout. For the work presented in this thesis, two different versions of PASCAL VOC were used i.e. PASCAL VOC 2007 and PASCAL VOC 2012. Both versions were used separately as well as in combination to train the networks. It contains around 10k images for training and around 5k images for testing and validation, categorized among 20 daily life object.
The PASCAL VOC dataset by default supports the evaluation metric for mean Aver-

age Precision (mAP) at IoU = 0.50 (which is referred to as AP_{50} in MS COCO dataset). The main motivation behind using this dataset was to select the best backbone architecture for the models and to provide the base for further evaluation on MS COCO dataset (subsection 5.1.2).

5.1.2 *Microsoft Common Objects in COntext (MS COCO)*

Microsoft Common Objects in Context (MS COCO) [LMB+14] is another publicly available dataset, widely used for object detection and segmentation task. MS COCO is relatively much larger dataset which consists of around 200k labeled images, containing 1.5 million object instances and categorizing them among 80 different categories. In addition, it provides an option to calculate the mean Average Precision (mAP) for the objects at different IoU scales and for the objects of different sizes as well (see Table 5.1.1).

5.2 EXPERIMENTAL SETUP

Both region proposal networks and object detection/segmentation networks were trained and tested on images of a single scale (i.e. 1000×600 pixels for Faster R-CNN and 1000×800 for Mask R-CNN). For Mask R-CNN, the rescaled images of scale 600×400 were used (reason explained in section 5.4). As original implementation of Faster R-CNN [RHGS15] suggests that the image pyramid or feature pyramid is not required to produce regions of multiple scales, hence only a single scale for images was considered in this work. For anchors, three aspect ratios of 1:1, 1:2 and 2:1 (k=9 anchors per sliding window of 3 x 3) and three scales with box areas of 128^2, 256^2 and 512^2 pixels were chosen. The anchor boxes that cross with image boundaries were neglected during training. In general, there are roughly 20,000 anchors for a 1000 x 600 image (i.e. 1000 x 600 x 9 $\approx$ 20000). After ignoring cross-boundary anchors, about 6000 anchor boxes per image are left. Including cross-boundary anchors in training produces large training error resulting in not converging. While training RPN, some proposals highly overlap each other. So, the Non-Maximum Suppression (NMS) was applied on proposals based on their **cls** scores to reduce redundancy. IoU=0.7 was chosen for NMS resulting in final 2000 proposals. Different settings for train-time and test-time proposals and three different RoI pooling methods namely ***RoIPool***, ***RoIPool*** and ***crop_and_resize*** (explained in section 5.4) were also used. A detailed comparison is given in Table 5.7.1 and Table 5.9.1.

AP at different IoU values	
AP	AP averaged over interval IoU=[0.50 0.95] (primary challenge metric)
AP_{50}	AP at IoU=0.50 (PASCAL VOC metric)
AP_{75}	AP at IoU=0.75 (strict metric)
AP across scales	
AP_{small}	AP for small objects (area $< 32^2$)
AP_{medium}	AP for medium objects ($32^2 <$ area $< 96^2$)
AP_{large}	AP for large objects (area $> 96^2$)

Table 5.1.1: Average precision across different scales

Two different version of PASCAL VOC dataset (i.e. 2007 and 2012) were used for the experiments. The main motive behind using the PASCAL VOC dataset was to select the best-performing backbone architecture for Faster R-CNN. Further experiments with Faster R-CNN and Mask R-CNN were carried out with selected best-performing backbone architectures on MS COCO dataset for further evaluation. All these experiments were carried out using Nvidia GeForce GTX 1080 GPU.

5.3 BACKBONE ARCHITECTURES

Multiple backbone neural network architecture were used in this thesis. This section provides a brief introduction of these backbone architectures. Selection of the best architecture will be justified later.

5.3.1 *VGG-16*

VGG is a Convolutional Neural Network model proposed by K. Simonyan and A. Zisserman [SZ14]. In the original implementation of Faster R-CNN, VGG-16 was used as the baseline backbone architecture. This deep neural network consists of thirteen convolutional layers along with three fully connected layers.

In this thesis, VGG-16 network was used as a reference for the training of Faster R-CNN. The modifications made to this specific backbone architecture will be explained later in section 5.4. An illustration of the architecture is shown in Figure 5.3.1.

The kernel height and width for all the convolutional layers in VGG-16 is same (i.e. 3). They also share the same stride and the same zero padding (i.e. 1). The pooling layers have a grid size of 2×2 with stride = 2. VGG-16 backbone for Faster R-CNN uses ReLU as the activation function (Equation 2.4.3), and Softmax as the loss function (Equation 2.5.4).

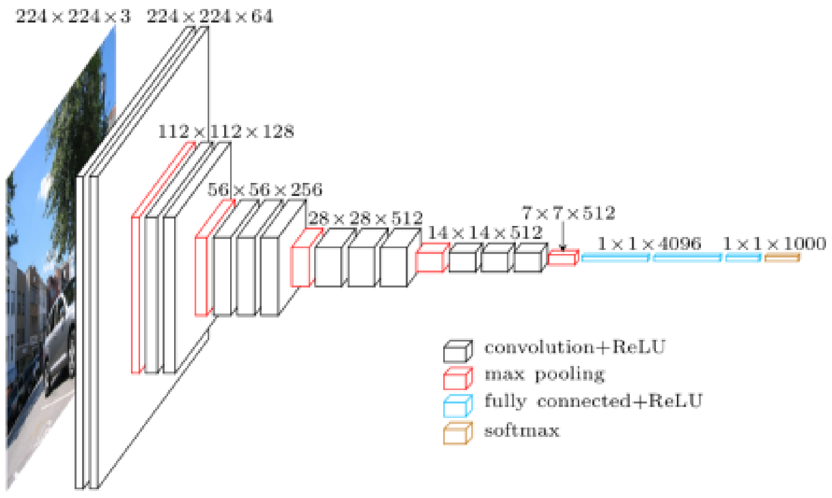


Figure 5.3.1: Macroarchitecture of VGG-16 [Cor]

5.3.2 *Residual Deep Neural Networks (ResNet 50, 101, 152)*

Residual deep neural network (ResNet) models were introduced in 2015 by He et al. [HZRS16]. ResNet models have boosted the performance of many computer vision applications due to their powerful representational ability.

Deep neural networks are prone to over-fitting because of their massive layers. Hence, making neural networks deeper is a common trend in the research community.

However, increasing network depth by simply stacking layers together might result in notorious vanishing gradient problem. As the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient infinitely small resulting in the rapid performance degradation. The solution to this problem is ”Deep Residual learning framework”.

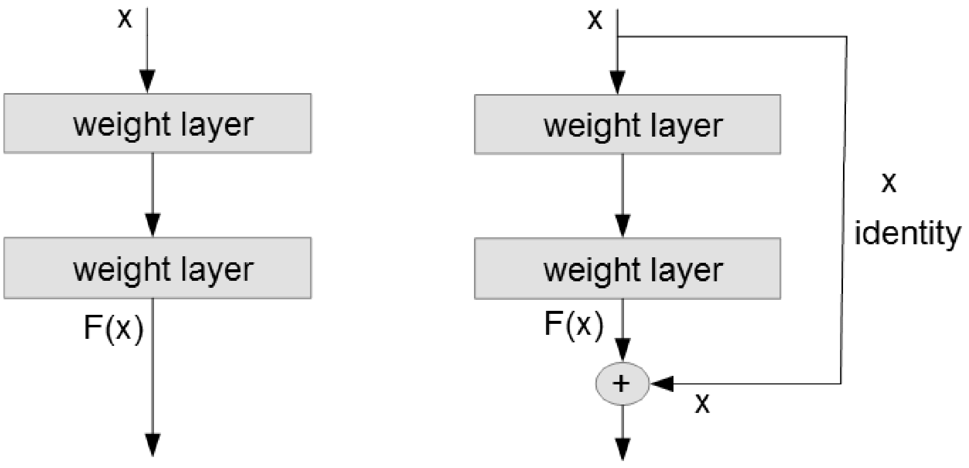
The deep residual networks consist of sequentially stacked residual blocks. Figure 5.3.2 shows the difference between a plain block and a residual block. Instead of learning a direct mapping of $x \to y$ using function $H(x)$ which is a few stacked non-linear layers (Figure 5.3.2, a), the author [HZRS16] introduces the "identity shortcut connections" (Figure 5.3.2, b) and defines a residual function as

$$F(x) = H(x) - x\,, \tag{5.3.1}$$

or

$$H(x) = F(x) + x\,, \tag{5.3.2}$$

with F(x) being the stacked non-linear layers and x the identity function (input = output). The author suggests [HZRS16] that it is easier to optimize the residual mapping function F(x) than to optimize the original, unreferenced mapping H(x).



(a) Plain block. (b) Residual block.

Figure 5.3.2: Identity mapping in Residual blocks

In this thesis, three different variants of ResNet models were used, namely ResNet 50, ResNet 101 and ResNet 152. A detailed comparison of three architectures is given in Table 5.3.1. Building blocks are shown in brackets, with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2 [HZRS16].

5.4 NOTABLE MODIFICATIONS

In this thesis, the implementation of Faster R-CNN and Mask R-CNN adopted the default end-to-end training/testing scheme as the original implementation. Learning

layer name	output size	ResNet 50		ResNet 101		ResNet 152	
conv1	112 x 112	7 x 7, 64, stride 2					
		3 x 3 max pool, stride 2					
conv2_x	56 x 56	$\begin{pmatrix} 1\text{x}1, 64 \\ 3\text{x}3, 64 \\ 1\text{x}1, 256 \end{pmatrix}$	x 3	$\begin{pmatrix} 1\text{x}1, 64 \\ 3\text{x}3, 64 \\ 1\text{x}1, 256 \end{pmatrix}$	x3	$\begin{pmatrix} 1\text{x}1, 64 \\ 3\text{x}3, 64 \\ 1\text{x}1, 256 \end{pmatrix}$	x 3
conv3_x	28 x 28	$\begin{pmatrix} 1\text{x}1, 128 \\ 3\text{x}3, 128 \\ 1\text{x}1, 512 \end{pmatrix}$	x 4	$\begin{pmatrix} 1\text{x}1, 128 \\ 3\text{x}3, 128 \\ 1\text{x}1, 512 \end{pmatrix}$	x 4	$\begin{pmatrix} 1\text{x}1, 128 \\ 3\text{x}3, 128 \\ 1\text{x}1, 512 \end{pmatrix}$	x 8
conv4_x	14 x 14	$\begin{pmatrix} 1\text{x}1, 256 \\ 3\text{x}3, 256 \\ 1\text{x}1, 1024 \end{pmatrix}$	x 6	$\begin{pmatrix} 1\text{x}1, 256 \\ 3\text{x}3, 256 \\ 1\text{x}1, 1024 \end{pmatrix}$	x 23	$\begin{pmatrix} 1\text{x}1, 256 \\ 3\text{x}3, 256 \\ 1\text{x}1, 1024 \end{pmatrix}$	x 36
conv5_x	7 x 7	$\begin{pmatrix} 1\text{x}1, 512 \\ 3\text{x}3, 512 \\ 1\text{x}1, 2048 \end{pmatrix}$	x 3	$\begin{pmatrix} 1\text{x}1, 512 \\ 3\text{x}3, 512 \\ 1\text{x}1, 2048 \end{pmatrix}$	x 3	$\begin{pmatrix} 1\text{x}1, 512 \\ 3\text{x}3, 512 \\ 1\text{x}1, 2048 \end{pmatrix}$	x 3
	1 x 1	average pool, 1000-d fc, softmax					

Table 5.3.1: ResNet 50, 101 and 152 architectures [HZRS16]

rate was set to be 0.001 for first 50k iterations for PASCAL VOC dataset and 350k for MS COCO and then it was reduced to 0.0001 for rest of the iterations. The first notable modification followed [HRS+17]. The ***crop_and_resize*** operator (ported from Tensorflow to Pytorch) was used instead of ***RoIPool*** as the RoI pooling method for Faster R-CNN. The ***crop_and_resize*** operator extracts crops from the input RoIs and resizes them using nearest neighbor sampling to a common output size of 14×14 and then max-pools them to 7×7 to match the input of fully connected layer. Secondly, instead of N = 2 images and R = 128 RoIs per image, N = 1 image and R = 256 RoIs were used during a single forward/backward pass. By doing this, it avoids the gradient accumulation across multiple batches which slows down the training and requires extra operators in Pytorch. The original implementations of Faster R-CNN and Mask R-CNN ignored the small proposals ($<$ 16 pixels in height or width in the original scale). The third modification is concerned with keeping such proposals. This step proved to be helpful for the detection of smaller objects.

Another notable modification in Faster R-CNN is concerned with the replacement of baseline backbone architecture VGG-16 with ResNet models (ResNet 50, 101, 152). For Mask R-CNN, the initial experiments with the same hyper-parameter settings as the original implementation were carried out. However, the ***RoIAlign*** layer as RoI pooling method requires much more GPU memory consumption than it was available for experimentation. The reason behind this problem is that the ***RoIAlign*** layer does not perform quantization, instead performs two steps of bilinear interpolation (from the original image to the feature map and from feature map to the RoI feature) to manipulate coordinates on the continuous domain. To find the solution to this problem, RPN batch size was reduced from R = 256 to R = 64 (keeping N = 1) along with the image scale (from 1000×600 to 600×400). The Mask R-CNN model with ResNet 101 backbone did not perform well and the mAP dropped by a considerable amount due to the smaller RPN batch size. Due to the smaller RPN batch size, several top-ranked anchor boxes were discarded resulting in huge mAP drop. Mask R-CNN with ***crop_and_resize*** as RoI pooling method, however, showed comparable results. A detailed comparison for the Mask R-CNN is given in Table 5.9.1.

5.5 FASTER R-CNN WITH DIFFERENT BACKBONE ARCHITECTURES

As a start, the experiments were conducted with Faster R-CNN and PASCAL VOC 2007 dataset. The shared convolutional layers were initialized with the pre-trained ImageNet weights. Other layers were randomly initialized by drawing weights from a zero-mean Gaussian distribution with standard deviation 0.01. Models were also trained from scratch (without using pre-trained ImageNet weights). Table 5.5.1 shows that the re-implementation of Faster R-CNN using baseline model VGG-16 with aforementioned modifications outperformed the original implementation which achieved the mAP of 69.9 with the detection rate of just 5 fps. The re-implementation of Faster R-CNN with VGG-16 backbone and ***crop_and_resize*** RoI pooling method performed almost 3 times faster achieving the mAP of 70.69 when trained using ImageNet weights. Also, this re-implementation of Faster R-CNN with VGG-16 backbone was done without gradients accumulation (i.e. $N = 1$ image per GPU). Faster R-CNN with ResNet models performed even better than VGG-16 architecture, however, the detection rate dropped. Increasing the layers in ResNet models results in higher mAP but detection becomes slower. The ResNet 152 achieves the highest mAP 75.59 on PASCAL VOC 2007.

Backbone architecture	From Scratch	Pre-Trained	Detection Rate
VGG16 (original)	**n/a**	**69.9**	**5fps**
VGG16 (modified)	71.02	70.69	14 fps
ResNet 50	67.86	74.15	8 fps
ResNet 101	74.21	75.09	7.5 fps
ResNet 152	75.57	75.59	6.6 fps

Table 5.5.1: Faster R-CNN with different backbones trained on PASCAL VOC training set 2007 and validated on PASCAL VOC validation set 2007

In the next experiment, the Faster R-CNN with the two best-performing backbone architectures in terms of accuracy and detection rate were trained on a combined PASCAL VOC training set (2007 + 2012). The networks were validated on PASCAL VOC validation set 2012 and tested on PASCAL VOC validation set 2007. The Faster R-CNN was able to achieve the highest mAP of 79.8 with ResNet 101 backbone with the detection rate of 7.5 fps.

Backbone architecture	Validation mAP	Test mAP	Detection Rate
ResNet 101	88.63	79.8	7.5 fps
ResNet 152	86.7	78.84	6 fps

Table 5.5.2: Faster R-CNN trained on PASCAL VOC training set 2007+2012 and tested on PASCAL VOC validation set 2007. The number of iterations increased from 70k to 110k.

5.6 FASTER R-CNN WITH DIFFERENT IOU SCALES AND OBJECT SIZES

Further experimentation with Faster R-CNN was done on MS COCO dataset. MS COCO dataset provides an option to evaluate the mAP for different IoU scales. The mAP averaged for IoU ϵ [0.5 : 0.05 : 0.95] (COCO's standard metric), the mAP at IoU = 0.75 and the mAP at IoU = 0.5 (PASCAL VOC's metric) were calculated. In addition, the mAP for objects of different sizes (AP_S, AP_M, and AP_L) was also evaluated (see Table 5.1.1). The Faster R-CNN with ResNet 101 and ResNet 152 backbone architectures was able to outperform the baseline model with VGG-16 backbone (Table 5.6.1).

The mAP at all IoU scales improved by a considerable amount (i.e. 10 - 13%). Note that, the re-implementation of Faster R-CNN includes the RoI pooling method of

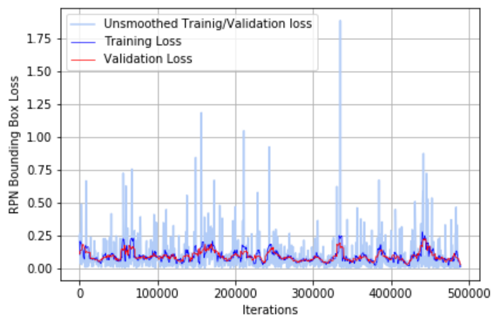
Backbone architecture	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	Detection rate
VGG16 (original)	**21.9**	**42.7**	**30.1**	**n/a**	**n/a**	**n/a**	**n/a**
ResNet 101	31.7	52.0	33.8	13.3	36.3	48.1	5.2 fps
ResNet 152	32.0	52.9	34.5	12.5	36.0	48.5	3.8 fps

Table 5.6.1: Faster RCNN with different backbone architectures trained on MS COCO 2014 dataset

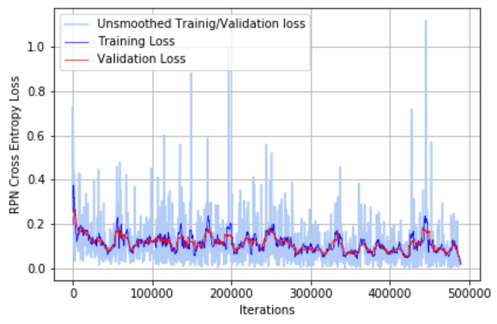
crop_and_resize instead of ***RoIPool*** on MS COCO dataset as well. . As it can be seen from Table 5.6.1 that the Faster R-CNN with ResNet 152 backbone showed slightly better results than the one with ResNet 101. However, when it comes to the objects of smaller size and medium size (i.e. under 96^2 pixels) ResNet 101 outperforms ResNet 152 backbone. In addition it gives better detection result of 5.2 fps. So, as a speed-accuracy trade-off, ResNet 101 was selected as the final backbone architecture and the experiments were performed with different settings for further evaluations. Figure 5.6.1 shows the evaluation of losses for the Faster R-CNN with ResNet 101 backbone trained on MS COCO 2014 dataset. Figure 5.6.1 (a) and Figure 5.6.1 (b) represent the bounding box regression loss and the classification loss for region proposal generator (RPN) respectively. Figure 5.6.1 (c) and Figure 5.6.1 (d) represent the bounding-box regression loss and the classification loss for the detector. The total average loss for unified network is shown in Figure 5.6.1 (e).

5.7 FASTER R-CNN WITH DIFFERENT NUMBER OF TRAINING/TESTING TIME PROPOSALS

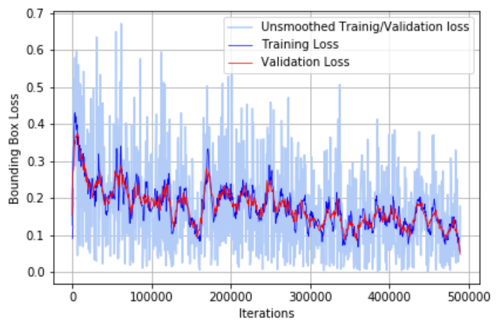
Next, the experiments were performed on the Faster R-CNN with different RoI pooling methods, NMS activation and different number of training/testing time proposals. First, the model was trained without Non-Maximum Suppression (NMS). NMS is kept off at test time as well. Without NMS, the RPN outputs 6000 region proposals during training and 2000 proposals at test time instead of 2000 and 300. As it can be seen in Table 5.7.1 the mAP droped by a considerable amount at all scales. It was conjectured that the reason of this gap is due the fact that without NMS each object is associated with multiple anchor boxes and multiple region proposals consequently. As a result, repetitive region proposals generate larger classification error. It was analyzed that this property mainly is attributed to the `cls` term of the RPN. In the next experiment, the NMS was kept on while training but disabled at test



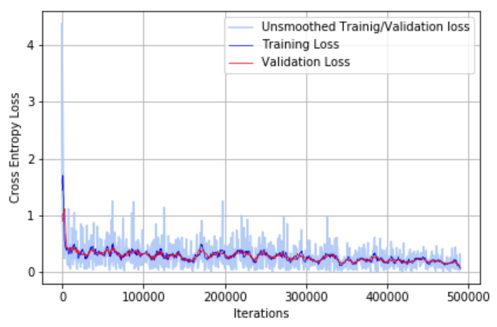
(a) RPN bounding-box loss



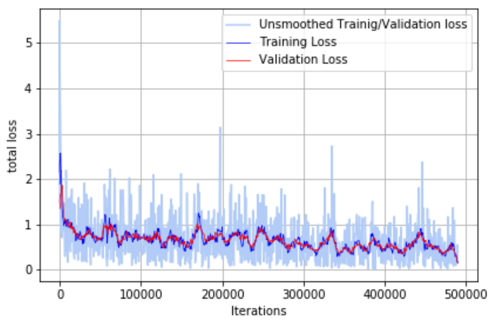
(b) RPN cross entropy loss



(c) Bounding-box loss



(d) Cross entropy loss



(e) Total Average Loss

Figure 5.6.1: Faster R-CNN with ResNet 101 backbone trained on MS COCO 2014 dataset. Total number of iterations is 490k. Learning rate is 0.001 for first 350k iterations and 0.0001 for the rest. RoI pooling method is ***crop_and_resize***.

time. The model showed comparable results with 2000 test time proposals instead of 300 proposals indicating that the top-ranked RPN proposals are accurate and it does not harm the detection mAP significantly. However, fewer proposals reduce the region-wise fully-connected layers' cost yielding better detection rate.

RoI pooling method	NMS	Train-time proposals	Test-time proposals	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
crop_and _resize	**On**	**2000**	**300**	**31.7**	**52.0**	**33.8**	**13.3**	**36.3**	**48.1**
crop_and _resize	Off	6000	2000	10.2	29.9	11.8	0.9	0.13	27.1
crop_and _resize	On	2000	2000	31.4	52.0	33.6	14.0	36.0	47.4
RoIPool	On	2000	300	22.2	48.4	16.7	11.0	26.6	30.9

Table 5.7.1: Faster RCNN (ResNet 101 Backbone) under different hyper-parameter settings

5.8 FASTER R-CNN WITH ORIGINAL ROI POOLING METHOD

Next, an experiment was set up with the original RoI pooling method for the Faster R-CNN using ***RoIPool*** layer instead of the modified ***crop_and_resize*** layer keeping other hyper-parameters fixed. The results obtained from this experiment verified that the object detector with modified RoI pooling layer (i.e. ***crop_and_resize***) performs better than the one with with original ***RoIpool*** by a considerable amount. The mAP for the objects of smaller size (AP_S) specifically dropped by ~15%. It is conjectured that this is due to the fact that original ***RoIPool*** involves two steps of coordinates quantization: from the original image to feature map and from feature map to RoI feature. Those quantizations cause a huge loss of information, specifically in case of objects of smaller size (under the area of 32^2 pixels).

5.9 MASK R-CNN WITH DIFFERENT BACKBONE ARCHITECTURES

In the next experiment, the Mask R-CNN was trained with the two best performing backbone architecture (deducted from the performance of Faster R-CNN) on MS COCO dataset. The original implementation of Mask R-CNN uses ResNet 101 and

ResNeXt 101 backbone architectures as baseline. The original implementation was done on Tesla M40 8 GPU machine (16GB per GPU with a batch size of 2 images per GPU). The re-implementation of Mask R-CNN with same hyper-parameters was not possible due to the limitations of computation power. As the available setup (Nvidia GTX1080 3 GPU machine with 8GB per GPU) is not powerful enough to perform the two steps of bilinear interpolation for R=256 RoIs per image of 1000×800 scale. In addition, one forward pass of the Mask R-CNN includes deconvolution and un-pooling operations to assign class labels to each pixel of the original image which requires additional memory for computation. To find the solution of said problems, the first experiment was conducted with the reduced image scale of 600×400 instead of 1000×800 and the reduced RPN batch size of 64 instead of 256. The mini-batch size was also reduced from $N = 2$ to $N = 1$ image per GPU. As it can be seen from Table 5.9.1, the Mask R-CNN with both ResNet 101 and ResNet 152 failed to replicate the results and the mAP dropped by a large amount.

Backbone architecture	RoI pooling method	Image scale	RPN batch size R	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
ResNet 101 (original)	*RoIAlign*	1000×800	256	33.1	54.9	34.8	12.1	35.6	51.1
ResNeXT 101 (original)	*RoIAlign*	1000×800	256	37.1	60.0	39.4	16.9	39.9	53.5
ResNet 101	*RoIAlign*	600×400	64	24.1	37.2	25.9	0.39	21.8	48.4
ResNet 152	*RoIAlign*	600×400	64	19.8	30.9	21.0	0.19	14.5	44.1
ResNet 101	*crop_and _resize*	1000×800	256	31.2	51.7	32.5	11.9	35.2	48.5

Table 5.9.1: Mask R-CNN with different backbone architectures trained on MS COCO 2014 dataset

As the deconvolutional network is an integral part of Mask R-CNN which is essential to produce masks of the objects, it cannot be compromised. However, the RoI pooling method could be replaced to deal with memory consumption issue. In the next experiment, The Mask R-CNN was incorporated with ***crop_and_resize*** layer from Faster R-CNN and replaced the ***RoIAlign*** layer. By replacing ***RoIAlign*** with ***crop_and_resize***, the two steps of bilinear intorpolation can be avoided, which reduces the memory consumption. With ***crop_and_resize*** as RoI pooling method, the Mask R-CNN was able to train on the original scale of images and the original RPN batch size. The Mask R-CNN model with ResNet 101 backbone architecture and ***crop_and_resize*** as RoI pooling method shows comparable results. It was able to achieve mAP of 51.7% on AP_{50} scale. In case of objects of smaller sizes (AP_S and AP_M), the Mask R-CNN with ***crop_and_resize*** was able to show very close results as the baseline model with ResNet 101 backbone architecture. In terms of detection rate, the Mask R-CNN with ***crop_and_resize*** performed at 4.2 fps as compared to the original implementation which has the detection rate of 5 fps with ResNet 101 backbone.

6 CONCLUSION

The goal of this thesis was to analyze the performance of state-of-the-art region-based deep learning architectures for object detection and instance segmentation using different backbone architectures along with other modifications and also finding a speed-accuracy trade-off for said object detection and segmentation frameworks in order to detect and segment objects with higher detection rate and accuracy.

Two such frameworks were explored for said tasks, namely Faster Region-based Convolutional Neural Network (Faster R-CNN) and Mask Region-based Convolutional Neural Network (Mask R-CNN). Both frameworks consist of two basic modules: a region proposal generator (i.e. RPN) and a detector. The Region Proposal Network (RPN) shares computation with the detector by utilizing the convolutional layers of the core CNN. The implementation of both Faster R-CNN and Mask R-CNN followed a 4-Step alternating training approach, where the RPN was first initialized with ImageNet pre-trained weights and fine-tuned end-to-end for the region proposal generation task followed by the training of a separate detection network using the proposals generated in the first step. The detection network then initialized the RPN, but the shared convolutional layers were fixed and only the layers unique to RPN were fine-tuned. Finally, keeping the shared convolutional layers fixed, the unique layers of the detector were fine-tuned. Hence, both networks shared the same convolutional layers resulting in a unified network. The Mask R-CNN, in addition, performs the deconvlolution operation that assigns class labels to each pixel of the original image. The Faster R-CNN was first re-implemented with the baseline backbone architecture VGG-16. The re-implementation was done with ***crop_and_resize*** RoI pooling method, where the input RoIs were cropped and resized using nearest neighbor sampling to a common outputs size of 14×14 and then max-pooled to the final 7×7 RoI feature. This RoI pooling method enabled the Faster R-CNN to outperform original implementation on the PASCAL VOC 2007 dataset. The re-implementation of Faster R-CNN with VGG-16 backbone achieved the mAP of 70.69% with the detection rate of 14 fps, which is nearly three times faster than the original implementation (i.e. 5 fps). The Faster R-CNN was then implemented using deep residual networks (ResNets). The Faster R-CNN with three different ResNet backbone architecture (i.e ResNet 50, 101, and 152) achieved the mAP of 74.15, 75.09, and 75.59 on the PASCAL VOC 2007 data set. It was concluded that going deeper with ResNet architecture

reduces the detection rate. The backbone architecture ResNet 101 with the detection rate of 7.5 fps was then selected as a speed-accuracy trade-off for further evaluation on MS COCO dataset. The Faster R-CNN was also implemented with the original RoI pooling method (using ***RoIPool*** layer), which verifies that removing quantization while mapping from original image to RoI feature improves mAP by a considerable amount.
The RoI aligning method (***RoIAlign*** layer), by Mask R-CNN also avoids the quantization. The main difference between ***RoIAlign*** and ***crop_and_resize*** is that the ***crop_and_resize*** still max-pools from 14×14 RoI feature to 7×7 window (input to the FC layer), while ***RoIAlign*** manipulates the coordinates on continuous domain and computes the exact values of the input feature using bilinear interpolation to map it to 7×7 window. However, the two steps of bilinear interpolation (from input to feature map and from feature map to RoI feature) for $R = 256$ RoIs per image requires a lot of computation and a powerful GPU is required for that. Instead, we incorporated ***crop_and_resize*** from Faster R-CNN to the Mask R-CNN. The Mask R-CNN with ResNet 101 backbone re-implementation was able to show the comparable results. It was able to achieve the mAP at scales of $AP = 31.2$, $AP_{50} = 51.7$, and $AP_{75} = 32.5$ (as compared to the original implementation = 33.1, 54.9, and 34.8). One drawback of ***crop_and_resize*** operator could be the small offset in object masks. As this operator includes one step of quantization from RoI feature to 7×7 common input. The information loss could effect the deconvolution operation, resulting in small offset between output mask and ground-truth mask.

6.1 FUTURE WORK

There are various aspects that can be considered for future work. Mask R-CNN framework could be extended to estimate human poses when adapted for key-points detection. Instead of detecting and segmenting a human as an object, if Mask R-CNN is trained to predict key-points (e.g K number of masks for each key-point e.g. left hand, right foot etc), the desired task can be achieved. Another possible future work for Faster R-CNN and Mask R-CNN could be the comprehension of natural language expressions referring to particular objects within an image (e.g "The man in black jacket" or "A window on the left"). This could be done by focusing on incorporating better measures of visual context into said models. Referring expression datasets are still under development, however, there are relatively smaller datasets available e.g. refCOCO that can serve the purpose.

BIBLIOGRAPHY

[Abd] Waleed Abdulla, *Splash of color: Instance segmentation with mask r-cnn and tensorflow.*, https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46/, Accessed: 2018-12-09.

[ADF12] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari, *Measuring the objectness of image windows*, IEEE transactions on pattern analysis and machine intelligence **34** (2012), no. 11, 2189–2202.

[APTB$^+$14] Pablo Arbeláez, Jordi Pont-Tuset, Jonathan T Barron, Ferran Marques, and Jitendra Malik, *Multiscale combinatorial grouping*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 328–335.

[Cas] Alex Castrounis, *Artificial intelligence, deep learning, and neural networks, explained*, https://www.kdnuggets.com/2016/10/artificial-intelligence-deep-learning-neural-networks-explained.html/, Accessed: 2018-12-03.

[Cor] Matthieu Cord, *A brief report of the Heuritech Deep Learning Meetup kernel description*, https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/, Accessed: 2018-11-21.

[CS11] Joao Carreira and Cristian Sminchisescu, *Cpmc: Automatic object segmentation using constrained parametric min-cuts*, IEEE Transactions on Pattern Analysis & Machine Intelligence (2011), no. 7, 1312–1328.

[DDS$^+$09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, *Imagenet: A large-scale hierarchical image database*, Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, IEEE, 2009, pp. 248–255.

[Den12] Li Deng, *The mnist database of handwritten digit images for machine learning research [best of the web]*, IEEE Signal Processing Magazine **29** (2012), no. 6, 141–142.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, Journal of Machine Learning Research **12** (2011), no. Jul, 2121–2159.

[DT05] Navneet Dalal and Bill Triggs, *Histograms of oriented gradients for human detection*, Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol. 1, IEEE, 2005, pp. 886–893.

[EH10] Ian Endres and Derek Hoiem, *Category independent object proposals*, European Conference on Computer Vision, Springer, 2010, pp. 575–588.

[EVGW+10] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman, *The pascal visual object classes (voc) challenge*, International journal of computer vision **88** (2010), no. 2, 303–338.

[Fuk88] Kunihiko Fukushima, *Neocognitron: A hierarchical neural network capable of visual pattern recognition.*, Neural networks **1** (1988), no. 2, 119–130.

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio, *Deep learning*, vol. 1, MIT press Cambridge, 2016.

[GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 580–587.

[Gir15] Ross Girshick, *Fast r-cnn*, Proceedings of the IEEE international conference on computer vision, 2015, pp. 1440–1448.

[Giv] *Sketchable histograms of oriented gradients for object detection*, https://www.semanticscholar.org/paper/Sketchable-Histograms-of-Oriented-Gradients-for-Given/d4881f8794acc300cce436a4d60f46d040a3af99/, Accessed: 2018-12-09.

[H+89] Lieve Hamers et al., *Similarity measures in scientometric research: The jaccard index versus salton's cosine formula.*, Information Processing and Management **25** (1989), no. 3, 315–18.

[HAGM14] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik, *Simultaneous detection and segmentation*, European Conference on Computer Vision, Springer, 2014, pp. 297–312.

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick, *Mask r-cnn*, Computer Vision (ICCV), 2017 IEEE International Conference on, IEEE, 2017, pp. 2980–2988.

[HRS+17] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al., *Speed/accuracy trade-offs for modern convolutional object detectors*, IEEE CVPR, vol. 4, 2017.

[Hui] Jonathan Hui, *map (mean average precision) for object detection.*, https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173/, Accessed: 2018-11-21.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep residual learning for image recognition*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[KB14] Diederik P Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980 (2014).

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, *Imagenet classification with deep convolutional neural networks*, Advances in neural information processing systems, 2012, pp. 1097–1105.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE **86** (1998), no. 11, 2278–2324.

[LBD+89] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural computation **1** (1989), no. 4, 541–551.

[LK15] Fei-Fei Li and Andrej Karpathy, *Convolutional neural networks for visual recognition*, 2015.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick, *Microsoft coco: Common objects in context*, European conference on computer vision, Springer, 2014, pp. 740–755.

[Low99] David G Lowe, *Object recognition from local scale-invariant features*, Computer vision, 1999. The proceedings of the seventh IEEE international conference on, vol. 2, Ieee, 1999, pp. 1150–1157.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell, *Fully convolutional networks for semantic segmentation*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3431–3440.

[MH80] David Marr and Ellen Hildreth, *Theory of edge detection*, Proc. R. Soc. Lond. B **207** (1980), no. 1167, 187–217.

[MP43] Warren S McCulloch and Walter Pitts, *A logical calculus of the ideas immanent in nervous activity*, The bulletin of mathematical biophysics **5** (1943), no. 4, 115–133.

[ON15] Keiron O'Shea and Ryan Nash, *An introduction to convolutional neural networks*, arXiv preprint arXiv:1511.08458 (2015).

[RHGS15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, Advances in neural information processing systems, 2015, pp. 91–99.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams, *Learning representations by back-propagating errors*, nature **323** (1986), no. 6088, 533.

[RHZS14] Shaoqing Ren, Kaiming He, Xiangyu Zhang, and Jian Sun, *Spatial pyramid pooling in deep convolutional networks for visual recognition*, European conference on computer vision, Springer, 2014, pp. 346–361.

[RL13] J Krause A Berg Russakovsky, J Deng and F Li, *Ilsvrc-2013*, URL http://www.imagenet.org/challenges/LSVRC/2013 (2013).

[Rud16] Sebastian Ruder, *An overview of gradient descent optimization algorithms*, arXiv preprint arXiv:1609.04747 (2016).

[SEZ$^+$13] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun, *Overfeat: Integrated recognition, localization and detection using convolutional networks*, arXiv preprint arXiv:1312.6229 (2013).

[SZ14] Karen Simonyan and Andrew Zisserman, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556 (2014).

[TH12] Tijmen Tieleman and Geoffrey Hinton, *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural networks for machine learning **4** (2012), no. 2, 26–31.

[UVDSGS13] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders, *Selective search for object recognition*, International journal of computer vision **104** (2013), no. 2, 154–171.

[VJS05] Paul Viola, Michael J Jones, and Daniel Snow, *Detecting pedestrians using patterns of motion and appearance*, International Journal of Computer Vision **63** (2005), no. 2, 153–161.

[ZD14] C Lawrence Zitnick and Piotr Dollár, *Edge boxes: Locating object proposals from edges*, European conference on computer vision, Springer, 2014, pp. 391–405.

Eidesstattliche Versicherung

Zafar, Muhammad Waleed	**0178994**
Name, Vorname	Matr.-Nr.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Masterarbeit mit dem Titel **"Object Detection and Segmentation using Region-based Deep Learning Architectures"** selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum	Unterschrift

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt und/oder eine falsche eidesstattliche Versicherung abgibt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -)
Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum	Unterschrift