# Visual Saliency Application in Object Detection for Search Space Reduction

by

Sai Rama Srivatsava Gorthy

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved August 2017 by the
Graduate Supervisory Committee:

Yu Cao, Chair
Jae-sun Seo
Sarma Vrudhula

ARIZONA STATE UNIVERSITY

December 2017

# ABSTRACT

Vision is the ability to see and interpret any visual stimulus. It is one of the most fundamental and complex tasks the brain performs. Its complexity can be understood from the fact that close to 50% of the human brain is dedicated to vision. The brain receives an overwhelming amount of sensory information from the retina – estimated at up to 100 Mbps per optic nerve. Parallel processing of the entire visual field in real time is likely impossible for even the most sophisticated brains due to the high computational complexity of the task [1]. Yet, organisms can efficiently process this information to parse complex scenes in real time. This amazing feat of nature relies on selective attention which allows the brain to filter sensory information to select only a small subset of it for further processing.

Today, Computer Vision has become ubiquitous in our society with several applications in image understanding, medicine, drones, self-driving cars and many more. With the advent of GPUs and the availability of huge datasets like ImageNet, Convolutional Neural Networks (CNNs) have come to play a very important role in solving computer vision tasks, e.g object detection. However, the size of the networks become prohibitive when higher accuracies are needed, which in turn demands more hardware. This hinders the application of CNNs to mobile platforms and stops them from hitting the real-time mark. The computational efficiency of a computer vision task, like object detection, can be enhanced by adopting a selective attention mechanism into the algorithm. In this work, this idea is explored by using Visual Proto Object Saliency algorithm [1] to crop out the areas of an image without relevant objects before a computationally intensive network like the Faster R-CNN [2] processes it.

ACKNOWLEDGMENTS

First and fore-most, I would like to thank my parents and my brother for their unwavering support throughout my master's education.

I would like to sincerely thank my professor Dr. Yu Cao for giving me the opportunity to work at NIMO and for providing guidance and support throughout my master's studies. Throughout my thesis project, Dr. Cao has always encouraged me to explore this relatively new field with his unwavering enthusiasm and high morale. I would like to extend my gratitude to Dr. Jae-sun Seo and Dr. Sarma Vrudhula for their invaluable inputs during the weekly meetings and for taking the time to serve as committee members. I am indebted to my colleague and friend, Abinash Mohanty, for his invaluable contribution, discussions and support throughout the course of this project. I would also like to thank my friends from NIMO: XiaoCong Du, Devyani Patra, Zihan Xu, Zheng Li and Deepak Vinayak Kadetotad, Minkyu from Dr. Jae-sun's group for their invaluable contributions, discussions, and support during the thesis work. I must also thank my graduate advisor Lynn Pratte for her active help with all the administrative procedures. Finally, I would like to thank Draper Labs and Johns Hopkins University for their collaborative research opportunity and their invaluable inputs for understanding the Visual Proto Object Saliency Algorithm.

TABLE OF CONTENTS

Page

LIST OF TABLES ..... v

LIST OF FIGURES ..... vi

CHAPTER

1. INTRODUCTION……1

1.1. Overview of Computer Vision……..1

1.2. Object Detection …….....3

1.3. Deep Learning in Computer Vision ……..4

1.4. Motivation……7

1.5. Thesis Organization……...……...9

2. OBJECT DETECTION APPROACHES……10

2.1. Region Based CNNs……10

2.1.1. R-CNN……...11

2.1.2. Fast R-CNN……13

2.1.3. Faster R-CNN……17

2.1.4. Summary of R-CNNs……19

2.2. Visual Proto Object Saliency……..20

2.3. Application of Saliency in Object Detection……...……23

3. REGION PROPOSALS USING SALIENCY……25

3.1. Modifications to the Saliency Algorithm……26

3.1.1. Grid Based Saliency Normalization……...26

3.1.2. Scale and Input Size……...……29

CHAPTER Page

3.2. Region Proposals with Saliency……..30

3.3. Implementation of Saliency with Faster R-CNN…….33

3.3.1. Tensor Flow……..…….....33

3.3.2. PASCAL VOC Dataset…….....33

3.3.3. Algorithm Implementation…….34

4. RESULTS & DISCUSSIONS……..36

5. CONCLUSION……41

REFERENCES ……....43

LIST OF TABLES

Table Page

4.1 Gt_coverage for Various IoG Threshold Values ……...……...38

4.2 Timing Results ……..……..38

LIST OF FIGURES

Figure Page

1.1 Challenges in Computer Vision……..……....1

1.2 Example of Object Segmentation…….……..2

1.3 Example of Object Detection……..……3

1.4 Classification Using CNNs……...……..5

1.5 ILSVRC Top 5 Error Rate……..6

1.6: Example Output of Visual Proto Object Saliency……..8

2.1: Performance Trend of Algorithms on PASCAL VOC……...10

2.2: Slow R-CNN Algorithm……...……...……...……12

2.3: Region Proposal Warping in Slow R-CNN……..……..13

2.4: Region Proposal Mapping onto Conv Features: Fast R-CNN…….……...14

2.5: RoI Pooling to a Fixed Length Feature Vector……...…15

2.6: Fast R-CNN Architecture (Test Time)……...……16

2.7: Faster R-CNN Architecture……...……...…… 17

2.8: Faster R-CNN: Anchor Boxes and Region Proposal Generation…….…...……...18

2.9: Distribution of Computation Times in Faster R-CNN……..……..19

2.10(a): Output of Visual Proto Object Saliency Algorithm…….....……...21

2.10(b): Output of Other Feature Based Algorithm……...……...21

2.11: Visual Proto Object Saliency Algorithm……...……...……23

3.1: Default Normalization of the Saliency Map……...…….26

3.2: Grid Based Normalization…….……..28

Figure Page

3.3: Saliency Output at Different No of Scales in the Algorithm……..……...……..29

3.4: Sample Image with Ground Truth Boxes…….……...……...31

3.5 Steps in the Proposed Approach……..……31

3.6: Examples of Saliency Region Proposals…….……...32

3.7: Classes in PASCAL VOC Dataset……...…….....34

4.1: Example Image for Area Saving Calculation…….……36

4.2: Examples of Coarse Region Proposals…….……..39

4.3: Issues in the Current Approach……..……39

# CHAPTER 1. INTRODUCTION

## *1.1.* Overview of Computer Vision

Computer Vision is the science of teaching computers to interpret visual stimulus like an image or a video. In simpler terms, it associates a meaningful interpretation with pixels, similar to how organisms see and interact with their surroundings. It could be labeling the image of a dog, detecting a pedestrian on the street, or recognizing a hand gesture. Computer vision problems in a realistic setting like detecting pedestrians, are very hard to solve. Figure 1 shows a few challenges in computer vision that make it the hard problem it is.

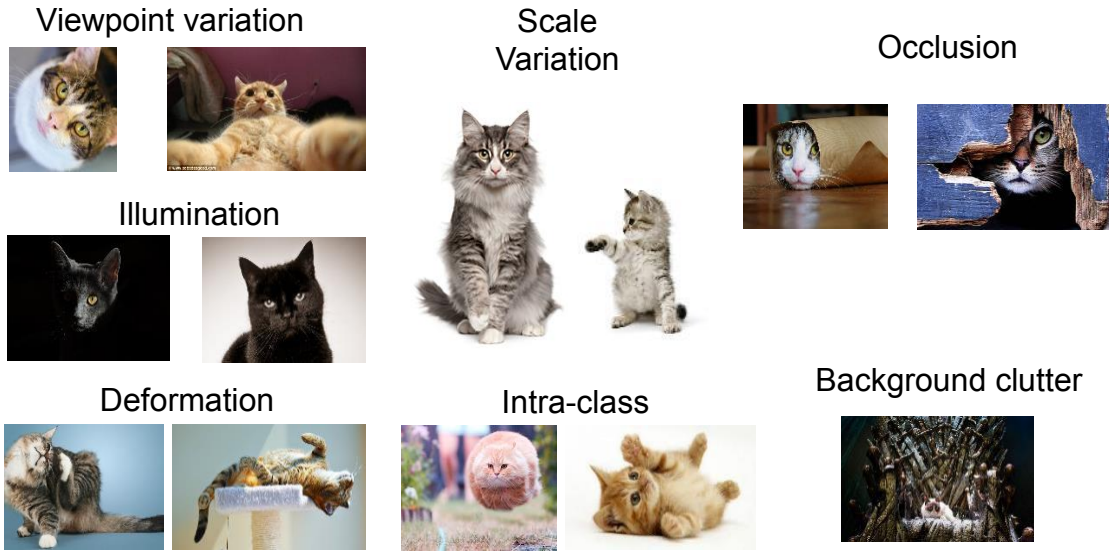



*Figure 1.1 Challenges in Computer Vision [3]*

As it can be seen, to classify an image, there are innumerous possibilities that need to be considered. A single instance of an object can be oriented in many ways with respect to the camera. The instance of an object in an image could be small or large leading to scale

variations requiring the computer vision algorithm to be scale invariant. Many objects of interest, like cats, are not rigid bodies and can be deformed in extreme ways, having different postures. The objects of interest can be occluded, making only a portion of the object visible or the object could be blended into the background like a black cat on a black couch. Needless to say, objects of the same class do not necessarily look the same, like different breeds of dogs, which have their own appearance, but are still the same class of objects.

Classification, detection and segmentation are the key problems in computer vision. Classification is the process of assigning a class to an input image. Object detection, as will be discussed in the next section, involves localization of an object in an image. Segmentation is a more complex problem, where a label is assigned to each pixel of an image, resulting in precise boundaries for each object. An example of object segmentation from the PASCAL VOC dataset [4] is shown in Fig 1.2

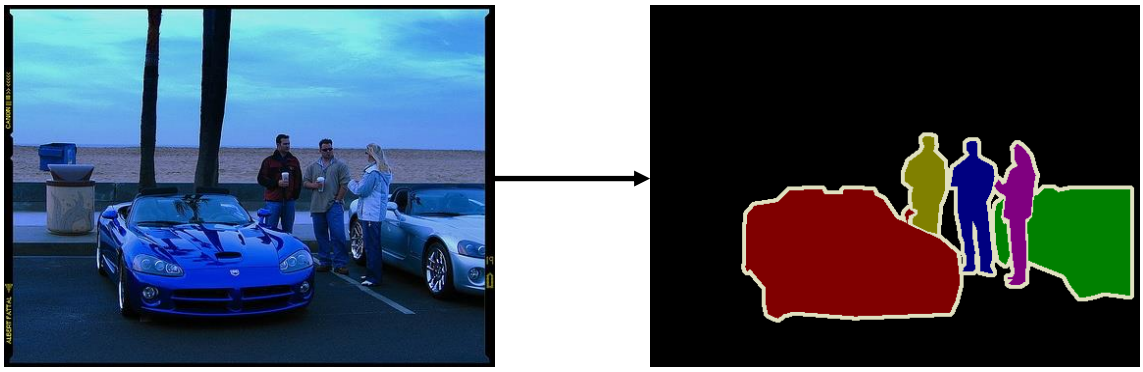


*Figure 1.2 Example of Object Segmentation [4]*

## *1.2.* **Object Detection**

Object detection is the process of localizing different objects in an image and classifying them. In simple words, it answers two important questions about an object: *what? and where?* It is one of the most fundamental tasks of vision. Figure 1 is a demonstration of object detection. It can be seen in the image that objects of different classes are labeled and localized.

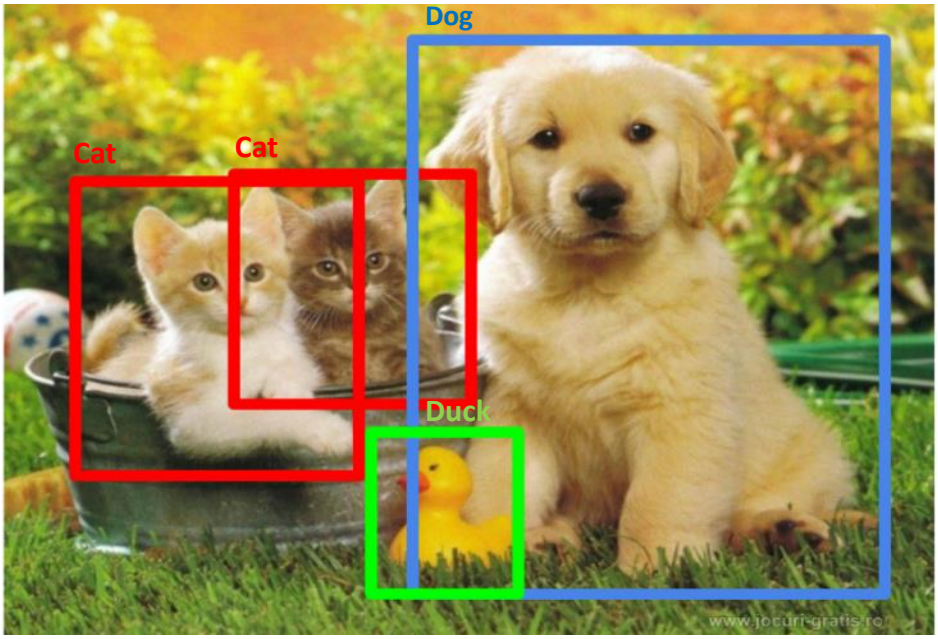



*Figure 1.3 Example of Object Detection [3]*

Object detection has several applications, some applications include: face detection – it is essentially a subset of object detection, autonomous driving – to recognize objects like sign boards and pedestrians and other obstacles in a driving environment to make the right decision. Medicine, security, manufacturing, robotics are amongst several other industries that benefit from object detection algorithms and computer vision in general.

In recent times, Convolutional neural networks(CNNs) have become very feasible to use, primarily because of advances in GPU technology and availability of datasets. As a result, several deep learning based object detection algorithms like the region based CNNs: R-CNN [6], Fast R-CNN [5], Faster R-CNN [2] have achieved a mAP(accuracy) over 60% on the PASCAL VOC challenge [4]. Some of these algorithms will be discussed in chapter 2 in detail.

### *1.3.* **Deep Learning in Computer Vision**

CNNs, like regular fully connected neural networks, are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. CNN architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the number of parameters in the network.

A typical CNN has an input layer, which holds the raw pixel values of the image. A *convolution layer* computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. The *ReLU layer* applies an elementwise activation function, such as the max(0,x) thresholding at zero. This leaves the size of the volume unchanged. A *pooling layer* performs a down-sampling operation along the spatial dimensions, resulting in a shrink in the volume of the feature maps. The *fully-connected layer* computes the class scores, which usually is a vector with as many elements as the number of classes in the dataset. Each number would represent a class score [3]. In this

way, CNNs transform the original image layer by layer from the original pixel values to the final class scores. Fig 1.4 shows the series of steps in a typical CNN that perform this transformation of pixels to scores.

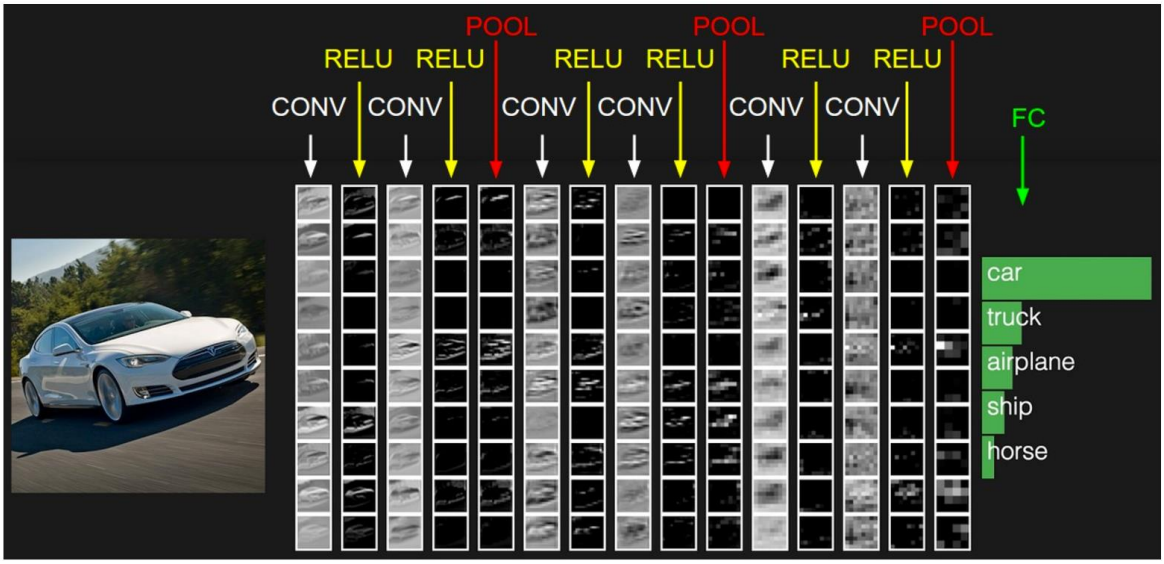



*Figure 1.4 Classification using CNNs [3]*

Convolutional Neural networks have been used to solve computer vision problems since the late 90s[8]. However, they were limited to lesser challenging tasks like handwritten digit recognition. The prohibitive amount of time to train and limited availability of data sets, compute resources, limited the scalability of neural networks to more complex vision tasks, with inputs of a respectable size, say 227x227. But the recent developments in the VLSI industry, powerful GPUs from companies like NVidia came into the market. Also, in 2009 a huge dataset called ImageNet with millions of images and 1000 classes, was introduced by Dr. Fei-Fei Li et al [7]. This quickly developed into an annual competition called ILSVRC (ImageNet Large Scale Visual Recognition Challenge). The introduction of this dataset, in many ways, is indeed the reason behind the current developments in deep learning. ImageNet opened up the arena for deep neural networks

and aided the development of better algorithms, by providing a huge database of images to train. In the year 2012, AlexNET [10] truly demonstrated how the combination of GPU computation and large datasets like ImageNet, could be used to apply deep neural networks to vision problems.

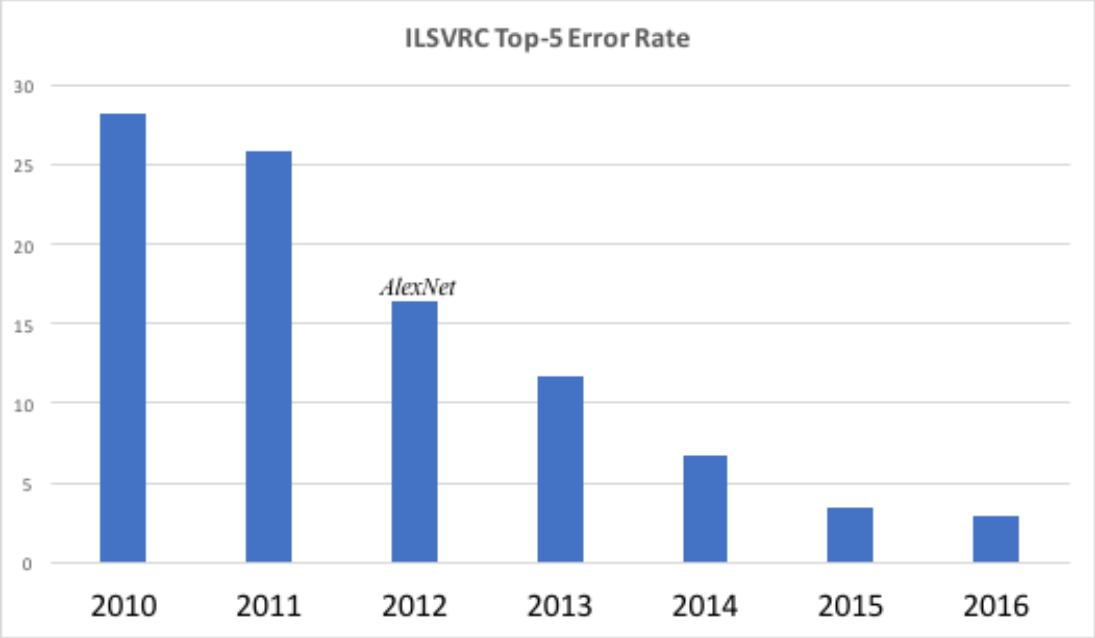



*Figure 1.5 ILSVRC Top 5 Error Rate [9]*

Fig 1.2 shows the trend of top 5 error rate on the ILSVRC since the introduction of ImageNet in 2009. It can be clearly seen that AlexNet, a deep neural network from University of Toronto, in the year 2012 brought a drastic dip in the top 5 error rate with 15.3% [10] which is substantially lower than the previous year ~ 26%. In the following years, several other neural network based approaches like ZF Net, VGG Net, Google Net, ResNet etc. achieved the best results at ILSVRC.

It is worth noting that to get higher accuracy the complexity of the CNNs and the computational intensity increases rapidly. For example, the VGG Net [11] has 140M

parameters when compared to the 60M parameters of AlexNet [10]. Also, it is clear from this trend that the capacity of achieving higher accuracies comes at a cost of having a deeper network with more number of layers and parameters, which implies increased computational complexity. So, in order to practically achieve high accuracy on a mobile platform, it is important to make these algorithms more computationally efficient.

### *1.4.* **Motivation**

In the previous section, a brief account of the evolution of deep neural networks as powerful tools to solve computer vision problems has been presented. It is also clear that transferring these algorithms to mobile platforms or embedded systems, within the power envelope of the embedded system, is the next step in the grand evolution of CNNs. Applications like autonomous driving, use deep learning algorithms like Faster R-CNN and SSD (Single Shot Detector) for object detection. As such, the computation speed and portability of these algorithms is of paramount importance in such applications.

Also, we have seen that vision is a very hard problem and is among the most important sensory systems in the human brain. About 50% of the human brain is dedicated to the visual cortex [3]. These resources are used to process an overwhelming amount of information that is received, about 100Mbps per optic nerve [1]. Despite receiving an overwhelming amount of data, it yet manages to achieve real-time performance in vision by employing selective attention schemes, which provide a mechanism to process only a certain relevant subset of the information.

An analysis on the PASCAL VOC dataset tells that only about 52.5% of the total image area on an average is the ground truth. The rest 47.5% contributes to computations

that would not affect the final result of the network, which is essentially wasted computation. An approach that would effectively eliminate the data that don't contribute to the final result, would be very similar to the selective attention mechanism in the brain. The Visual Proto Object Saliency is an algorithm that brings out the most prominent areas in a scene by suppressing the background. Fig 1.6, shows a sample output of the algorithm.

This work explores using the visual proto object saliency algorithm as a pre-processing step before Faster R-CNN, which is a VGG based object detection framework.

A detailed explanation of the approach can be found in chapter 3.

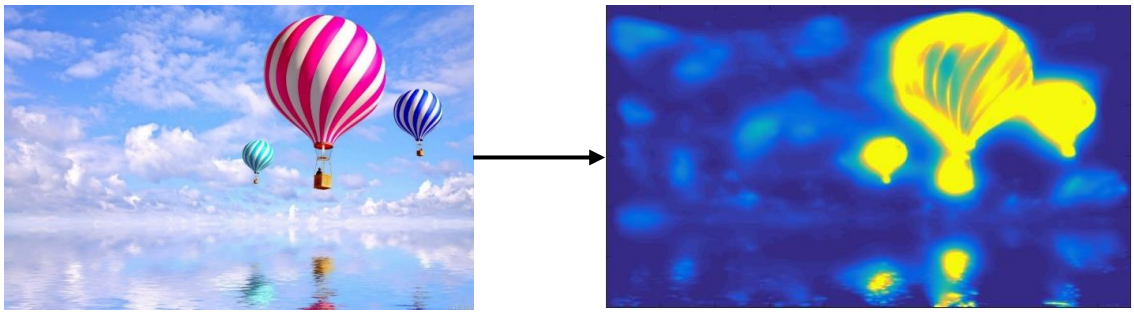


*Figure 1.6: Example output of Visual Proto Object Saliency*

## *1.5.* **Thesis Organization**

Chapter 2 discusses region based CNNs or R-CNNs, which include Faster R-CNN the algorithm used in this work. Also, the visual proto object saliency is introduced and its application in object detection is briefly discussed in this chapter. Chapter 3 discusses the implementation of the Visual Proto Object Saliency algorithm, acceleration of the algorithm using TensorFlow, PASCAL VOC dataset, proposed approach of using saliency as a pre-processing step for Faster R-CNN. Chapter 4 presents the key results and provides a discussion on the results. Chapter 5 summarizes this work and sheds light on some possible future work ideas.

# CHAPTER 2. OBJECT DETECTION APPROACHES

## *2.1.* Region Based CNNs

As stated in Girshick et al.[6], object detection performance, as measured on the PASCAL VOC dataset, had plateaued in the last few years before the region based CNNs. R-CNN is a scalable detection algorithm that improved mean average precision (mAP) by more than 30% relative to the previous best result on VOC 2012—achieving a mAP of 53.3%. The idea behind region based CNNs is to apply high-capacity CNNs to bottom-up region proposals in order to localize and segment objects. Also, the methodology of supervised pre-training for an auxiliary task, followed by domain-specific fine-tuning was introduced. This is called transfer learning, i.e. a deep CNN is trained on a large database like ImageNet and the last few layers are fine-tuned based on the application. Based on this idea of combining regions with CNNs, Ross Girshick et al. proposed a series of region based CNNs which will be discussed in this chapter.

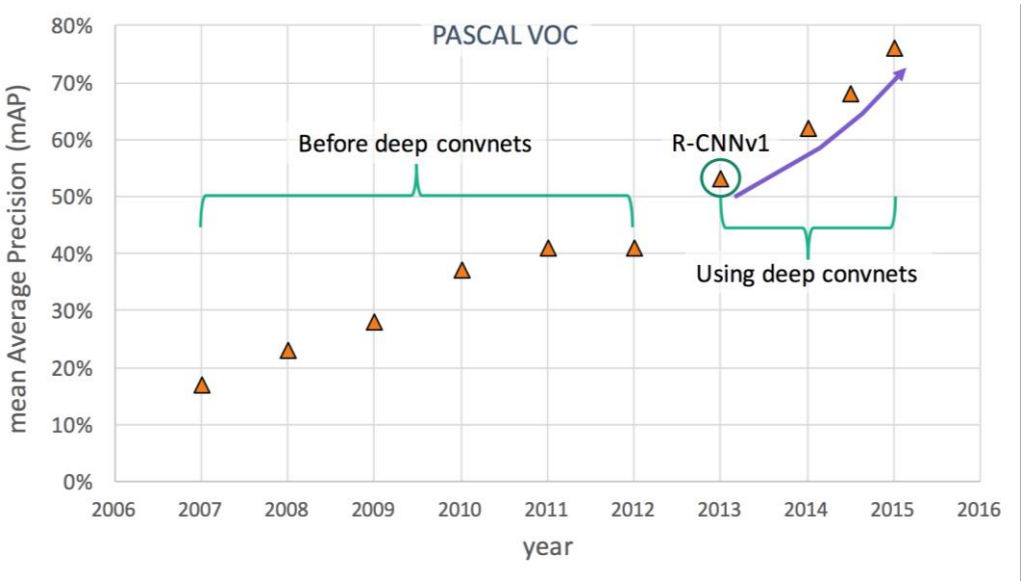



Image credit: Ross Girshick

*Figure 2.1: Performance trend of algorithms on PASCAL VOC*

Figure 2.1, shows the trend of the performance of algorithms on the PASCAL VOC object detection challenge. Clearly R-CNN, a region based CNN method, has shown unprecedented improvement in the performance in 2013, which otherwise looks to have plateaued with the SIFT and HOG methods. Its successors, like Fast R-CNN, Faster R-CNN have achieved better results. In the following subsections, a summary of each of these approaches has been presented**.**

### *2.1.1.* **R-CNN**

Region based CNN also known as Slow R-CNN [6], is a deep learning based object detection algorithm. Inspired from the ground-breaking performance of AlexNet at ILSVRC 2012, Ross Girshick et al. proposed the R-CNN in an attempt to generalize the good results of CNNs on classification to object detection. The aim of R-CNN is to take an input image, a set of pre-computed region proposals and identify where the main objects in the image are.

The algorithm takes as input, an image and a fixed set of externally generated region proposals (about two thousand). Each of these proposals is warped to process through a pre-trained ConvNet, like AlexNet. At the end of the ConvNet or the CNN, there is a classification head and a regression head. The feature vector generated at the end of the ConvNet is given as an input to the classification head (SVMs), to classify the regions and to the bounding box regressors, to output tighter co-ordinates, which indicate the object position in the image. Figure 2.2 shows the data flow in the algorithm.

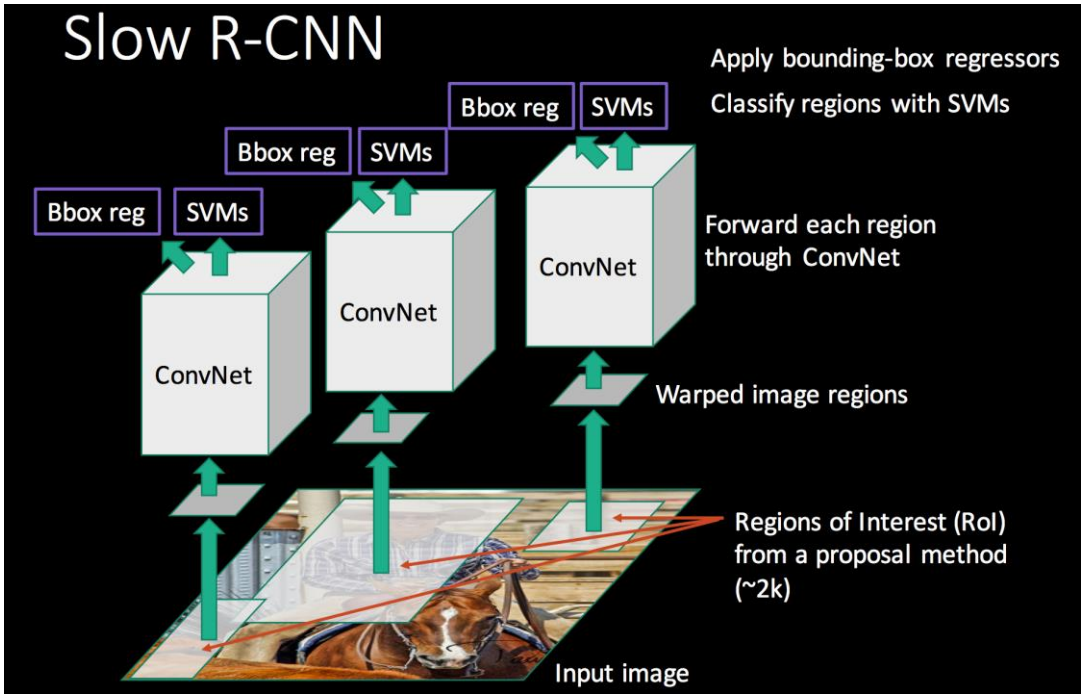



Image credit: Ross Girshick

*Figure 2.2: Slow R-CNN algorithm*

However, despite having an excellent mAP of 66% on PASCAL VOC07, R-CNN had several drawbacks like: training was a complex multi-stage pipeline where, the CNN to generate features, bounding box regressors for localization and classifiers for object classification are trained in separate stages of training, with different objective or cost functions. Also, for very deep networks like VGG16, it took 2.5 GPU days for training it with the 5k images from VOC07 trainval set, on a Nvidia K40 GPU. The features extracted from each region proposal were cached in a hard-drive, demanding huge amounts of memory (about 200 GB for PASCAL VOC). Another major limitation of R-CNN was the test time. It took 47s per image as it performs a forward pass in the CNN for each object proposal, without sharing any computation [5], which is about 2000 forward passes per image.

### *2.1.2.* **Fast R-CNN**

Addressing the limitations of the slow R-CNN, Ross Girshick proposed the Fast R-CNN [5], which had a single stage training methodology with a multi-task loss. Clearly, a lot of proposed regions for the image invariably overlapped, making the multiple forward CNN passes redundant. The core idea behind Fast R-CNN is to run the CNN just once, rather than as many times as the number of proposals, and share that computation across all the proposals. This computation sharing in the algorithm is achieved by using the *RoI (Region of Interest) pooling* technique.

***RoI Pooling:*** A clear overview of the RoI pooling technique is vital to understanding the efficiency and advantages of Fast and Faster R-CNNs. From the discussion on R-CNN, it is known that the region proposals are warped (resized) to fixed dimensions before the CNN forward pass. This is done to make sure that the features extracted from the conv and pool layers, are of a size that is compatible with the fully connected layer input size. The fully connected layer inputs have a restriction on the input size (same as the no of neurons), unlike the conv layers. Figure 2.3 shows how region proposals in an image are warped before a CNN forward pass in the slow R-CNN.

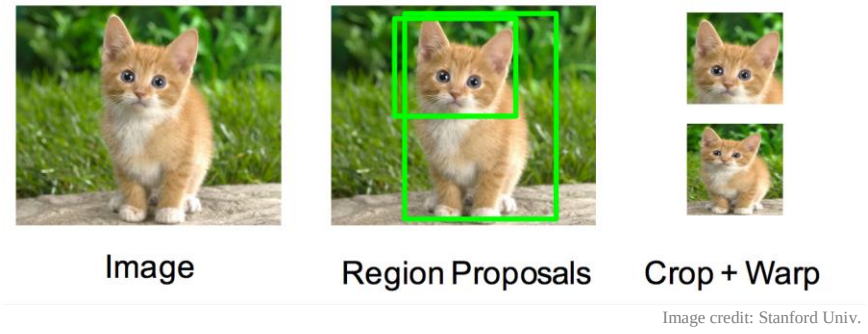



*Figure 2.3: Region Proposal warping in slow R-CNN*

This limitation on the input size was eliminated by the RoI pooling technique. If the layers that appear before the fully-connected layers (conv, pool, ReLU) are considered as the feature extracting trunk of a CNN, RoI pooling steps in after this trunk, maps a region proposal onto the features extracted from this trunk, and performs pooling from that region of the feature map to make its size compatible to the input size of the fully-connected layer. A very important outcome of this technique, as shown in figure 2.4, is that, the input image can now be of any size (high res). Hence, it is essentially like a bridge between, the trunk and the fully-connected layers. Figure 2.4, 2.5 demonstrate the idea of RoI pooling.

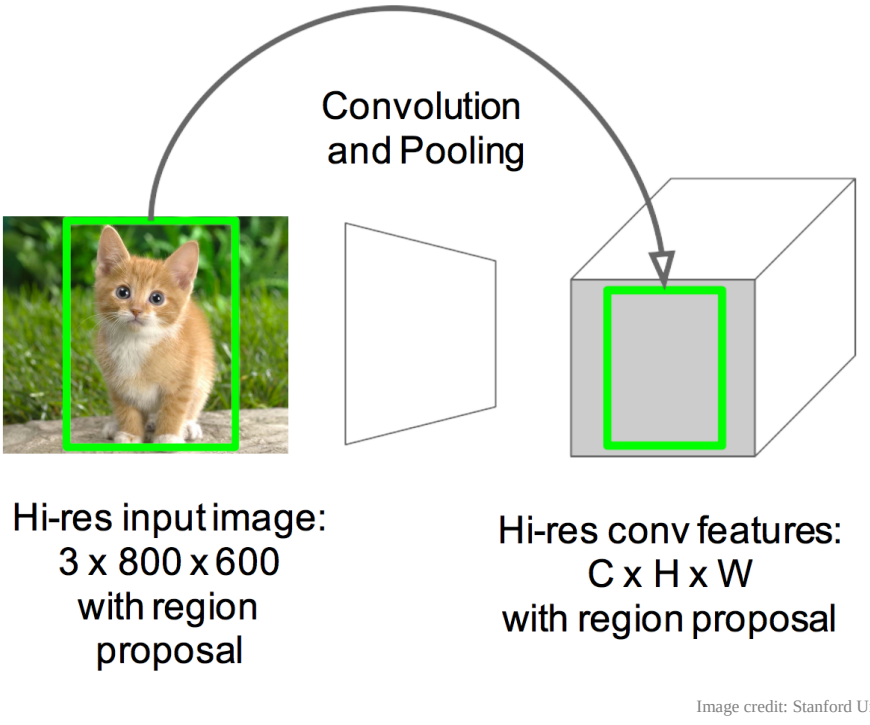



*Figure 2.4: Region Proposal mapping onto conv features: Fast R-CNN*

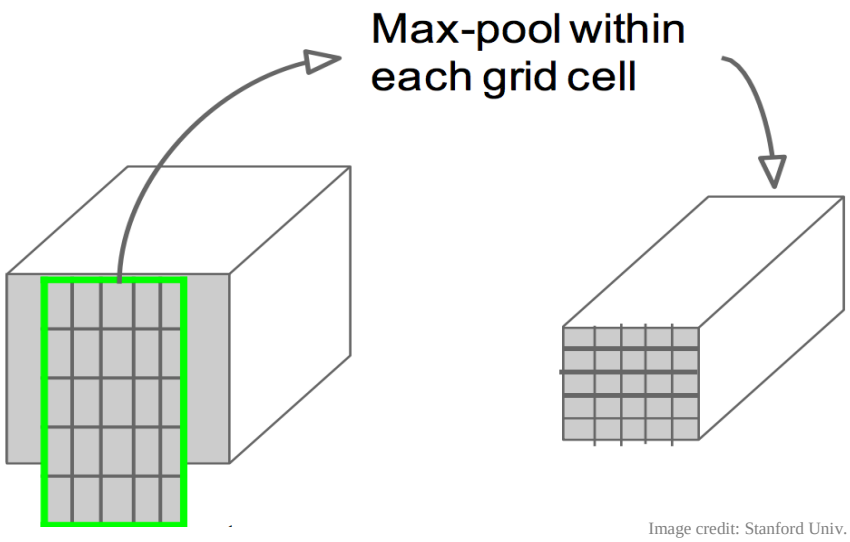



*Figure 2.5: RoI pooling to a fixed length feature vector*

It can be seen in figure 2.4, 2.5 that a region proposal is mapped onto the feature map, which is extracted by the trunk, and pooling is applied to get a fixed size feature map. This is the central idea of Fast R-CNN that enabled sharing of CNN computations and sped up the network during test time. Also, a mAP of 66.9%, which is slightly higher than the R-CNN was reported on VOC07. From the discussion on slow R-CNN, it is known that the test time per image was 47s, for Fast R-CNN the test time is just 0.32s. If the time taken by selective search, to generate region proposals is also considered, slow R-CNN takes ~ 50s and Fast R-CNN takes 2s per image. Clearly, this affected the performance of Fast R-CNN. The algorithm by itself was not the performance bottleneck anymore, it was the region proposal methodology which hindered performance. This problem is addressed in another algorithm called the Faster R-CNN, discussed in the next sub-section.

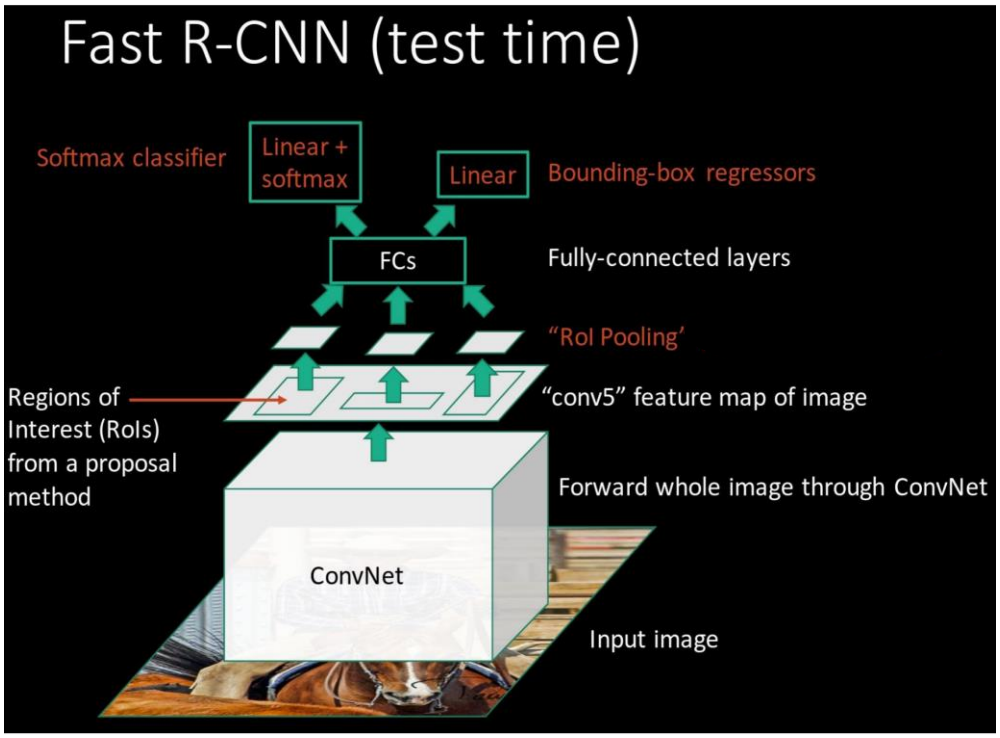



Image credit: Ross Girshick

*Figure 2.6: Fast R-CNN architecture (test time)*

Figure 2.6 shows the architecture of Fast R-CNN as discussed in this section. The ConvNet corresponds to the trunk, conv5 features are the output of the trunk for a VGG-16 architecture.

### *2.1.3.* **Faster R-CNN**

As discussed above, even with the RoI pooling layer, the test time speed of Fast R-CNN was limited by the external region proposal algorithm like, selective search. To address this problem, Ren et al. proposed the Faster R-CNN [2]. Region proposals depended on features of the input image that were already calculated with the forward pass of the CNN. The idea behind Faster R-CNN was to exploit this dependence and generate region proposals internally, within the detection network. This was achieved by having a *Region Proposal Network* (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. As a result, generation of region proposals was no longer external to the detection network.

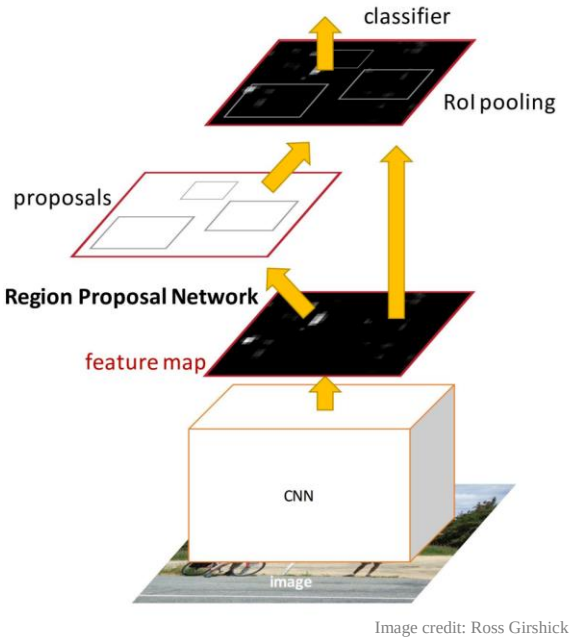



*Figure 2.7: Faster R-CNN architecture*

An RPN is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. The RPN is integrated into the Fast R-CNN network as shown in figure 2.7 to generate region proposals, while sharing computation with the CNN. The generated proposals are processed through an RoI pooling layer just like the Fast R-CNN.

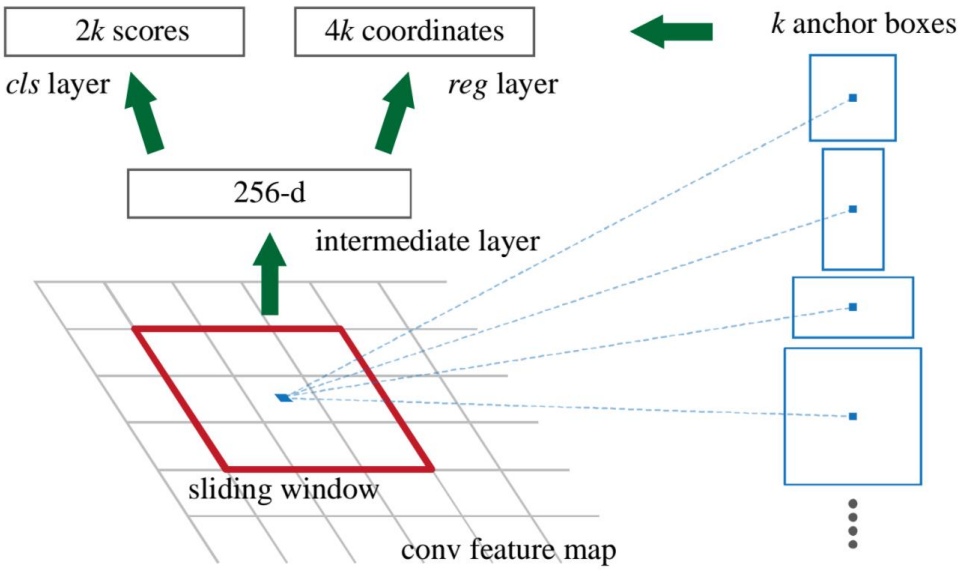



*Figure 2.8: Faster R-CNN: Anchor boxes and region proposal generation [2]*

The RPN, which is a fully convolutional neural network, operates in a sliding window fashion on the features generated from the conv layers of the detection network (e.g VGG-16). For each sliding window position, *k* boxes of different aspect ratios are proposed. It can be seen in figure 2.8, how each of the *k* proposed anchor boxes differs in aspect ratios. Each box, also has an objectness score and a set of bounding box co-ordinates, associated with it. Objectness score really is a metric or probability of the box being an

object or the background. It should be noted that this score is not the actual classification probability. It is a class agnostic, binary probability of whether the box has an object or not. It can be noticed that for *k* boxes, the no. of classification outputs is 2*k*, this is because the classifier outputs *p*, *(1-p)* and the 4*k* outputs for the bounding boxes are the four parameters used to represent *k* boxes. These proposals are then processed through a NMS (non-maximum suppression) and the resultant proposals with a set threshold of objectness score, are passed on to the RoI pooling layer. The rest of the algorithm works just like Fast R-CNN.

### *2.1.4.* **Summary of R-CNNs**

It is clear that by sharing computations, Faster R-CNN could do away with the region proposal algorithms and have a test time of just 0.2 seconds. The Faster R-CNN also had a mAP of 66.9%, the same as Fast R-CNN. Figure 2.9 [2] shows how the distribution of computation times across various operations. The convolutional layers take 141ms, proposal takes 10ms, region wise operations like NMS, fc, softmax take 47ms.

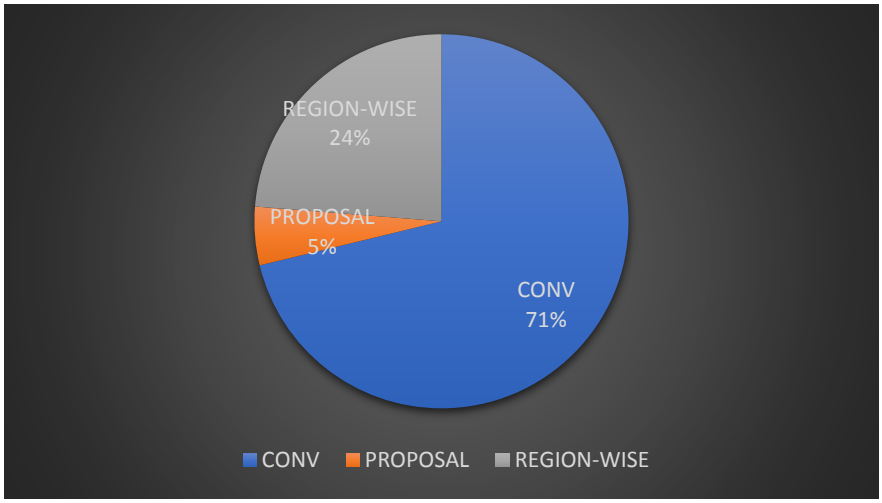



*Figure 2.9: Distribution of computation times in Faster R-CNN*

Clearly, a reduction in the computations involved in the conv layers can speed up Faster R-CNN significantly. As discussed in section 1.4, we explore in this work, a method of reducing the compute intensity of the conv layers, using the visual proto object saliency algorithm. In the next section, the saliency algorithm will be introduced.

## *2.2.* **Visual Proto Object Saliency**

A brief discussion on the Visual Proto Object Saliency algorithm [1], which will be interchangeably, referred to as *saliency algorithm*, was presented in section 1.4. The algorithm was proposed by Russell et al. from Johns Hopkins University. In this section, the key aspects of saliency algorithm will be discussed.

The ability of an organism to process relevant information in a scene in real time, comes from two different mechanisms working together. The first, top down attention, is controlled by the organism itself and biases attention based on the organism's internal state and goals. The second mechanism, bottom up attention, is based on different parts of a visual scene having different instantaneous saliency values [1]. The saliency algorithm only deals with the second mechanism which is the objective analysis of a scene.

Organisms use the process of selective attention to optimally allocate their computational resources to the instantaneously most relevant subsets of a visual scene, ensuring that they can parse the scene in *real time* [1]. Gestalt psychologists, argue that humans perceive whole objects before they analyze individual features. Supporting this philosophy, the saliency algorithm computes saliency as a function of proto objects which make up a scene and each step has direct neural correlates.

In an experiment from [12], subjects were asked to identify the color of a target element, from a bunch of 'L' shaped objects in the display. It was observed that, reaction times were the fastest when the target element was part of an object, slowest when the target was outside the object and intermediate when no objects were present in the scene. The proto object saliency has a very similar behavior, as seen in figure 2.10(a), 2.10(b). It is clear that the algorithm outputs saliency by giving more priority to objects in the scene, unlike other approaches, which are feature based.

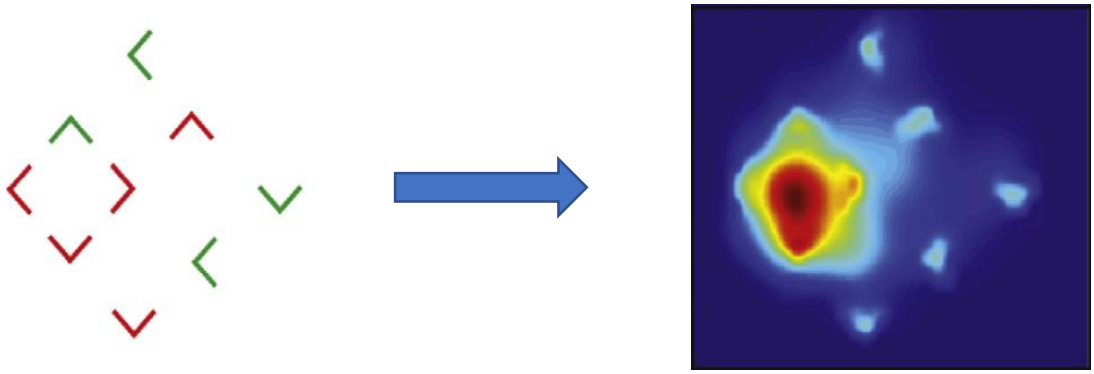


*Figure 2.10(a): Output of Visual Proto Object Saliency algorithm [1]*

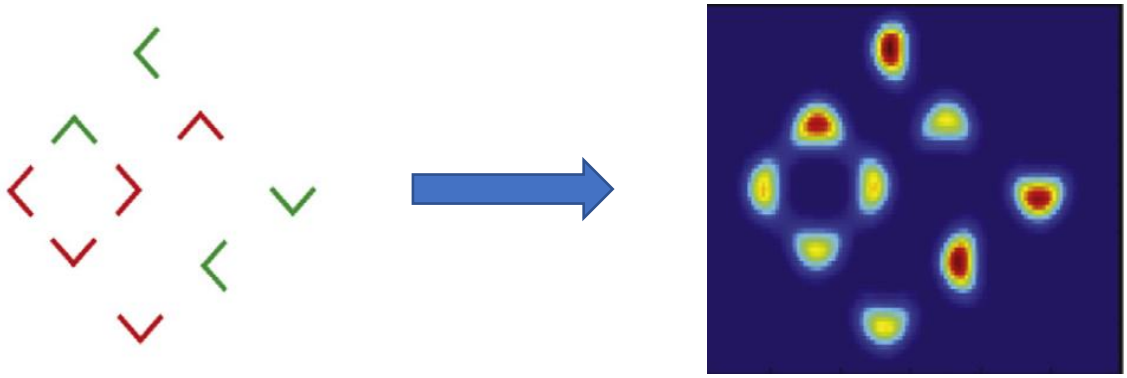


*Figure 2.10(b): Output of other feature based algorithm [1]*

In the saliency algorithm, the input image is split into 9 channels: *One intensity channel, four color opponency channels and four orientation channels,* each having 10 scales. This image pyramid of 10 scales spans 5 octaves, helping achieve scale invariance. The first stage of processing extracts object edges using 2D Gabor filters [13], which approximate the receptive fields of simple cells in the primary visual cortex [1]. The even and odd components of the simple cell response are combined to get the edge information in an image by the complex cells. To infer whether the edges in the complex cell response belong to figure or ground, information about the objects in the scene is required [1]. This context information is obtained from a center surround mechanism. The border-ownership mechanism combines the information from complex cells and center-surround mechanism to assign the edges in the image to either object or ground. The final stage of the algorithm calculates grouping cell responses by integrating the winning border-ownership cell activity. The grouping activity calculation is the stage of the algorithm that enforces the gestalt principles of object based saliency. The grouping responses from each channel are combined to form the final proto-object saliency map. Figure 2.11 shows the data flow in the saliency algorithm.

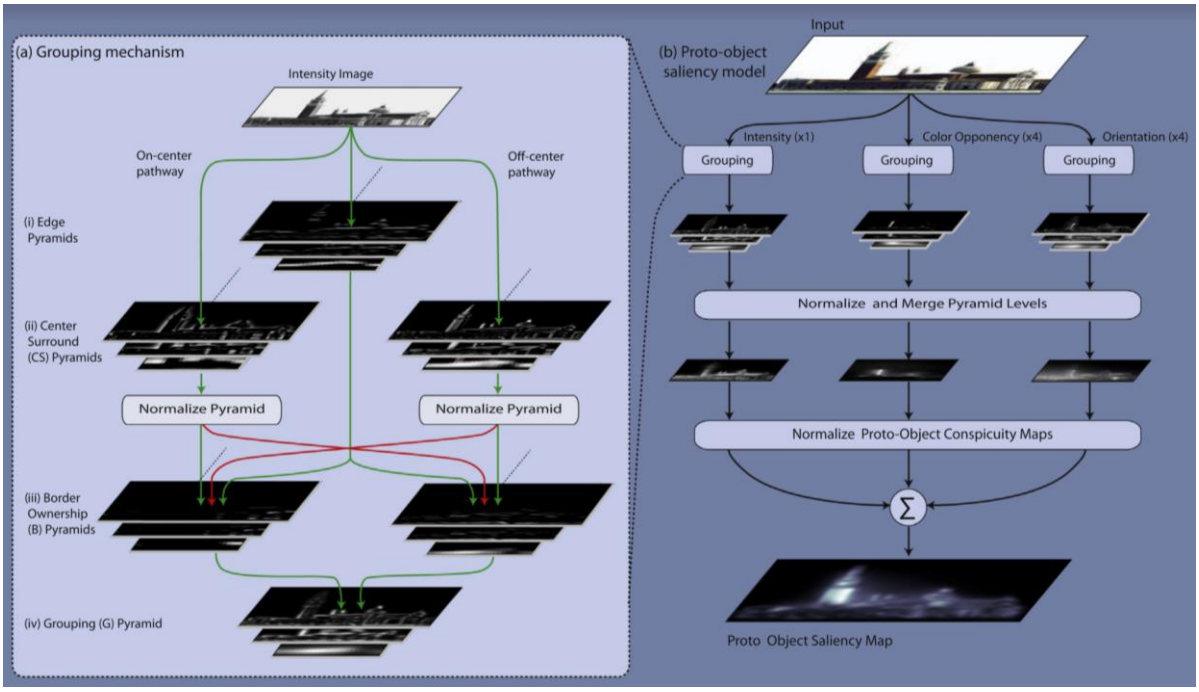



*Figure 2.11: Visual Proto Object Saliency algorithm [1]*

## *2.3.* **Application of Saliency in Object Detection**

This attribute of being biased to the objects in a scene and being class agnostic, make saliency a potential tool for object detection. This algorithm, as it can be seen, is completely unsupervised and hence, doesn't have to go through the computationally expensive procedure of training a deep CNN and it does not require huge datasets of images either.

As discussed in chapter 1, an analysis on PASCAL VOC07 tells us that only 52.5 % of the images are actually the ground truths. In chapter 2, we have seen how the convolution step of the R-CNNs is a performance bottleneck in object detection and also how saliency can help these algorithms, similar to how attention mechanisms help

organisms perform vision tasks very efficiently. Thus, in order to fit the high-performance object detection algorithms, within the power envelope and resources of a mobile platform, it is required that the algorithms are made more compute efficient, thereby reducing the power consumption.

So, in this work, a method for using saliency into the Faster R-CNN framework is proposed. A detailed description of the proposed idea can be found in the next chapter.

# CHAPTER 3. REGION PROPOSALS USING SALIENCY

It is clear from the discussion in chapter 2, that visual proto-object saliency is a computationally expensive algorithm, as it has a series of convolution, resizing and normalization operations over 10 scales and across 9 feature channels. So, in order to use saliency as a pre-processing step, for creating coarse region proposals by cropping out most of the background, before Faster R-CNN, it is important that the computation overhead created by the saliency algorithm is lesser than the number of computations saved in the VGG-16 based, Faster R-CNN. A coarse region proposal in this context means a region of the image that would have the ground truth and also any other regions of the image that do not necessarily fall into the ground truth. In some cases, if the object is big, it can be the entire image.

In order to meet the computational overhead criterion, the saliency algorithm has been modified, while still preserving the proto-object approach to computing saliency. The modified version of saliency algorithm has been implemented in TensorFlow for GPU support. All the timing numbers reported in this chapter are using the TensorFlow (*see section 3.3.1 for more details on TensorFlow*) platform and NVidia TitanX GPU. Also, the dataset used for evaluation of this approach is PASCAL VOC07 (*see section 3.3.2 for more information on the dataset*). The area results reported in this chapter are based on images from PASCAL VOC07. A description of the modifications to saliency, proposed flow for generating coarse region proposals and its implementation, will be seen in the subsequent sections of this chapter.

## *3.1.* **Modifications to the Saliency Algorithm**

### *3.1.1.* **Grid Based Saliency Normalization**

The output of the grouping stage, which is the step before normalization, is normalized to get an output saliency map by dividing every pixel intensity with the maximum value in the map.

Eqn 3.1 shows the default normalization in the saliency algorithm, $\Omega(x,y)$ is the intensity value of a pixel at $(x,y)$ in the non-normalized saliency map and ß is the maximum value of Ω. μ is the normalized saliency map. Figure 3.1 shows the result of this type of normalization (image on the left and normalized map on the right).

$$\mu(x,y) = \frac{\Omega(x,y)}{ß} \qquad \textit{(Eqn. 3.1)}$$

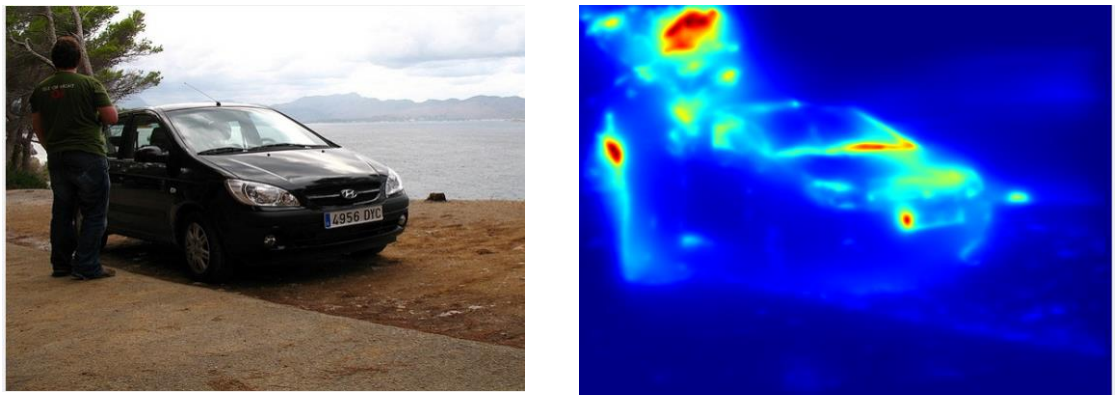


*Figure 3.1: Default normalization of the saliency map*

As it can be seen not many pixels from the object are above a certain threshold. If a thresholding mechanism is used to generate bounding boxes, this approach might completely miss the objects. Instead of doing a global maximum normalization, a more local approach of normalizing the saliency map gives better results. This can be called the

grid based normalization. The idea is to divide the entire saliency map into a certain number of grids and normalize each grid with respect to its local maximum. Eqn 3.2 shows the proposed normalization technique. Ω is the intensity value of a pixel at a given *(x,y)* in the non-normalized saliency map, *M* is the local maximum of the area under consideration, ß is the global maximum (meaning max value of Ω) and ø is an empirical value that controls the influence of the global maximum on the final pixel value.

$$\mu(x,y) = \frac{\Omega(x,y)}{max(M,ß \div ø)} \qquad (Eqn\ 3.2)$$

Figure 3.2 shows examples of grid based normalization on the same image for different grid sizes.

As it can be seen more pixels inside the object (car and a person in this case) are highlighted when grid based nrmalization is used.

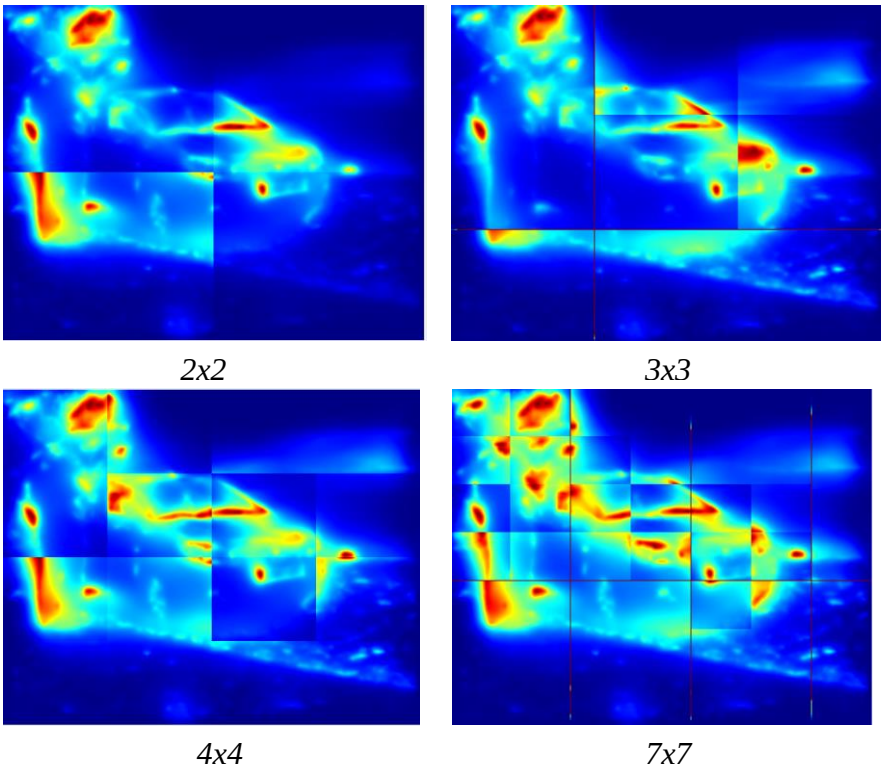



*Figure 3.2: Grid Based Normalization*

Clearly grid-based normalization helps produce more meaningful saliency maps from coarse region proposal perspective. This style of normalization, focuses more on the pixels nearby to a pixel under consideration. By doing this, the influence of just one pixel on the entire map can be reduced, bringing out more local information about the objects in the scene. It is to be noted that the image shown here has been taken from the PASCAL VOC07 dataset.

### *3.1.2.* **Scale and Input size**

The average test time of Faster R-CNN on titanX GPU is measured to be 168 ms. The matlab CPU implementation of saliency took over 30s per image of PASCAL VOC07 dataset. The GPU implementation of saliency with a regular sized image (as-is from the dataset) and ten scales, took over 300ms to run on titanX GPU, which is, clearly, much higher than the Faster R-CNN. However, a full-sized image would give a detailed saliency map, which is not necessary for its application as a coarse region proposer. In this application, only a tentative location of the object is required and the information about the features can be ignored as it will be dealt with in the detection CNN in a later stage.

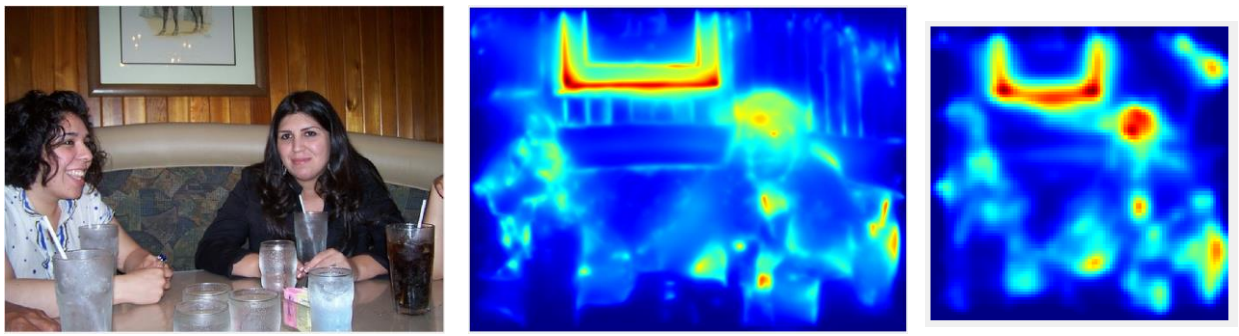


*Image* *10 scale output* *2 scale map*

*Figure 3.3: Saliency output at different no of scales in the algorithm*

Figure 3.3 demonstrates that a very similar saliency map can be obtained with just 2 scales when using an input of size 100x100. With this observation, a version of saliency with 2 scales and an input size of 150x150 was implemented in TensorFlow. This version of the algorithm took just 24 ms per image to compute saliency. Also, a version with an input size of 100x100 was implemented which took just 15 ms.

### *3.2.* **Region proposals with saliency**

A modified version of the saliency algorithm, as discussed above, with 100x100 input size, 2 scales and with normalization parameters: 4x4 grid size, ø = 2 is used to propose coarse grain regions. The motivation behind these modifications is to speed up the saliency algorithm and not lose any essential information from the saliency map. In this section, we'll see how the saliency algorithm is used to generate coarse region proposals.

The first step of the proposed approach is to resize the input image to a size of 100x100, as discussed above. This resized image is given as input to the saliency algorithm. A saliency map is generated and normalized according to the grid-based scheme discussed earlier.

This normalized map is thresholded into a binary image, by setting all the pixel values greater than or equal to θ to one and the rest to zero. It is to be noted that the values in the saliency map would lie between 0 and 1 after normalization. Boxes are inferred on the thresholded binary image at the 100x100 scale. These boxes are mapped on to the original size of the image. These boxes should be cropped out of the image and fed to the detection CNN for processing. Fig 3.4 shows an example image with ground truth boxes from PASCAL VOC07 dataset.




*Figure 3.4: Sample Image with ground truth boxes*

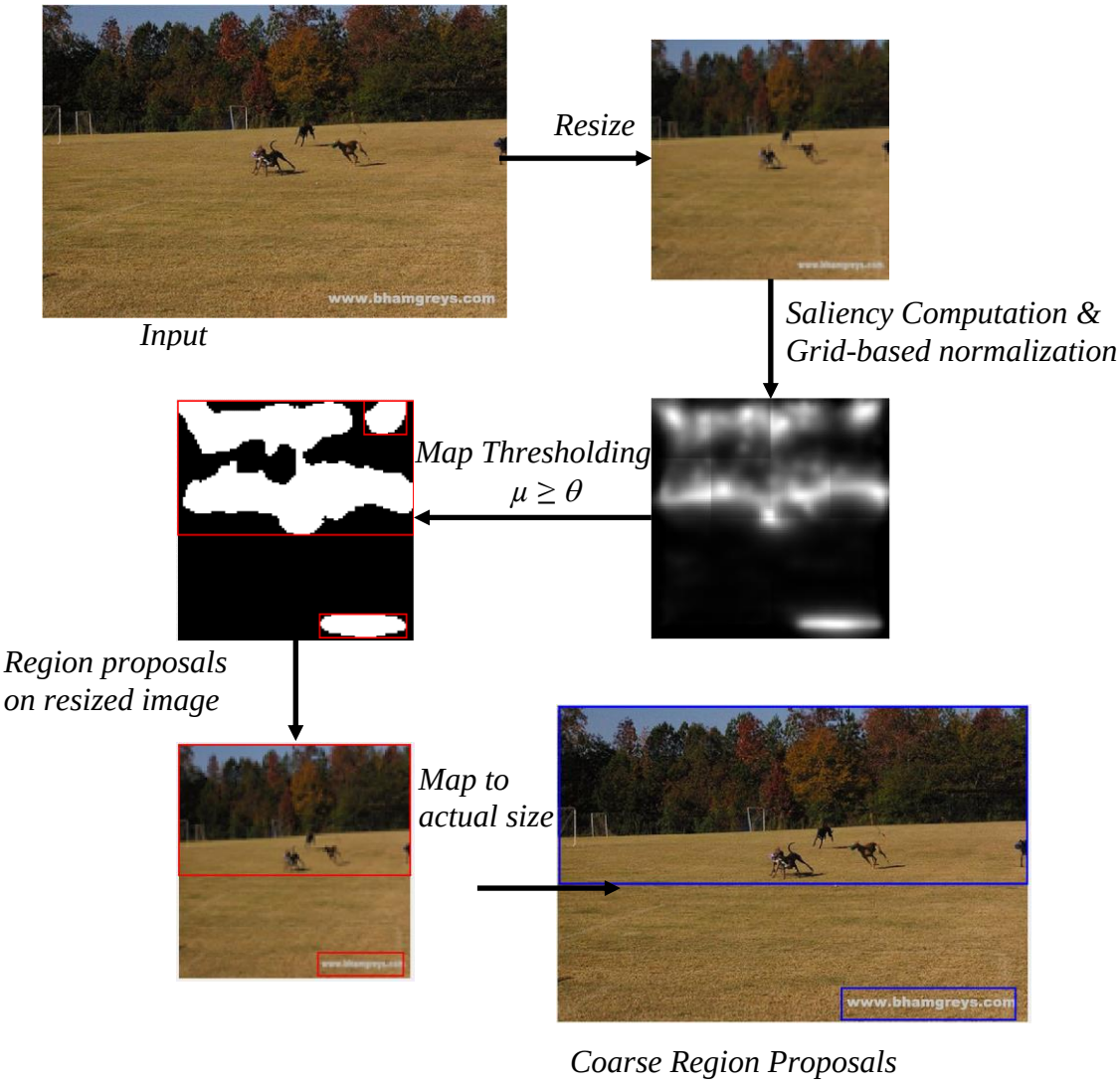



*Figure 3.5 Steps in the proposed approach*

Figure 3.5 above, shows each step and its output with an example image from the dataset. It can be seen that the final output boxes of the algorithm (blue bounding boxes) still have the ground truth boxes (see fig 3.4) inside them. Also, the area outside these boxes is the area saved in the image which translates into saved computations in the detection CNN.

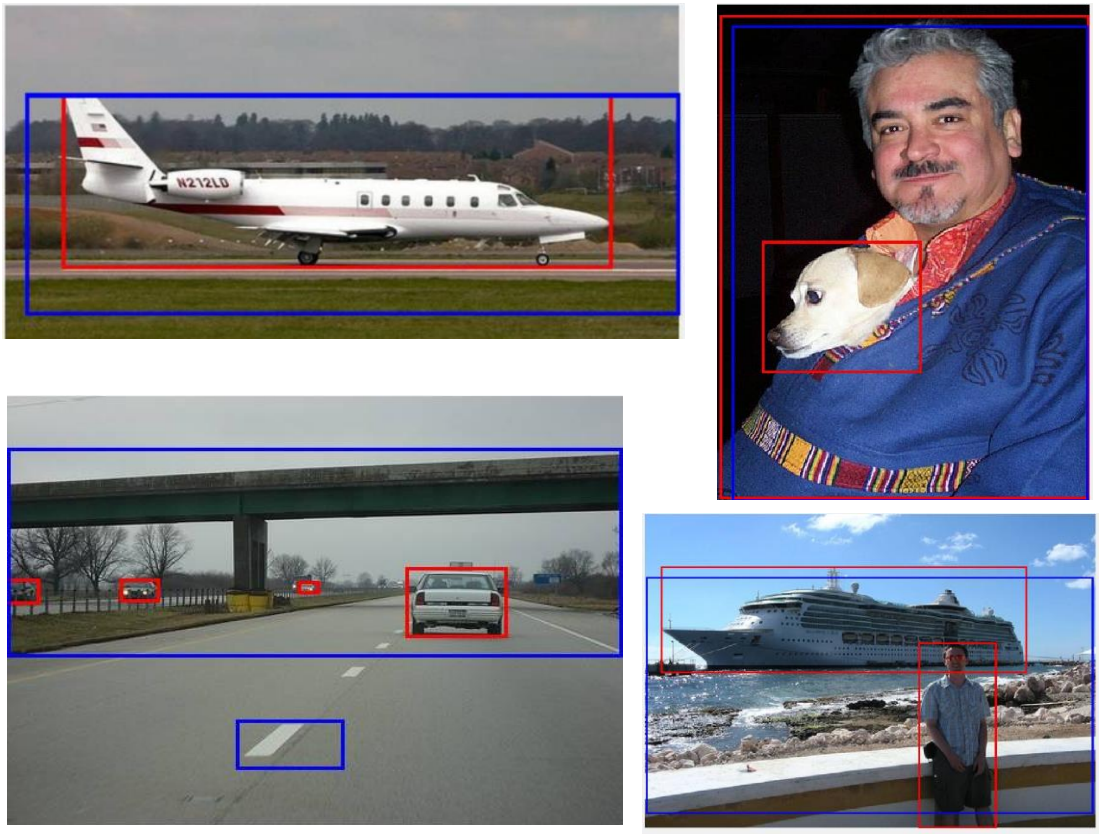



*Figure 3.6: Examples of saliency region proposals*

Figure 3.6 shows some examples of the region proposals from the proposed algorithm. The boxes in blue are the outputs of the proposed approach and the red boxes represent ground truth. The idea is to include as many red boxes in the blue ones or have a

significant overlap of the proposal with the ground truth. A quantitative approach to measuring this metric will be presented in the next chapter.

## *3.3.* **Implementation of Saliency with Faster R-CNN**

### *3.3.1.* **TensorFlow**

TensorFlow™ is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. TensorFlow was developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research [14]. TensorFlow can be called as a package from python. Every TensorFlow function is internally linked to its CUDA implementation, which enables users to access GPU without actually having to learn CUDA.

The saliency algorithm is implemented using this tool for GPU support to speed up the algorithm to fit in the Faster R-CNN framework. As saliency is an unsupervised algorithm, the supervised learning features in TensorFlow have not been used in this implementation. The matlab code from Johns Hopkins University was directly translated to TensorFlow with the modifications discussed in earlier sections.

### *3.3.2.* **PASCAL VOC Dataset**

The PASCAL VOC07 dataset has about 10,000 images and 20 object classes. The detection performance of an algorithm is measured in terms of mAP (mean Average

Precision). Average precision is the area under the precision-recall curve for a given class. Mean average precision or mAP is the mean of Average Precisions across all classes. The VGG based Faster R-CNN has a mAP of 69.9% on this dataset.

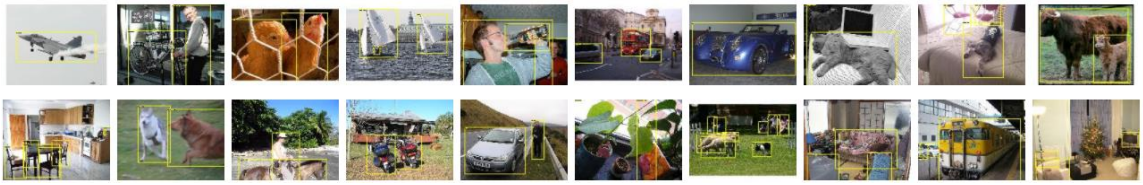


*Figure 3.7: Classes in PASCAL VOC dataset [4]*

### *3.3.3.* **Algorithm Implementation**

In this section key details of the implementation of the coarse region proposal using saliency will be discussed. As discussed in section 3.3.1, the relative ease of using TensorFlow and the availability of developed code base for Faster R-CNN are the main reasons for using it as an implementation tool for saliency. However, several features that enable training have been left unused in this implementation because saliency is an unsupervised algorithm. The primary functions used in this implementation come from the *Image* package of TensorFlow. A few key functions, among several other that were used, are:

- tf.depthwise_conv2d
- tf.resize
- tf.concat

All the conv filters used in the algorithm are stored in a file from which they would be loaded into the memory during runtime. This avoids computation of the kernels every time the algorithm is run, unlike the matlab implementation.

The conv2d function from the neural network library performs a dot product of the corresponding elements between the 3-D filter and 3-D input. These dot products are then combined to form a single slice of the output. However, in depthwise_conv2d, the depth of input and kernel are the same, like conv2d, but the kernel at a depth D only convolves with the feature map at depth D in the input. This feature allowed to stack all the 9 channels of saliency into a bundle of 9 feature maps.

Saliency convolution kernels are 2-dimensional array of values. At each stage of the computation the required kernels were stacked 9 times to form a hybrid 3-D kernel with a depth compatible with the input feature map. So, instead of having to operate the kernel on the channels in sequence, this function enables to get the output of a given stage of saliency, for all channels at once. The skimage package was used in the code to use the *morphology* module to remove noise like portions from the thresholded saliency map. The idea is to not consider portions of the image that are indicated as salient, but are too small to be an object. In this code, the highlighted regions with less than 100 pixels inside are removed. This technique helps in background noise mitigation.

# CHAPTER 4. RESULTS & DISCUSSIONS

The saliency based region proposal algorithm has been implemented to fit into the Faster R-CNN framework. A much lesser run time for saliency than Faster R-CNN is important for its successful integration into the detection framework. Also, the saliency output must encompass as much ground truth area as possible, so that the detection accuracy is not impacted adversely. An evaluation of speed and performance is presented in this section.

The average area savings using the saliency are 19.8%. This metric tells us how efficient the algorithm is in cropping out irrelevant portions of the object. It is measured by running the modified saliency algorithm over all the images in the dataset and by calculating the mean of area savings per image. The area saved per image can be obtained as:

$$Average\ area\ saving = 1 - \frac{Area\ inside\ the\ proposal}{Area\ of\ the\ image}$$

This is equivalent to the ratio of total area of blue boxes to the total area of the image under consideration, subtracted from unity. For example, the image in Fig 4.1 has 80.08% area saving.



*Figure 4.1: Example Image for area saving calculation*

To quantify the quality of the coarse region proposals from saliency we define a parameter called *IoG* (Intersection over Ground truth), which is similar to IoU, as shown below:

$$IoG = \frac{Area\ of\ intersection\ of\ saliency\ prediction\ with\ the\ ground\ truth}{Area\ of\ the\ ground\ truth}$$

This is different from IoU because here, the denominator is not the union of the two boxes in consideration, namely the saliency output and the ground truth. It is the area of the ground truth that appears in the denominator. We also define a positive, in the context of a saliency proposal. A saliency proposal would be a positive if its IoG exceeds a certain threshold. Also, another parameter called gt_coverage is defined as the ratio of no. of positives to the no. of ground truth boxes.

$$gt_coverage\ per\ image = \frac{number\ of\ positives}{number\ of\ ground\ truth\ boxes}$$

This metric tells us how well the output of saliency has predicted regions without missing the ground truth boxes. The average of this metric over the entire dataset is defined as the *gt_coverage.*

$$gt_coverage = \frac{\Sigma\, gt_{coverage}\ per\ image}{Total\ number\ of\ images\ in\ dataset}$$

Table 4.1 shows the gt_coverage for various IoG thresholds on the PASCAL VOC07 dataset.

| *IoG threshold* | *gt_coverage* |
|---|---|
| 0.9 | 77.77% |
| 0.8 | 83.88% |
| 0.7 | 88.09% |
| 0.6 | 91.67% |
| 0.5 | 95.17 % |
| 0.4 | 98.64 % |

*Table 4.1: gt_coverage for various IoG threshold values*

Table 4.2 shows the timing information on various steps in the algorithm. All the numbers are reported on NVidia TitanX GPU. The *Faster R-CNN* implementation in TensorFlow on average takes 168 ms per image at test time. The saliency step which includes saliency computation, grid based normalization and box generation completes in about 14ms. The next column is an estimate of the time Faster CNN would take on the cropped version of the input. This estimate assumes the linear dependence of computation on the input image area which is 134 ms. When this is combined with the saliency overhead of 14 ms it amounts to 148ms. This is 20 ms or 12 % lesser than the actual time taken by Faster R-CNN.

| **Faster R-CNN** | **Saliency Step** | **Faster R-CNN on cropped input (estimate)** | **Saliency + faster R-CNN** | **Reduction** |
|---|---|---|---|---|
| 168 ms | 14 ms | 134 ms | 148 ms | 12% |

*Table 4.2: Timing Results*

Figure 4.2 shows a few examples of the coarse region proposals from the saliency algorithm.

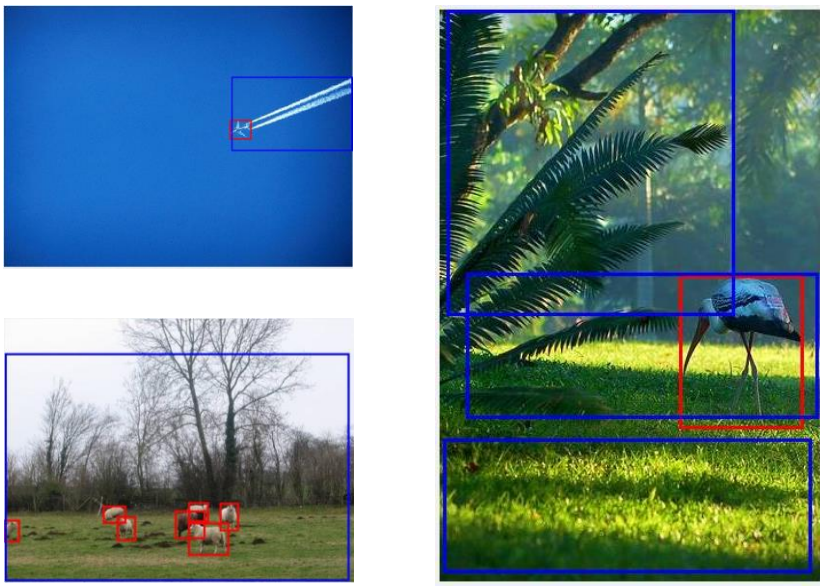


*Figure 4.2: Examples of coarse region proposals*

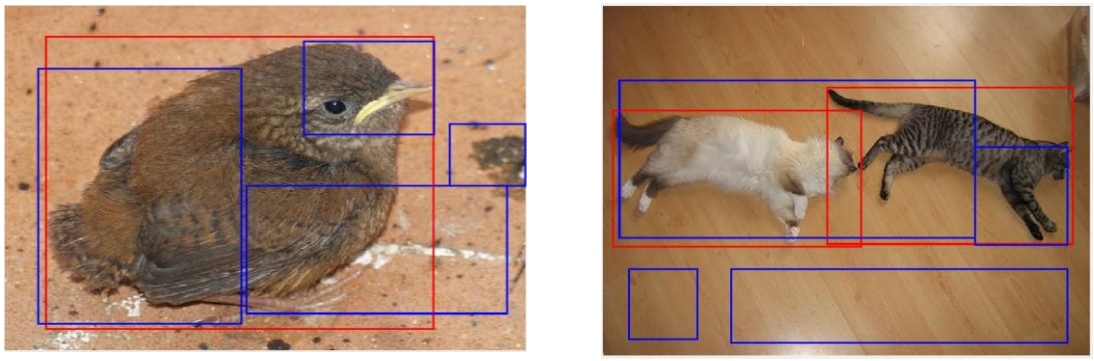


*Figure 4.3: Issues in the current approach*

Figure 4.3 shows the existing issues in this approach which are splitting of a single object into multiple regions (left) and missing out a portion of the object (right). These issues can affect the detection accuracy of the CNN and might require re training the CNN to mitigate such issues.

# Chapter 5 CONCLUSION

In summary, this work is driven by the motivation of mimicking the usage of selective attention mechanisms used in the visual cortex of the brain and applying it to solve the computational bottlenecks in computer vision problems and to get closer to the real-time mark in mobile vision applications.

With the recent developments in neural networks, GPUs, availability of huge datasets like ImageNet, the accuracy of algorithms on computer vision problems has surpassed human level accuracy. However, the algorithms with such accuracies would not fit within the power envelope of embedded systems due to the high computational demand. In an attempt to make these cutting-edge algorithms more suitable for mobile platforms and get closer to the real-time mark for mobile vision, we use visual proto object saliency to reduce the computations in detection CNN of Faster R-CNN, an object detection framework. It has been shown that the conv step of Faster R-CNN amounts to ~ 70% of the total computations involved. So, the target was to reduce these computations, while not badly affecting the detection accuracy.

The algorithm to generate coarse region proposals that would reduce the search space for CNNs has been implemented. The algorithm gives a 19.8 % saving in the search space, can cover over 88% of the ground truth boxes at an IoG threshold of 0.7. It can also potentially speed up the algorithm by 12%. Arriving at the idea of grid-based normalization earlier and implementation in tensorflow at an earlier stage would have saved some time and implementation efforts in Caffe, CUDA and python.

Some key future works include, developing a method to train and test the detection CNN with the generated proposals, enhancing saliency implementation by implementing

it in CUDA (to overcome the restrictions imposed by pre-implemented functions), exploring other saliency algorithms that would fit the purpose and share computations with the detection CNN.

REFERENCES

[1] A. F. Russell, S. Mihalas, R. von der Heydt, E. Niebur, and R. Etienne-Cummings, "A model of proto-object based saliency," Vision Research, vol. 94, pp. 1–15, 2014.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards 14 real-time object detection with region proposal networks," in Neural Information Processing Systems (NIPS), 2015.

[3] http://cs231n.stanford.edu/

[4] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J. and Zisserman, A., "The PASCAL Visual Object Classes (VOC) Challenge" International Journal of Computer Vision, 88(2), 303-338, 2010

[5] R. Girshick, "Fast R-CNN," in IEEE International Conference on Computer Vision (ICCV), 2015.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," in International Journal of Computer Vision (IJCV), 2015.

[8] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995

[9] http://www.image-net.org/challenges/LSVRC/

[10] Krizhevsky, A., Sutskever, I., and Hinton, G.E. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.

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

[12] Kimchi from saliency

[13] Kulikowski, J., Marcelja, S., & Bishop, P. (1982). Theory of spatial position and spatial frequency relations in the receptive fields of simple cells in the visual cortex. Biological Cybernetics, 43, 187–198.

[14] https://www.tensorflow.org/