REPRESENTING IMAGES AND REGIONS FOR OBJECT RECOGNITION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

İLKER BUZCU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2015

Approval of the thesis:

REPRESENTING IMAGES AND REGIONS FOR OBJECT RECOGNITION

submitted by **İLKER BUZCU** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Gönül Turhan Sayan _____
Head of Department, **Electrical and Electronics Engineering**

Prof. Dr. A. Aydın Alatan _____
Supervisor, **Elec. and Electronics Eng. Dept., METU**

Examining Committee Members:

Prof. Dr. Uğur Halıcı
Electrical and Electronics Engineering Department, METU _____

Prof. Dr. A. Aydın Alatan
Electrical and Electronics Engineering Department, METU _____

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering Department, METU _____

Assist. Prof. Dr. Elif Vural
Electrical and Electronics Engineering Department, METU _____

Assist. Prof. Dr. Nazlı İkizler Cinbiş
Computer Engineering Department, Hacettepe University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: İLKER BUZCU

Signature :

ABSTRACT

REPRESENTING IMAGES AND REGIONS FOR OBJECT RECOGNITION

Buzcu, İlker
M.S., Department of Electrical and Electronics Engineering
Supervisor : Prof. Dr. A. Aydın Alatan

September 2015, 83 pages

We can represent images in entirely different ways, in order to fulfill different purposes. For object recognition, power of a representation comes from its discriminative ability. In this thesis work, handcrafted representations that dominated the last decade of computer vision are evaluated against the current paradigm of Deep Learning, to try and pinpoint the reasons behind why and how the fairly old Artificial Neural Network (ANN) framework suddenly emerged as the state of the art in discriminative representations. We observe, through our experiments, that true capabilities of Deep ANN's can only be achieved by having very large amounts of labeled data that have been made available only recently. This thesis work also deals with ensembles of both handcrafted and ANN based approaches to reinforce the new technology with years of established knowledge behind handcrafted feature based approaches. For this purpose, we propose a novel extension, based on Fisher Vectors, to the well known Selective Search algorithm, called the Fisher-Selective Search algorithm, and obtain a 10% relative increase in Average Precision at virtually no additional computation cost.

Keywords: Visual Object Recognition, Image Representations, Fisher Vectors, Convolutional Neural Networks, Selective Search

ÖZ

NESNE TANIMA İÇİN GÖRÜNTÜ VE BÖLGELERİN BETİMLENMESİ

Buzcu, İlker
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. A. Aydın Alatan

Eylül 2015 , 83 sayfa

Görüntüleri farklı amaçlara yönelik, tamamen farklı biçimlerde betimleyebiliriz. Nesne tanıma uygulamalarında, görüntü betim yönteminin gücü ayrıştırıcı niteliğine bağlı olmaktadır. Bu tez çalışmasında, geçtiğimiz on yıl boyunca hüküm sürmüş el yapımı betimleme yöntemleri, günümüzün Derin Öğrenme paradigmasıyla karşılaştırılmakta; bu yolla, nispeten eski kökleri olan Yapay Sinir Ağları (YSA) yapısının ani yükselişinin temel nedenlerine inilmesi amacı güdülmektedir. Öne sürdüğümüz deney sonuçları, Derin YSA'ların gerçek performansına ulaşabilmesi için yakın zaman öncesine kadar sahip olmadığımız, çok büyük, etiketlenmiş veri kümelerine gereksinim duyduğunu göstermektedir. Bu tez çalışması aynı zamanda el yapımı yöntemler ile YSA tabanlı yaklaşımların birleşimi üzerine kurulu metotlara değinmektedir. Bu şekilde, yeni teknolojinin el yapımı yöntemlerin yıllar içerisinde oluşturduğu birikim ile desteklenmesi amaçlanmaktadır. Bu doğrultuda Seçici Arama algoritmasına Fisher Vektörü tabanlı, Fisher-Seçici Arama isimli bir eklenti önermekteyiz. Algoritmamız ek bir hesaplama zamanı harcanmaksızın, ortalama kesinlik değerini % 10 arttırmaktadır.

Anahtar Kelimeler: Görsel Nesne Tanıma, Görüntü Betimleri, Fisher Vektörü, Evrişimli Sinir Ağları, Seçici Arama

In memory of my father

ACKNOWLEDGMENTS

I am grateful to Prof. Dr. A. Aydın Alatan for his constant encouragement, and for his unfailing guidance through the hardest of times.

My deepest gratitudes to Havelsan and Aselsan for their financial support.

A huge thank you to the overwhelmingly supportive group of coworkers and close friends in the Multimedia Research Group: Beril Beşbınar, my companion for the all-nighters; Emrecan Batı, our all-knowing guide on the path to enlightenment, a.k.a. Deep Learning; Yeti Ziya Gürbüz, for his words of wisdom; Akın Çalışkan, Ömürcan Kumtepe, Ece Selin Böncü, and Alper Koz.

A whole lot of love to my significant other, Meltem Çiçek; who stayed up with me for many long nights to keep me company, who in the face of struggle gave me the willpower to continue, who is my source of strength and inspiration.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ALGORITHMS

ALGORITHMS

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
BING	Binarized Normed Gradients
BLAS	Basic Linear Algebra Subroutines
ConvNet	Convolutional Neural Network
GPU	Graphical Processing Unit
FK	The Fisher Kernel
FSS	Fisher Selective Search
FV	Fisher Vector
NG	Normed Gradients
PCA	Principal Component Analysis

CHAPTER 1

INTRODUCTION

Inductive inference is the tool we use to make predictive, general statements from limited information. It is our way of extending knowledge beyond what we can directly observe. The typical example, given as inductive reasoning leading to an incorrect generalization, is:

- All of the swans we have seen are white.
- Therefore, (it is probable that) all swans are white.

Turns out, black swans exist! But that is fine, as long as most of our predictions and generalizations are correct, we are better off trying to extend our knowledge this way, instead of not making any generalizations at all.

When it is computers who generate predictions and generalizations out of incomplete data, we call it **machine learning**. Machine learning mimics human induction: in supervised learning, the training dataset consists of a number of input-output pairs. The learner tries to find a general relation between the input and the output, such that it can be used to predict the output of input features it has not seen in the training stage. In unsupervised learning the situation is similar; the learner tries to find patterns in the data that are applicable to test inputs as well, i.e. the findings should be generalizable.

The advantage of computers over us is their ability to work with numbers much more efficiently - they are called computers, after all! The problem, however, with computer vision based machine learning problems, such as image classification, is that the numbers that represent images, i.e. the pixel values, are not directly related to the

content of the image. It is not reasonable to expect a machine learning algorithm to magically produce generalizations beyond finding out that the background of ship images are blue, whereas the background for car images would be mostly gray (Figure 1.1).

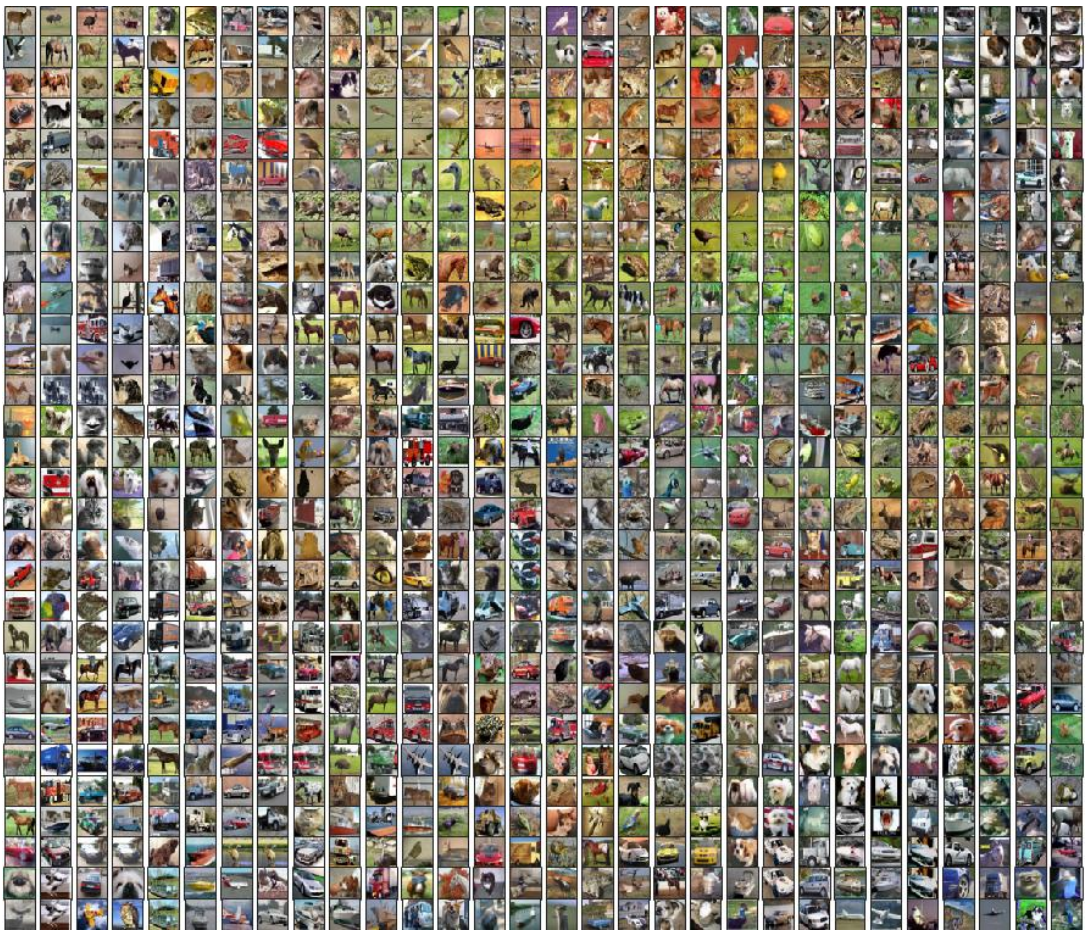

Figure 1.1: Images from CIFAR-10 dataset, grouped according to their raw pixel value similarity using t-SNE [74]. It can be observed that images with similarly colored backgrounds are grouped together, regardless of the object class.

For classification problems, we should evaluate representations in terms of their discriminative power. That is to say, a *distorted* representation that amplifies distinguishing features of each class usually results in higher classification accuracy. Such a distortion can be done via a non-linear transformation on the original feature space. Figure 1.2 demonstrates with a toy example how the original representation can be turned into a linearly separable one by applying such a transformation. On the same note, a visually rich photograph can be much harder to classify than a simple drawing

(Figure 1.3)-even though they are both trivial to the human visual system.

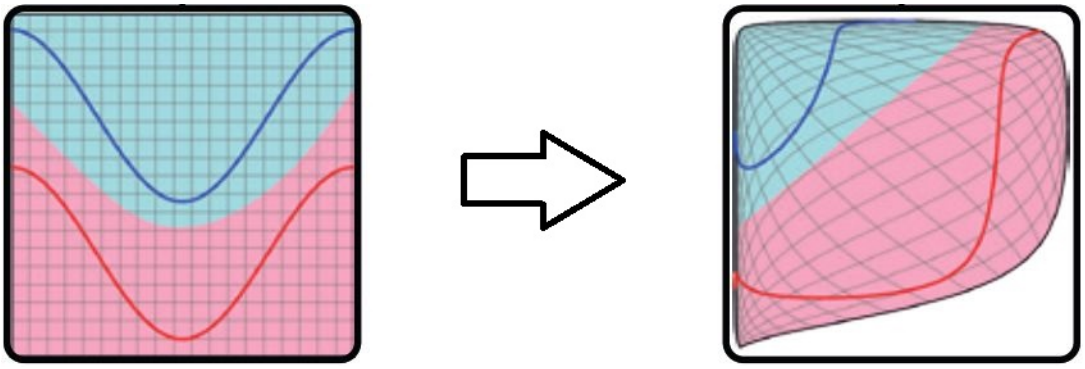

Figure 1.2: Non-linear transformations of the feature space are useful in obtaining more discriminative representations. Taken from http://colah.github.io

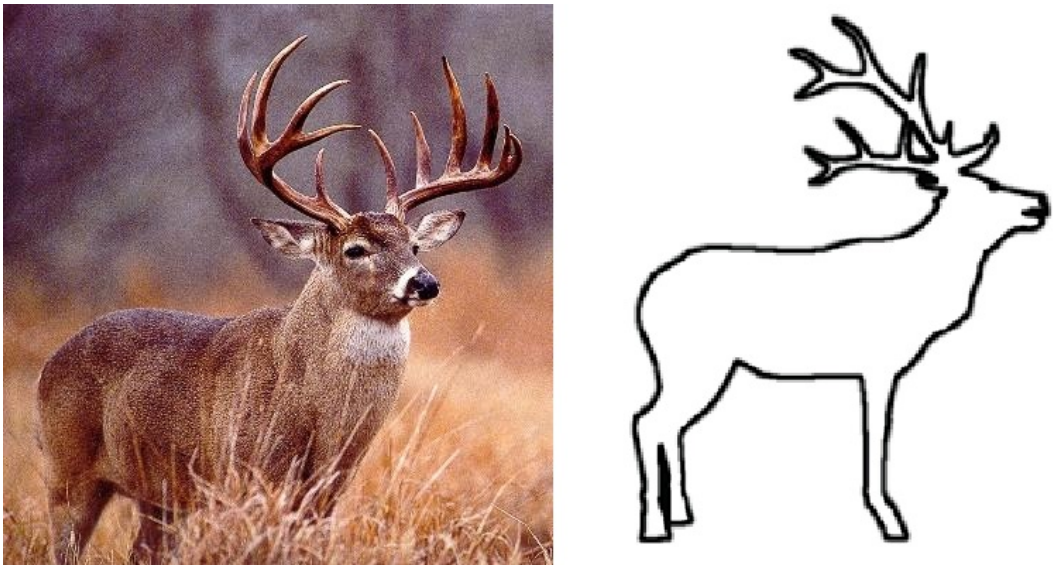

Figure 1.3: Simpler representations may hold more discriminative power.

The point to be made here is that in order to apply Machine Learning techniques to Computer Vision problems, we *need* better representations of data, beyond raw pixel values. How to obtain those representations, particularly for images, is one of the two focal points of this thesis work; the other is to extend those representations to regions of an image in search of solutions to the problem of localization.

1.1 Problem Statement

If Machine Learning in general is an imitation of inductive inference, then applying Machine Learning to Computer Vision is trying to emulate the evolution of human visual system. At the very least, it is the benchmark any learning-based computer vision system is ultimately compared against [45] [44]. It is a tall task; the evolution of the eye started more than 500 million years ago [59]. While our methods may be more sophisticated and efficient, it is still quite hard to match the huge processing power, dataset size and training time in the hands of evolution.

Nevertheless, we have been making some leeway in all of these areas: the PASCAL Visual Object Classes(VOC) project started in 2005 with a dataset of 1578 images of 4 classes (bicycles, cars, motorbikes, people). By 2011, the size of the dataset was increased to 11,530 images of 20 classes [23]. Its successor, the ImageNet Large Scale Visual Recognition Challenge(ILSVRC) escalated the progress even further, with the number of images in the competition dataset surpassing 500,000 [66]. Meanwhile, the processing power available for the state of the art systems improved significantly, mostly thanks to the parallelized architecture of neural networks which allows training to be done in GPU (Graphical Processing Unit) clusters, with batches of images at a time [80]. The critical point of computers being competitive with humans at specific vision tasks has been reached, as exemplified by reports of surpassing human performance in the task of classifying images [36].

An elegant definition of Machine Learning is made in [54]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." Within this formulation, we may define the image classification problem as follows:

- C = A set of object classes.
- T = A set of tasks, such that:
  For each object class c_i, the task t_i is to recognize whether an object belonging to c_i exists in that image.

- E = An image with ground truth annotation that indicates whether (and optionally, where) an object belonging to a class of objects exists in that image is inputted to the learner.
- P = An evaluation metric, accompanying a test set of images.

In PASCAL VOC, the tasks are evaluated separately. The algorithms are expected to output 20 score values, one for each class task, for each test image. The scores output by the algorithm ought to relate to its confidence in its prediction: too high/too low scores indicate with very high confidence that an object of the class in question certainly exists/does not exist. For each task, a precision/recall curve is drawn according to the scores output by the algorithm for the corresponding class. To do this, the scores are sorted from highest to lowest, and the values higher than some threshold are labeled as belonging to that class. Comparing these labels with the true labels, we count the number of True Positives(tp) - the number of images that contain an object of the class that are classified as such; False Positives(fp) - the number of images that do not contain any object from that class, but classified as such; and False Negatives(fn) - the number of images containing a class object but misclassified as not having any. For a given threshold, precision and recall values are derived from these counts as:

$$prec = \frac{tp}{tp + fp}, rec = \frac{tp}{tp + fn}. \tag{1.1}$$

By varying the threshold, we obtain different precision and recall values, which we draw into the Precision-Recall curve (Figure 1.4). From the curve the average precision (AP) value, which corresponds to the area under the curve, is computed, and used as the performance metric for a given class. The mean average of all class AP values can be used to evaluate the overall performance of an algorithm [23]. Different formulations for computing the AP are used in the literature, which result in slightly different AP values as in Figure 1.4.

For ImageNet, the sheer scale of the dataset complicates the annotation process greatly. For this reason, a very simplistic strategy is used: each image is assigned to a single object class, even if multiple objects of a variety of classes exist in that image. Algo-

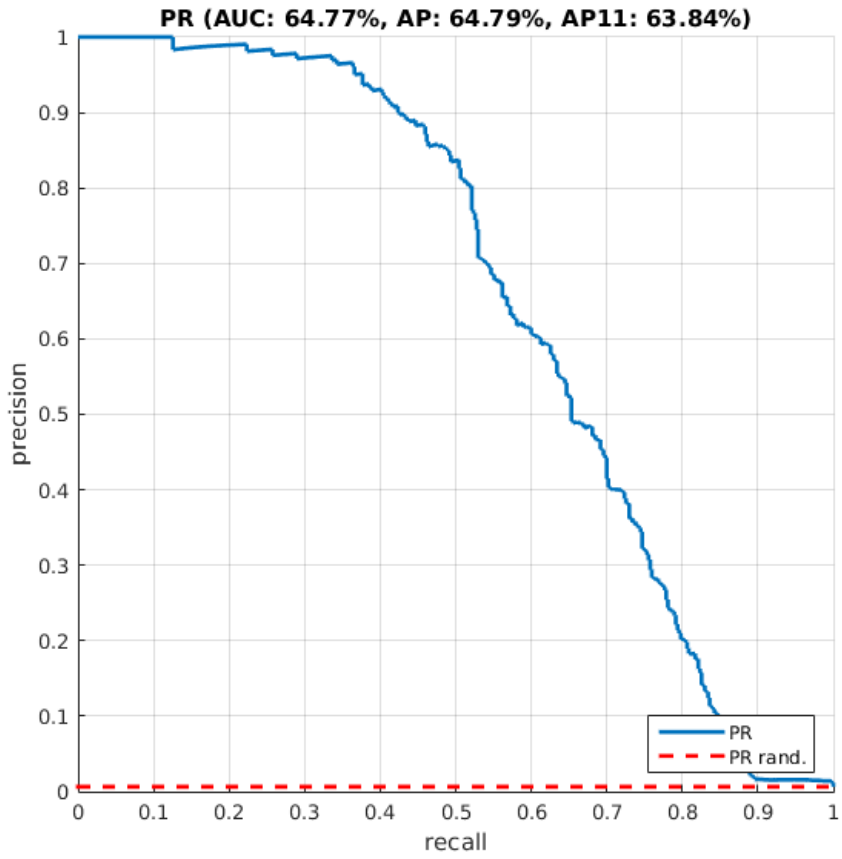

Figure 1.4: An example Precision-Recall curve, made with the VLFeat Toolbox for MATLAB [75]. The numbers at the top correspond to the Average Precision values computed in different ways.

rithms are expected to make guesses about the assigned class of a test image, sorted according to their likelihood. Top 5 guesses of an algorithm are taken into account to compute its top-5 accuracy, according to which competing algorithms are evaluated.

Image classification is a comparatively simpler task for a computer to perform; the space of possible answers that the computer can give is quite limited. The VOC classification challenge boils down to 20 binary classification problems. By contrast, the object detection problem is more complicated - for a successful detection, the object should be correctly classified in addition to its location being identified (Figure 1.5). Let us give a formulation of the detection problem, in line with its definition in ImageNet and VOC challenges:

- C = A set of object classes.
- T1 = To find the bounding box of each object, if any, that belongs to some member of C, within a given image.
- T2 = A (possibly empty) set of tasks, defined as:
  For each object o_i (found in T1), decide correctly on its class c_i.
- E = An image with ground truth annotation of bounding boxes for all objects that belong to some member of C is inputted to the learner.
- P = An evaluation metric, accompanying a test set of images.

Alternatively, E can be the same as in classification, i.e., the learner may be told only what the objects are, but not where they are. In that case the learner has to infer the location of the object in an unsupervised manner.

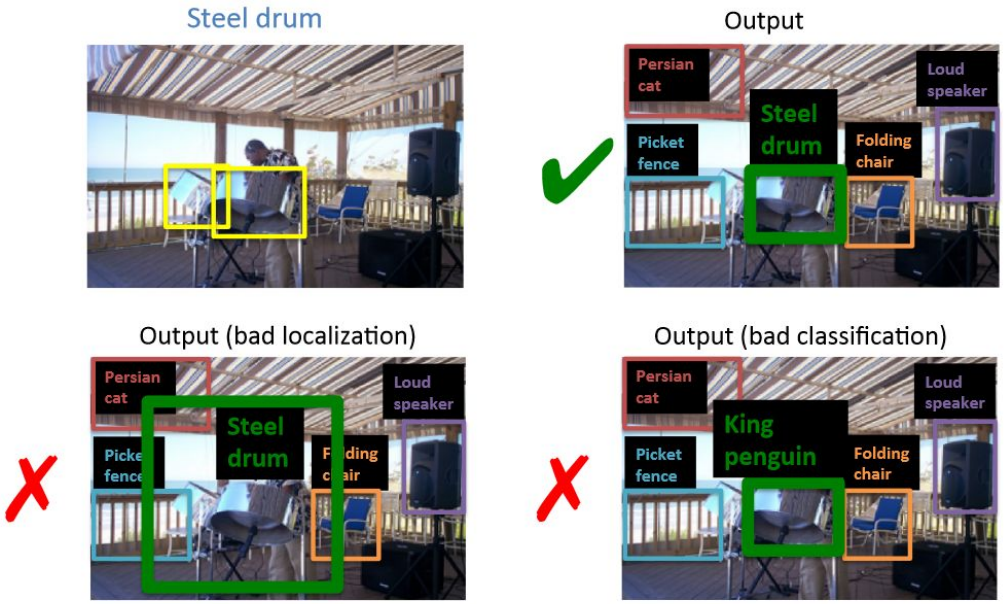

Figure 1.5: The ImageNet localization challenge. A detection is deemed correct only if it succeeds at both classification and localization. Taken from [66]

Since the detection problem essentially contains the task of classification, it should be no surprise that its solution commonly involves a classifier stage. The baseline sliding window approach, used for a wide variety of detection tasks ([76], [17], [34], [25], [24]), consists of a blindly exploring the image with a window, taking a crop at the

location of the window every few steps and classifying the cropped part of the image (Figure 1.6). The computationally expensive nature of this approach has been cause for more sophisticated approaches to reduce the number of candidate windows at a reasonable computational cost have been proposed ([72], [13]). Chapter 4 is devoted to exploring this relation between classification and detection. It should be noted that detection methods are also applied to classification problems to improve accuracy by localizing the object of interest ([35], [56]); but this inverse relation is not explored in this thesis.

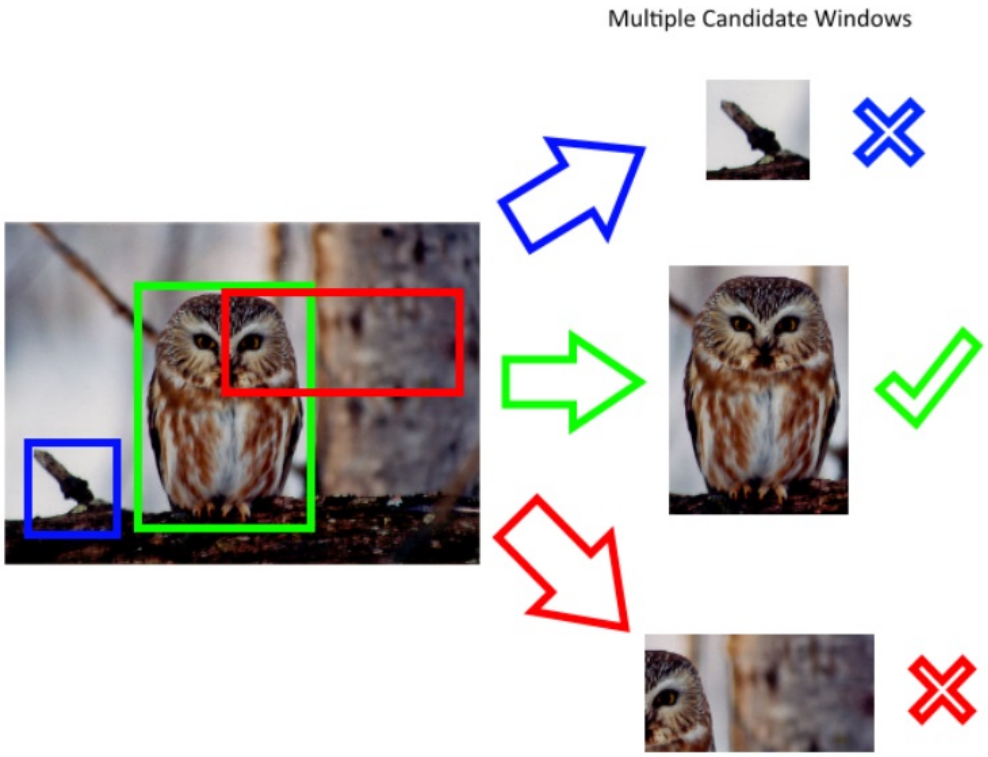

Figure 1.6: The typical detection framework: multiple candidate windows are extracted from the image, which are in turn classified.

1.2 Representing Images and Regions

For visual recognition tasks, we need representations of the visuals that are well suited to the task at hand. In its simplest form, a computerized representation of an image is a two-dimensional array of raw pixel values. For the problem of image classifica-

tion, we need the representations of images to hold discriminative power, such that a classifier trained with them can give accurate results. Image representation techniques can be divided into two groups: handcrafted versus learned representations - although other taxonomies certainly also exist. Handcrafted representations are based on some engineered feature extraction algorithm, whereas for learned representations the computer learns to extract the features from pixel values.

For handcrafted representations, learning starts after some initial computation of features. This is not to say that there is no aspect of machine learning in obtaining the final image or region representation; it only means that the raw -or preprocessed- pixel values never enter a learning system as inputs.

Another way of making the distinction between handcrafted and learned representation is to look at where the representation starts to depend on not only the image itself, but also the training dataset. In 2.3 we discuss Fisher Vectors, which is an encoding of local features of an image in terms of the generative model that best describes the whole set of local features in the training data. While the encoding of the local features depends on the model which in turn depends on the features used to train the generative model, the extraction of those local features is a product of a handcrafted algorithm. Therefore, we classify the Fisher Vector as a handcrafted representation.

Representing an image region is not much different from representing the whole image. The subset of rectangular regions can be represented in the exact same way as images, and some representation techniques can work just as well for irregular regions. However, working with structured, rectangular inputs is absolutely necessary for other methods. Convolutional neural networks (ConvNets), for instance, even require input images to rescaled to a certain width and height. Therefore, it has become quite common to limit the regions of interest to rectangular ones only.

1.3 Contributions and Outline of the Thesis

Chapter 2, which is the first main body of this thesis work, gives the theoretical background of discriminative representations experimentally explored in the chapters that follow. The other main chapters are more specialized in their scope, with Chapter

3 focusing on the problem of image classification while Chapter 4 deals with object detection.

In Chapter 3, we describe and compare two completely different archetypes of classifiers which are both considered state-of-the-art in their own niches, with specific focus on experiment designs that bring out conclusive results about the strengths of the approaches relative to each other.

The biggest novel contribution of this thesis work is described in Chapter 4: The Fisher-Selective Search algorithm, an enhancement applied to the original Selective Search algorithm. We also discuss the relation between our two problems of interest, and explain why detection is in many cases solved through classification.

Finally, in Chapter 5, the analyses of the results established in the previous chapters are reiterated, and future work discussed.

CHAPTER 2

REPRESENTING IMAGES AND REGIONS FOR OBJECT CLASSIFICATION AND DETECTION

We can represent images in entirely different ways, in order to fulfill different purposes. For instance, both lossless and lossy compression standards are proposed to generate storage-friendly representations of images. In its simplest form, a computerized representation of an image is a two-dimensional array of raw pixel values; any other representation may be derived from it, while the converse might not be possible for every derived representation.

For the purpose of classification, the distinction is made in a different direction: the discriminative power of the representation. A lossy representation is acceptable, even encouraged; as long as the losses come from non-discriminative parts of the image. A good representation technique for the task of classification, then, would be one that describes images of same objects, or same scenes, similarly. In other words, a good representation technique generates, for similar images, feature vectors that are located closely in the feature space. For the pure two-dimensional pixel value array representation such a relation does not generally exist, making it unusable for most machine learning approaches.

Studies have shown that completely different paradigms of machine learning, when applied properly, give reasonably similar results when applied to the same dataset [52]. This amplifies the need to have good representations: if we can obtain a good feature vector for each image in terms of their discriminative power, the classification problem becomes almost trivial. This chapter is therefore dedicated to the various paths taken toward obtaining a good representation of images.

Representing image regions is a straightforward extension to representing images; for a rectangular cropping of the image the exact same strategies apply, whereas for arbitrary regions we are more limited, or have to be more creative, in how we make use of spatial information, since many of the structured approaches cannot be applied.

2.1 Global Features of Images

Global features of an image are those that are computed directly from the pixel values of an image. An example of global features are graylevel or color histograms, which indicate the intensity or color distribution of pixels within the image. A simpler global feature would be one that takes the average value of all pixels for each color band, generating a 3-dimensional feature vector for an RGB image.

For machine learning tasks, the discriminative information provided by global features are considered to be incomplete. Therefore, they are used often as complementary features, attached to some other representation which is lacking in some other manner. An exemplary method suited for this purpose is found in [70], where the global representation explicitly aims to capture the **gist** of the image, that is, a low dimensional representation that informs of the general context of the scene, as opposed to the detailed information that describes the objects in the scene.

2.2 Pooled Local Features

The alternative to obtaining a complete global representation of a whole image, directly, is to first consider local features of said image and then *pool* those local features into a representation of the whole image. Local features are useful as a starting point, since there are several patch feature extraction methods (e.g. HOG [18], SIFT [53] and its variants) that are quite sophisticated, in the sense that most of the useful information in the image patch is preserved.

The pooling process can be done in different ways. A basic strategy might involve taking the average of all local features in an image; but doing this directly will result in an unacceptably large loss of discriminative power. To avoid this, the local features

must go through some sort of feature encoding stage. The generic framework is summarized in Figure 2.1.

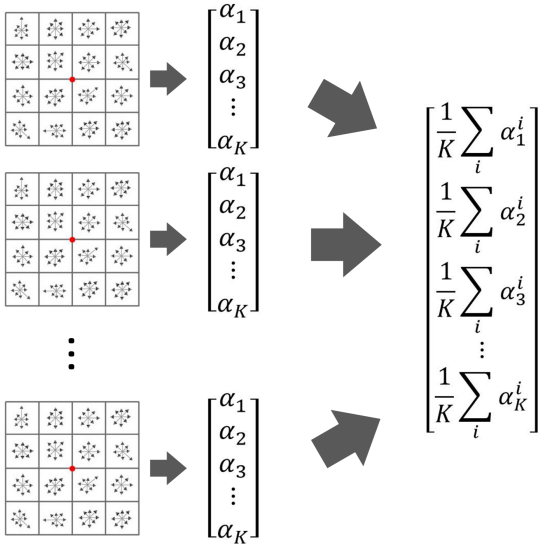

Figure 2.1: The pooled local features framework. Local features are first coded according to some encoding scheme; then the coded local features are pooled to generate the feature vector for the whole image. Here, the average pooling scheme is demonstrated.

In [9], two major pooling strategies are described, in addition to the two most popular encoding schemes used. These strategies merit detailed explanation.

2.2.1 Feature Encoding

For our problem, encoding is the act of assigning a set of values for a descriptor vector of a patch feature. We may consider the feature encoding step as a transformation to a new feature space; in the new feature space, the codewords are the basis vectors, and each patch feature is described in terms of a linear combination of the codewords. Since the selection of codewords defines the feature space, it greatly affects the classification performance.

The codewords are usually chosen to minimize the loss of information, depending on the feature encoding strategy; suboptimal codeword selection will result in coded descriptors having less discriminative power, which in turn diminishes classification performance. Another issue is sparsity of the codes: pooling strategies need to summarize contributions from several features. With sparse representations there is less

overlap between representations of different features, generating little information loss at the pooling stage. To summarize, there is a tradeoff between obtaining good codes that replicate most of the information in the raw feature vector, and sparse codes that adhere well to pooling.

2.2.1.1 Hard Quantization Encoding

In hard quantization encoding, each possible local feature is assigned to a single one of the codewords. The resulting codes are binary and very sparse, in fact only a single element has the value '1' while the rest of the vector takes zero values. For a feature space constructed with K codewords, the feature vector values are computed as:

$$\alpha_i^j \epsilon \{0,1\}, \alpha_i^j = 1 \iff ||\mathbf{x}^j - \mathbf{w}^k|| > ||\mathbf{x}^j - \mathbf{w}^i||, i \neq k, \tag{2.1}$$

where x_j indicates the j'th local feature vector, w_k denotes the k'th codeword and α's are the code weights. In non-mathematical terms, we can simply say that the feature is assigned to the nearest codeword. The quantization in this approach occurs due to the fact that there is only K different codes, not nearly enough to have a unique mapping for all possible patch features. In fact, all features that have the same codeword as the closest are mapped to the same point in the feature space, and from this point onward become indistinguishable from each other.

2.2.1.2 Sparse Encoding

In sparse encoding, the goal is to obtain sparse representations that preserve as much of the information during the encoding step as possible. Naturally, the encoding scheme turns into solving an optimization problem with the sparsity constraint adjusted by a parameter, λ. Instead of hard assignments, the local features are coded as a linear combination of several codewords. In this soft assignment process, the sparsity constraint limits the number of codewords that contribute to representing each local feature. The code vector can be computed as:

$$\alpha^j = \arg\min_{\alpha} ||x^j - \mathbf{W}\alpha^j||_2 - \lambda||\alpha^j||_1. \tag{2.2}$$

L1 norm is used with a regularization parameter to induce sparsity. The optimization problem can be tackled in two ways: the dictionary matrix **W** can be learned separately beforehand with an algorithm such as k-means; or dictionary learning and learning of sparse representations can be considered together as a single optimization problem, tackled by iteratively adjusting codewords followed by representation coefficients.

2.2.2 Pooling

Pooling is the step required to summarize the coded patch features in a single vector that is supposed to represent the whole region of interest. Even though the relation between coded features and the corresponding pooled representation is not one-to-one, with a well constructed sparse encoding scheme the final representation retains most of the discriminative power of the patch features, but at the level of the whole image, or the whole region of an image.

2.2.2.1 Average Pooling

In average pooling, we simply take the average of the codes of features within the region of interest. A representation for the whole image is the average of all coded patch features, calculated as:

$$\mathbf{\Phi} = \frac{1}{N}\sum_{j=1}^{N} \alpha^j, \tag{2.3}$$

where N is the number of patch features, α^j is the j'th coded patch feature, and $\mathbf{\Phi}$ is the final image representation.

2.2.2.2 Max Pooling

The alternative strategy is to use max pooling, in which the maximum valued coefficient in each codeword's dimension is selected, and combined to obtain the final representation. It can be formalized as:

$$\Phi_i = \max_j \alpha_i^j. \tag{2.4}$$

Effectively, max pooling enforces additional sparsity by discarding the smaller coefficients, at the cost of discarding the information contained in those coefficients. The choice between average and max pooling, again, is closely related to the tradeoff of sparsity versus representative power.

2.2.3 Bag of Visual Words

The benchmark approach, bag of **visual** words(BoV) (Figure 2.2), was inspired by techniques used in natural language processing. Given a text, most of the discriminative information can be obtained by simply counting the number of occurrences of each word. The word counts, by themselves, give enough information for us to discriminate between technical and non-technical texts, classify its topic, or to give a more specific example, decide between spam and non-spam e-mails. The resulting representation is named "the bag of words", which paints a nice visualisation of the process: Grammatical structure and word ordering are discarded as all words within the text are thrown in the bag, then the feature vector is obtained by counting the occurrences of each word inside.

The bag of words representation can be described as the pooling of local features in a text, i.e., words. The final representation is obtained by normalizing the counts, so in this sense, the bag of words representation is an average pooling of local features. We see that its image counterpart, BoV, is intuitively quite similar: if visual words are local features of an image, then the *bagging* is the process of discarding all spatial structure, and the final representation becomes the average pooling of local feature occurrences.

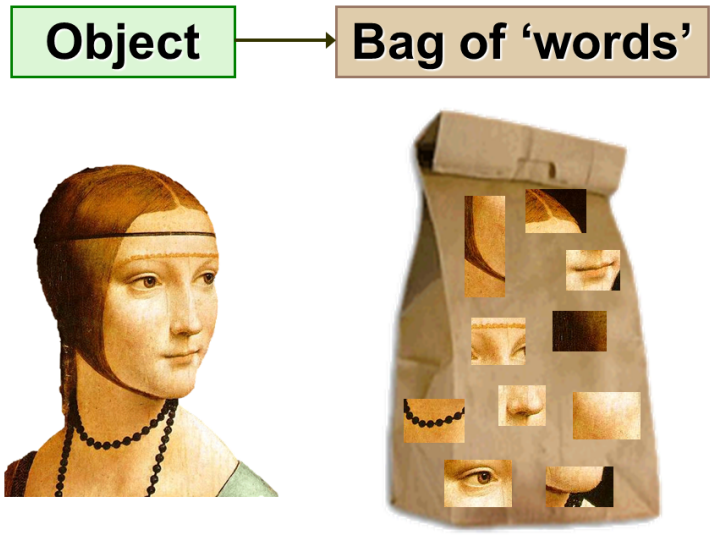

Figure 2.2: The bag of visual words representation. The object is described as a summary of all local visual features. Spatial information and correspondences between features are lost in the process. Taken from http://sensblogs.wordpress.com

In the generic BoV framework, the dictionary is computed via the k-means algorithm. The visual words are hard quantized; in other words, a given feature vector is described in terms of which dictionary codeword it is closest to - and no other information is given. Taking the histogram of visual word counts is equivalent to average pooling hard quantized representations, which turns out to be another way of describing the BoV methodology.

The limitations and drawbacks of the standard pooling strategy is demonstrated by the heavy quantization inherent to the BoV representation. Most of the information loss due to quantization may be restored by using soft assignments instead of hard assignments, i.e. with sparse encoding. Kernel codebooks [31] and Locality-constrained Linear Coding [78] are examples of this approach. The alternative is to increase the dimensionality of the representation beyond the size of the dictionary, and incorporate the lost information into the added dimensions in a form that describes the difference between each dictionary codeword and the descriptor to be encoded. The Fisher Vector (FV) [62] and the Super Vector [83] are such representations that found wide use in visual recognition applications.

In [10] a comprehensive comparison study put these different image representations

to the test. In a fair playing field of the same local features (SIFT), and a linear SVM classifier for each representation technique, the experiments concluded that the Fisher Vector's performance surpassed all others by a significant margin in terms of classification accuracy. Thus, for this thesis study, the Fisher Vector is used in all experiments as the representative handcrafted representation technique. The following section describes the FV framework in detail.

2.3 The Fisher Vector

The Fisher Vector (FV) representation is the state-of-the-art approach to handcrafted representations based on the pooling of local features. The main idea was first published in [42] in the form of the Fisher Kernel, a general way of deriving a discriminative kernel from a generative model of data. It is favored over other kernels due to its enhanced discriminative power, as well as being a comparison metric between examples, naturally induced from the model.

The reign of Fisher Vectors lasted for quite a long time. The 2012 edition of the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC2012) [66] was populated largely by FV based methods, even though they were all demolished by the sole learned representation-based entry, the AlexNet [48]. Even as Convolutional Neural Networks started to dominate visual recognition competitions, the Fisher Vector was kept on life support for a while longer, with innovations like deep Fisher networks [68].

In this section, first, the Fisher kernel (FK) will be introduced as a valid kernel to measure the similarity between two features, and the Fisher vector is derived. Then, the closed form representations for the Fisher vector coefficients will be derived for our specific problem. Finally, improvements towards better classification performance, as well as tricks of the implementation will be explored.

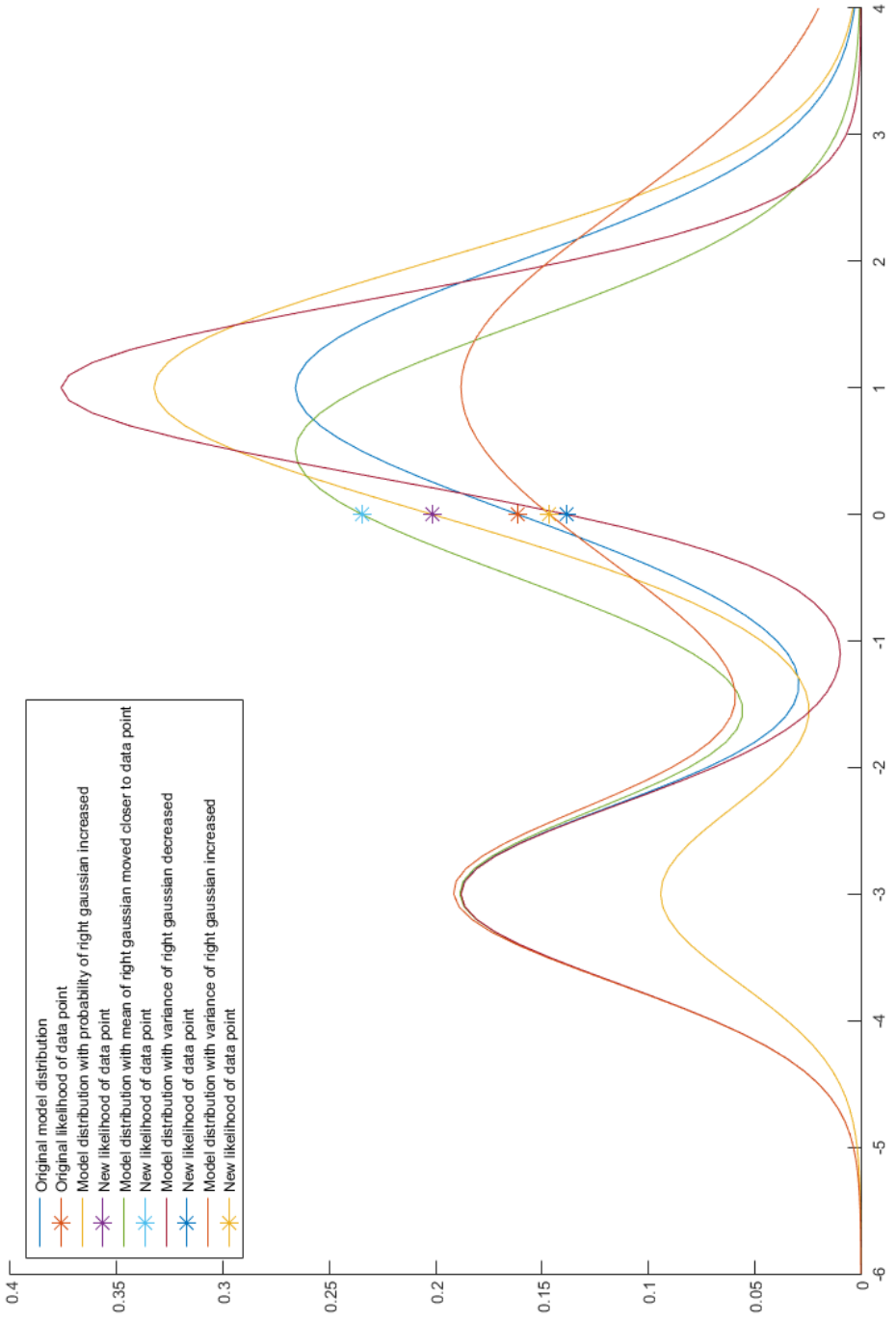

Figure 2.3: A 1-dimensional example showing how changes in model parameters affects the likelihood of data.

2.3.1 The Fisher Kernel

The Fisher kernel induces a similarity measure between features generated by the same generative model, defined as $p(\mathbf{x}|\Theta)$ for the original local feature space, where Θ indicates model parameters. The contribution of each parameter to the generation of a particular feature $\mathbf{x}$ can be described by the partial derivative of the log-likelihood with respect to said parameter; so, the contribution of all parameters can be described by the gradient vector as follows:

$$\hat{\mathbf{\Phi}}(\mathbf{x}) = \nabla_{\Theta} \log p(\mathbf{x}|\Theta). \tag{2.5}$$

The gradient is a valid representation of $\mathbf{x}$, since we may infer the location of $\mathbf{x}$ in the original feature space by how the generative model parameter changes its likelihood. For instance, if the generative model chosen is a Gaussian mixture model (GMM), gradient with respect to the weight parameters gives the same information as the BoV representation with soft assignments [60]. A toy example for 1 dimensional data modeled by 2 Gaussian components is given in Figure 2.3.

From Figure 2.3, we first observe that changes in the parameters of one Gaussian affects mostly the likelihood of data that it describes, i.e., data that is closer to its mean than others. This results in sparse representations. We also see that the changing the weight parameter of one Gaussian results in a more distributed effect on the entire feature space, since the weight parameters are controlled to sum up to 1, in order to have a valid probability distribution. The change in the mean affects the likelihood of data in a simple fashion: as the mean is moved closer to a point, the likelihood of that point increases. The change in variance has a more nuanced effect; in our example, for the indicated point $p(x|\sigma)$ is at a local maximum, so both increasing and decreasing the variance decreases the likelihood of this point.

To obtain the FK distance between two feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, we compute:

$$K(\mathbf{x}^i, \mathbf{x}^j) = \hat{\mathbf{\Phi}}(\mathbf{x}^i)^T \mathbf{H}^{-1} \hat{\mathbf{\Phi}}(\mathbf{x}^j), \tag{2.6}$$

where the $\hat{\Phi}$ vectors are the gradients as computed in (2.5), and $\mathbf{H}$ is the Fisher infor-

mation matrix, defined as:

$$\mathbf{H} = E_{\mathbf{x}}\{\hat{\mathbf{\Phi}}(\mathbf{x})\hat{\mathbf{\Phi}}(\mathbf{x})^T|\Theta\}. \tag{2.7}$$

The gradient multiplied by the Fisher information matrix is called the *natural gradient*, and has the effect of restricting the gradient to directions that can be taken on the manifold defined by the class of generative models [42]. The Fisher kernel is the inner product of the ordinary gradient representation with the natural gradient representation.

Finally, we observe:

$$K(\mathbf{x}^i, \mathbf{x}^j) = \hat{\mathbf{\Phi}}(\mathbf{x}^i)^T\mathbf{H}^{-1}\hat{\mathbf{\Phi}}(\mathbf{x}^j) \tag{2.8a}$$

$$= \hat{\mathbf{\Phi}}(\mathbf{x}^i)^T\mathbf{H}^{-1/2^T}\mathbf{H}^{-1/2}\hat{\mathbf{\Phi}}(\mathbf{x}^j) \tag{2.8b}$$

$$= (\mathbf{H}^{-1/2}\hat{\mathbf{\Phi}}(\mathbf{x}^i))^T(\mathbf{H}^{-1/2}\hat{\mathbf{\Phi}}(\mathbf{x}^j)). \tag{2.8c}$$

In other words, computing the FK distance of two features is equivalent to computing the inner product of their Fisher vector representation, defined as:

$$\mathbf{\Phi}(\mathbf{x}) = \mathbf{H}^{-1/2}\hat{\mathbf{\Phi}}(\mathbf{x}). \tag{2.9}$$

The FV representation of an image is obtained by pooling the patch features in the high dimensional FV feature space. Also note that the process of going from the gradient vector to the FV is simply a whitening transformation. [47].

2.3.2 Derivation of the Closed Form Fisher Vector

In the generic FV framework, the D-dimensional patch features of images are modeled as samples generated from a Gaussian mixture model (GMM), meaning that the distribution of features in the original feature space is described as a sum of Gaussian distributions. In this generative model, the likelihood at any point in the original feature space is given as:

$$p(\mathbf{x}|\mathbf{\Theta}) = \sum_{i=1}^{N} w_i p_i(\mathbf{x}|\mathbf{\Theta_i}), \tag{2.10}$$

where Θ designates the model parameters, whereas Θ_i'is the portion of the parameters for the i'th Gaussian only; N is the number of Gaussians, w_i stands for the weight of the i'th Gaussian, and p_i's are given by:

$$p_i(\mathbf{x}|\mathbf{\Theta_i}) = \frac{1}{(2\pi)^{D/2}(\det \Sigma_i)^{1/2}} \exp[-\frac{1}{2}(\mathbf{x} - \mu_\mathbf{i})^T \Sigma_i^{-1}(\mathbf{x} - \mu_\mathbf{i})]. \tag{2.11}$$

To ease calculations, the covariance matrix Σ_i's are assumed to be diagonal, i.e., $\Sigma_i = diag(\sigma^\mathbf{2}{}_i)$. The parameters of the GMM are made up of the mean and variance vectors, as well as the weight of each Gaussian:

$$\mathbf{\Theta} = (\mu_i, \sigma^\mathbf{2}{}_i, w_i : i = 1, 2, ..., N). \tag{2.12}$$

For each Gaussian distribution contributing to the mixture, we can write the gradient of likelihood as:

$$\nabla_{\Theta_i} p(\mathbf{x}|\Theta_i) = p(\mathbf{x}|\Theta_i)\mathbf{g}(\mathbf{x}|\Theta_i), \tag{2.13a}$$

$$\mathbf{g}(\mathbf{x}|\Theta_i) = \begin{bmatrix} \mathbf{g}(\mathbf{x}|\mu_i) \\ \mathbf{g}(\mathbf{x}|\sigma^2{}_i) \end{bmatrix}, \tag{2.13b}$$

$$\mathbf{g}(\mathbf{x}|\mu_i)_j = \frac{x_j - \mu_{i,j}}{\sigma^2{}_{i,j}}, \tag{2.13c}$$

$$\mathbf{g}(\mathbf{x}|\sigma^2{}_i)_j = \frac{1}{2\sigma^2{}_{i,j}}[(\frac{x_j - \mu_{i,j}}{\sigma^2{}_{i,j}})^2 - 1], \tag{2.13d}$$

and the gradient of log-likelihood for the mixture distribution as:

$$\nabla_{\Theta_i} \log p(\mathbf{x}|\Theta) = q_i(\mathbf{x})\mathbf{g}(\mathbf{x}|\Theta_i), \tag{2.14a}$$

$$q_i(\mathbf{x}) = \frac{w_i p_i(\mathbf{x}|\mathbf{\Theta_i})}{\sum_{i=1}^{N} w_i p_i(\mathbf{x}|\mathbf{\Theta_i})}. \tag{2.14b}$$

If the model describes the data well, then most of the examples will be located near a center of some Gaussian. Moreover, if the dimensionality of data is large enough, the Gaussians will be separated from each other by a good amount of distance. So, $q_i(\mathbf{x})$ likely can be approximated as equal to 1 for a single Gaussian component and 0 for the others; essentially assigning that portion of the feature space to that Gaussian, similar to the hard assignments in the BoV method. Under this approximation, the Gaussians are already uncorrelated, making the Fisher information matrix a diagonal one, whose parameters can be calculated as:

$$H_{\mu_{i,j}} = \frac{w_i}{{\sigma^2}_{i,j}}, H_{{\sigma^2}_{i,j}} = \frac{w_i}{2{\sigma^4}_{i,j}}, \tag{2.15}$$

and finally, we make use of the general FV formula at (2.9) to obtain the final FV representation for our model:

$$\Phi_{\mu_{i,j}}(\mathbf{x}) = q_i(\mathbf{x})\frac{x_j - \mu_{i,j}}{\sqrt{w_i}\sigma_{i,j}}, \tag{2.16a}$$

$$\Phi_{{\sigma^2}_{i,j}}(\mathbf{x}) = \frac{q_i(\mathbf{x})}{\sqrt{2w_i}}[(\frac{x_j - \mu_{i,j}}{{\sigma^2}_{i,j}})^2 - 1]. \tag{2.16b}$$

The final FV representation for a point feature is the concatenation of these coefficients for all values of i and j. To obtain the representation for the whole image, we use average pooling. It should be noted that the weight terms are omitted here, since they are very often left out of practical FV implementations as they do not introduce any additional discriminative information, as evidenced by a lack of increase in classification performance when they are used.

2.3.3 Improving the Fisher Vector

In [62] two improvements to the basic FV were proposed. The new formulation, named "improved Fisher vector" (IFV), involves:

1. Using Hellinger's kernel to compute distances,

2. L2 normalization.

When used together, these two ideas were shown to significantly increase performance. The relation between the original and the improved formulation goes, for each element of a given FV:

$$\Phi_k^I(X) = sign(\Phi_k(X))\sqrt{\frac{|\Phi_k(X)|}{\sum_k |\Phi_k(X)|}}. \tag{2.17}$$

Another improvement, which may be applied to all pooled local representations, is to incorporate some weak spatial information into the model by first dividing the image into a few slices, and computing separate FV representations for each slice, as well as for the whole image. The final representation, named **spatial pyramid**[49], is obtained by concatenating all of those FV representations into a single vector.

2.4 Learned Representations

The success of handcrafted local feature based methods is evident: in the PASCAL VOC competition that ran yearly between 2007-2012, such methods dominated the classification contest [23]. The best performing method throughout the years of the competition came in its last year, when National University of Singapore reported 82% mean accuracy with their method partly based on [82]. The move from the PASCAL VOC to the new competition based on the huge ImageNet dataset, ILSVRC [66], coincided with the rebirth of deep learning. The entries to last year's ILSVRC consisted entirely of a roster of deep networks, a trend that does not look to change soon.

Even though the PASCAL VOC competition missed the rebirth of deep learning, many works on classifying images with deep networks still report their results on the VOC datasets. A comparable study in [58] reports an improvement to 83% mean accuracy, which marks the end of the dynasty of classical, handcrafted methods in the task of classification. It should be noted that deep networks trained on the much larger ImageNet dataset, then fine-tuned on the PASCAL VOC training set can achieve even better accuracy [32].

Representation learning is not synonymous with deep learning. Nevertheless, in this

thesis work learned representations almost exclusively refer to deep artificial neural networks (ANN), due to the significant difference in performance in visual recognition related tasks - while shallow nets can be brought to the same capacity (in terms of the range of functions/transformations that can be accomplished with said network), empirical findings indicate that they cannot be trained to leverage the added capacity as easily [20]. To add insult to injury, one rare study that reports competitive performance levels with a shallow network accomplishes this by teaching the shallow net to *mimic* [1] a deep network [5].

We further refine our focus to consider for the most part only networks trained via supervised training methods. It is not a reasonable approach to expect a computer to find out the difference between classes by itself, and initializing networks with weights learned via unsupervised training is not found to improve performance compared to purely supervised training with proper random weight initialization - in fact, it may even **hurt** performance [50].

2.4.1 The Artificial Neuron

The basis of an ANN is the artificial neuron, which is in turn based on its biological counterpart (Figure 2.4). The biological neuron is a simple processing unit that takes input signals from other neurons connected to its dendrites and outputs a response accordingly. Neurons are adaptable; they can adjust themselves to perform different operations on the input data depending on the task [77]. The real power of neurons come from sheer numbers; on average, the human brain is estimated to consist of 86 billion neurons [4]. The goal of deep learning is not to create processing units that model biological neurons as accurately as possible; but to make use of the idea that layers upon layers of simple, adaptable processing units can solve a large variety of difficult problems. Therefore, while initially inspired by the biological neuron, the basic unit of an ANN has evolved in a different direction than what current biological models suggest.

The modern artificial neuron does resemble its biological counterpart in terms of its

[1] Instead of using class labels, a network can be trained with the output values of another network. The optimization then becomes a regression problem between the outputs of the *student* and the *teacher* network. In layman's terms, the student tries to mimic the teacher.

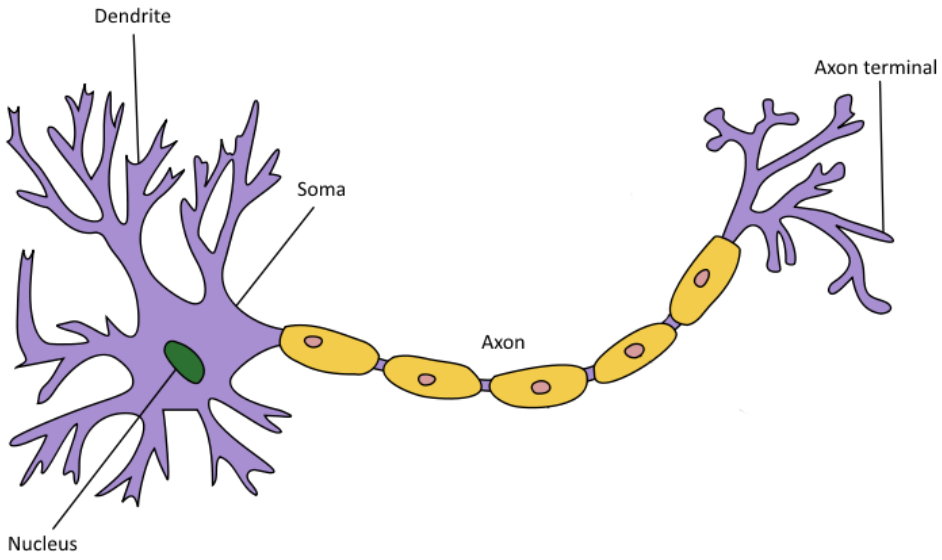

Figure 2.4: The biological neuron (simplified). Inputs from other neurons, received by the dendrites, are processed by the soma and transmitted to other neurons via the axon. Taken from http://wikimedia.org

functionality. It computes a weighted sum of its inputs, adds a bias value to the sum and finally applies a non-linear function to this value to generate the output (Figure 2.5). In mathematical terms, the output y in terms of the inputs x_i's, the neuron weights w_i's and the bias b:

$$y = g(b + \sum_i w_i x_i), \tag{2.18}$$

where $g(.)$ is a (generally) non-linear function called the activation function. With proper selection of the activation function, a feedforward ANN with a single hidden layer with an appropriate number of neurons can approximate any smooth function, giving the ANN's the *universal approximator* property [39]. For this property to hold, the activation function needs to be one that *squashes* its inputs, e.g. to between 0 and 1 as in the case of the sigmoid function. It is vital to note that this gives us no guarantees about whether we can find the solution for the parameters for the ANN that approximates a given function; only that a solution exists. Nevertheless, this property, among others, has led to squashing functions being used predominantly in earlier, shallow ANN's.

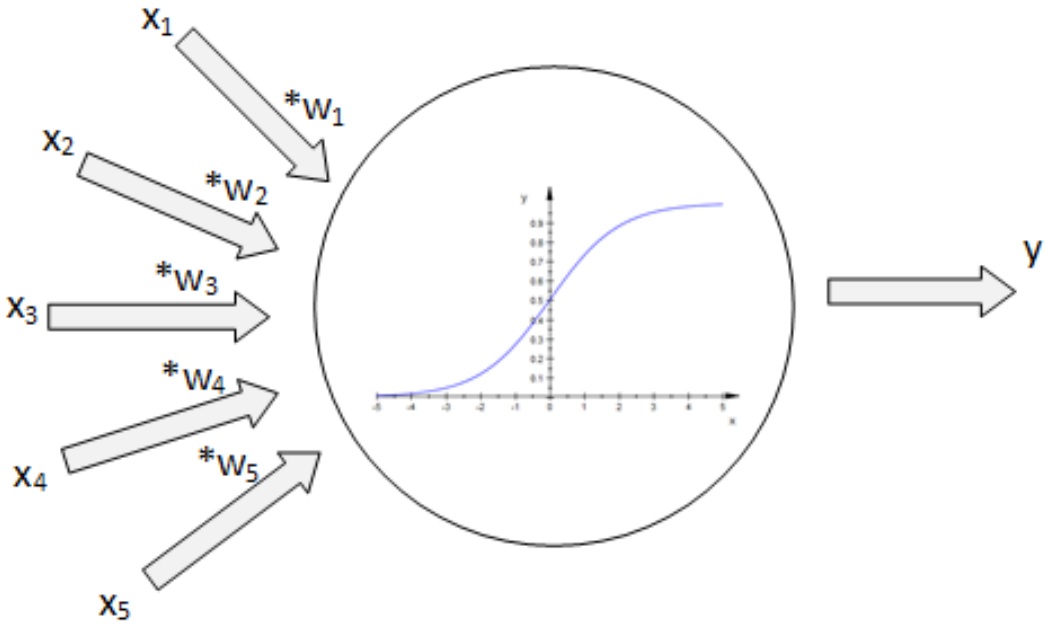

Figure 2.5: The artificial neuron. Its output is a non-linear function of a biased, weighted sum of its inputs.

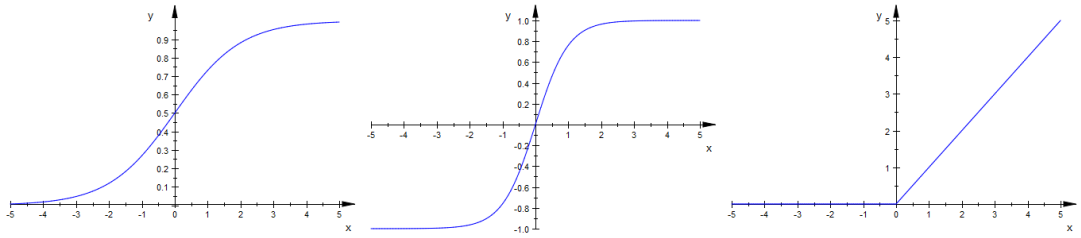

Figure 2.6: Commonly used activation functions. From left to right: sigmoid, hyperbolic tangent, and the rectifier function.

Some of the properties to look for in an activation function are (piecewise) differentiability, monotonicity, and saturated outputs. Differentiability is especially critical, since the primary algorithm for training ANN's, backpropagation (Section 2.4.2.2), depends on analytical differentiation.

Some of the commonly used activation functions are (Figure 2.6):

- The sigmoid function:

$$g(u) = \frac{1}{1 + e^u}, \tag{2.19}$$

  The sigmoid function maps all inputs to a value between 0 and 1, so it works

well as a probability estimate.

- The hyperbolic tangent:

$$g(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}, \tag{2.20}$$

  The output of the hyperbolic tangent function is unbiased, i.e. if the input data is generated from a symmetric distribution with zero mean, the output distribution will also be symmetric with zero mean.

- The rectifier function:

$$g(u) = max(0, u). \tag{2.21}$$

  A neuron with rectifier activation function is called a Rectified Linear Unit(ReLU) [55].

ReLU and its variants (Leaky ReLU, Parametric ReLU [36], etc.) are almost exclusively used in state of the art deep learning systems [81], even though the rectifier is not a squashing function. The main reason is that networks of ReLU's do not come across the problem of *vanishing gradients*[38] during training: if the input of a squashing function grows to be too large, the output becomes practically invariant to small changes in the parameters of the neuron because the output has saturated. Unsurprisingly, it comes down to which type of activation function can be trained more easily, rather than a problem of capacity. The exception is the linear activation function, i.e. $g(u) = u$, since that does not extend the capacity of the network beyond linear combination of its inputs. For an ANN with a single hidden linear activation layer:

$$y = b_1 + \sum_i (b_{0,i} + \sum_j w_{j,i} x_j), \tag{2.22a}$$

$$= (b_1 + \sum_i b_{0,i}) + \sum_j x_j \sum_i w_{j,i}, \tag{2.22b}$$

Define a new bias term, b^*, and new weights w_j^*:

$$b^* = b_1 + \sum_i b_{0,i}, \tag{2.23a}$$

$$w^*{}_j = \sum_i w_{j,i}, \tag{2.23b}$$

$$y = b^* + \sum_{i,j} w_j^* x_j, \tag{2.23c}$$

which is equivalent to a single neuron with a linear activation function.

2.4.2 Feedforward Neural Networks

A feedforward neural network consists of layers of several artificial neurons (Figure 2.7). Subsequent layers are typically fully connected to each other; the outputs of a layer of neurons are concatenated into the input vector for all units belonging to the next layer.

In the standard notation of feedforward ANN's, there exist two special layers - namely the input and output layers. The input layer simply holds the input vector values. The output layer must give some sort of a meaningful value at its output, such as a vector of class probabilities between 0 and 1 for a classification problem. For this reason, the sigmoid function is often used for the activation of the output layer. The layers in between, usually layers of ReLU's, are called hidden layers.

The layered formulation of the feedforward ANN allows us to use matrix notation to describe input-output relation of a whole layer in one go. Even more importantly, the matrix based approach makes very large and deep networks feasible [12]. Many of the computing tools used in deep learning (Caffe [43], Theano [7], Torch to name a few) make use of the Basic Linear Algebra Subroutines (BLAS) library to compute matrix multiplication efficiently. Furthermore, implementations to perform ANN computations on widely available Graphical Processing Units (GPU) exist which introduces multiplicative increases in performance. This is made possible due to the parallelizable nature of ANN computations [12], both over units and examples.

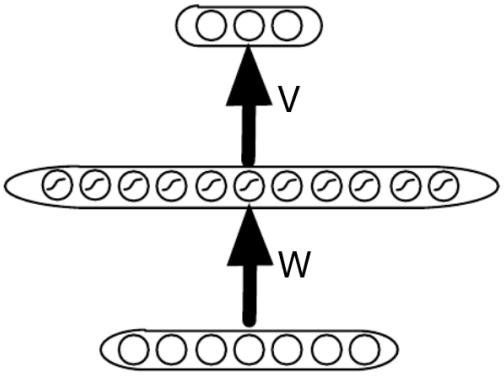

Figure 2.7: A shallow feedforward neural network with a single hidden sigmoid layer. Taken from [6].

For the network depicted in Figure 2.7, the output can be written, in matrix notation, in terms of the inputs and the parameters of the network as:

$$\hat{\mathbf{y}} = sigmoid(\mathbf{b_1} + \mathbf{V} * sigmoid(\mathbf{b_0} + \mathbf{W}\mathbf{x})), \tag{2.24}$$

where $\hat{\mathbf{y}}$ and $\mathbf{x}$ are the output and the input vectors, $\mathbf{W}$ and $\mathbf{V}$ respectively are the weight vectors of the hidden and the output layers, and $\mathbf{b_i}$'s are the bias vectors for the hidden and the output layer. The sigmoid function is an elementwise operation as defined by Equation 2.19. One may notice that while this direct output-input relationship looks simple for a shallow ANN, it quickly becomes convoluted as the number of hidden layers are increased, even though the layerwise input-output relation remains the same.

2.4.2.1 The Loss Function

To fully define a complete framework of a learning system for an ANN, one needs a loss/cost function and a training method in addition to the input-output relationship (Figure 2.8). The loss function depends on the problem to be solved by the ANN. The following types of loss functions are common in supervised training with labels for classification problems:

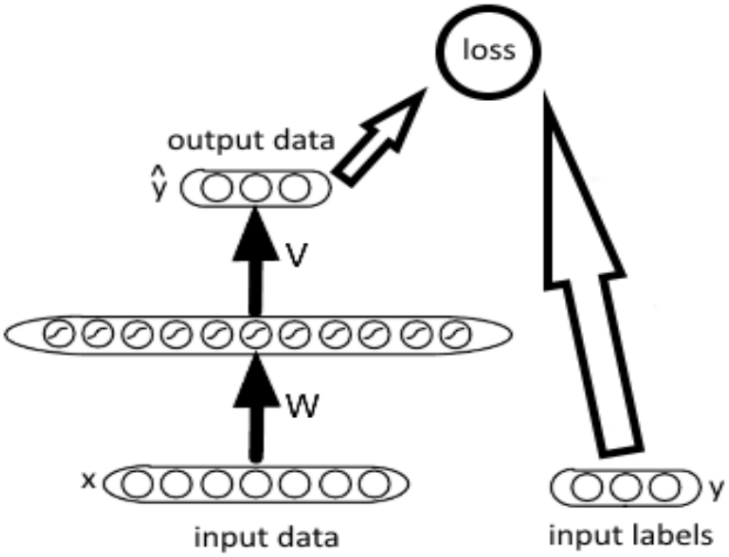

Figure 2.8: The complete framework for a feedforward ANN.

- **Logistic loss:**

$$L = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]. \tag{2.25}$$

The logistic loss is only applicable to binary classification problems, such as classifying images of tumors as benign or malignant. For such problems, the label for each input is a binary scalar, and the output of the ANN is always a value between 0 and 1. We can interpret the output as answering a yes/no question with some confidence: if the output is higher than 0.5, the answer is yes; and as the output gets closer to 0 or 1 rather than somewhere in the middle, we can say that the ANN gets more and more confident in its answer.

Investigating the loss function, one may observe that it harshly punishes misclassifications where the ANN is confident in its answer, as the negative of the output of one of the logarithms diverges towards infinity.

There are two extensions to the logistic loss for multiclass problems, depending on the mutual exclusivity of the problem, which are listed below.

- **Multinomial logistic loss:**

$$L = -\log \hat{y}_k, \tag{2.26}$$

where $\hat{y}_k$ is the output probability value that corresponds to the correct class for the given input. The multinomial logistic loss is used if the classes are mutually exclusive,

i.e. each input datum belongs to a single one of many classes. In this case, the output is a vector of probabilities that sums up to 1 to satisfy the axioms of probability: there exist N mutually exclusive events (a given datum belonging to class number k,$1 \leq k \leq N$) which covers the whole sample space, so their probabilities must sum up to 1.

The summation to 1-property is critical; otherwise, since the output values for other classes are disregarded in computing the loss function, the optimal solution becomes an output of all 1's. In practice, this property is enforced by subjecting the outputs of the output sigmoid layer to a softmax function:

$$\hat{y}_k^* = \frac{\hat{y}_k}{\sum_{n=1}^{N} \hat{y}_n}. \tag{2.27}$$

- **Sigmoid cross-entropy loss:**

$$L = -\frac{1}{n} \sum_{n=1}^{N} [y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)]. \tag{2.28}$$

The sigmoid cross-entropy loss is useful when classes are **not** mutually exclusive, i.e. an input datum may belong to more than one class, or possibly even none of the classes. In this case, the problem can be stated as N binary classification problems, i.e. N yes/no questions, with N being the number of classes. In that case, the input labeling can be described by a binary vector, and the loss function becomes the average of N logistic losses.

2.4.2.2 Training Neural Networks

Learning in ANN's corresponds to optimizing the parameters of each layer for given topology. Topology, or architecture, of an ANN refers to its number of layers, as well as the number and type of neurons in each layer. This definition presumes that the network has a layered structure, which is true for the networks studied in this work; an arbitrary topology can also be defined by the properties of each neuron and their interconnections [27]. The topology of an ANN is predetermined; it defines a class of functions that can be realized by the network, as well as a flow graph of computations to implement those functions. Parameters that define the topology

of an ANN are hyperparameters; they are not part of training but can be optimized according to which topology performs best on a validation set.

The optimization is generally performed by applying Gradient Descent to the loss function:

$$\theta^* = \theta - \alpha \nabla_\theta L(\theta; [\mathbf{X}, \mathbf{y}]), \tag{2.29}$$

where $L(\theta; X)$ is the loss function in terms of ANN parameter vector θ, array of input vectors $\mathbf{X}$ and the corresponding labels $\mathbf{y}$, and α is the learning rate that dictates how far the parameters are updated with respect to their previous values. This update is performed iteratively until the stopping condition is reached.

It should be noted that the ultimate goal of learning is not to minimize the loss with respect to the training data; but to maximize performance on the test data. This changes the stopping condition of optimization: instead of stopping the training when a local minimum is reached, the ANN with current parameter values are tested on validation data periodically. The training is stopped when the performance on the validation data starts to diminish due to overfitting, as seen in Figure 2.9. The assumption is that the training data is a good sample of the general data distribution; so optimizations over training data are likely to generalize to previously unseen data - up to some point where the learner starts to memorize minute details about training data that does not generalize. This strategy is called early stopping [63] (Figure 2.9).

Another note is regarding the non-convexity of the loss function of ANN's. This property makes it so that smaller networks' convergence is unreliable; that is, their performance fluctuates wildly depending on the values of paramaters during initialization; but in deeper networks most local minima have similar errors on test data, and "bad" critical points are exponentially more likely to be saddle points [19] [14]. Given that for learners the real optimization goal is not to find the global optimum anyway, the non-convexity ceases to be a problem for deep ANN's.

For the huge size of datasets deep learning requires, this formulation of gradient descent may not be sensible; a single update of parameters requires calculating the gradient of the loss with respect to the whole array of inputs. A specific feature of

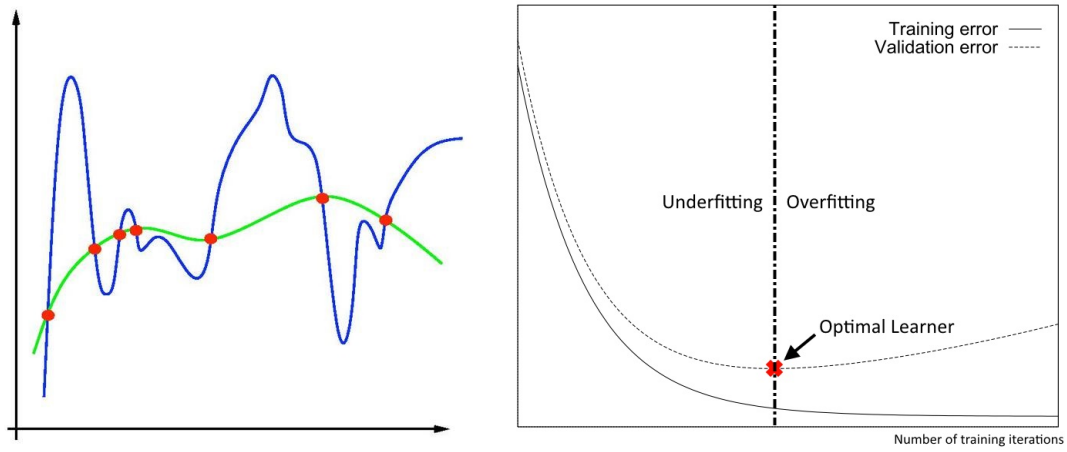

Figure 2.9: To the left: demonstration of overfitting to training data, to the right: demonstration of early stopping.

optimizing ANN's is that the loss functions can be defined in terms of a single input example, and the true loss with respect to the whole dataset becomes the average of per-example losses; so it is possible to compute gradients in terms of a fraction of the dataset. This method of optimization is called mini-batch gradient descent [16]. GPU based implementations of ANN's bounds the size of mini-batches from both sides: a number too small will waste processing power due to not enough parallelization, while a number too large will cause the whole data to exceed the GPU memory size.

2.4.2.3 Backpropagation

Backpropagation is the name of the method for computing gradients in ANN's, as required for the optimization process. It is an optimal algorithm in terms of computational complexity, that is, no other algorithm is faster in O(.) notational sense [6]. It is an application of the chain rule in calculus:

$$\frac{\partial f(\vec{v}, x)}{\partial x} = \sum_i \frac{\partial f(\vec{v}, x)}{\partial v_i} \frac{\partial v_i}{\partial x}. \tag{2.30}$$

Now, let us apply the chain rule to a very simple "network", consisting of an "ADD"

and a "MULTIPLY" neuron:

$$q = 3x + y, \tag{2.31a}$$

$$f = qz. \tag{2.31b}$$

Then, the gradients for each neuron are:

$$\frac{\partial f}{\partial z} = q, \frac{\partial f}{\partial q} = z, \tag{2.32a}$$

$$\frac{\partial q}{\partial x} = 3, \frac{\partial q}{\partial y} = 1. \tag{2.32b}$$

Now, since we are really interested in the gradients of the output, we observe that can obtain it from the neuron gradients and the chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = 3z, \tag{2.33a}$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = z. \tag{2.33b}$$

So, just like in a forward pass of a neural network where the outputs of earlier layers are passed on to compute the outputs of the next layers, we can calculate the gradients with respect to the inputs of earlier layers by making use of the gradient values for the following layers. During the computation of gradients, everything is backwards: first, the gradients with respect to the output layer inputs are calculated, which are propagated to the last hidden layer. This process continues until gradients with respect to the first hidden layer weights are computed.

For practically useful ANN's, the process remains exactly the same; only the analytical expressions for neural gradients may get slightly more complicated, depending on the type of neuron. For a neuron with sigmoid nonlinearity, the gradient can be simplified more if written in terms of its output:

$$\frac{df(u)}{du} = f(u)(1 - f(u)). \tag{2.34}$$

This means that the gradient can be computed much more efficiently if the output value is cached in memory during the forward pass.

2.4.3 Convolutional Neural Networks

2.4.3.1 Introduction

For visual recognition, fully connected ANN based solutions quickly become intractable. For a 200x200 RGB image, a fully connected feedforward network's first hidden layer neurons will each have 200x200x3=120,000 weights. This, multiplied by the number of neurons, makes for an enormous amount of parameters. When we also add in the other layer parameters, we suddenly enter the realm of the curse of dimensionality, which means that the further increases in the number of parameters start to decrease the reliability of the estimation of their optimal values [41]. Convolutional Neural Networks (ConvNet) are the solution to this problem.

ConvNets are different from fully connected feedforward ANN's in terms of connections between each layer. Unlike the fully connected networks, ConvNet neurons connect only to a local neighborhood of neurons. These neighborhoods are defined in terms of two dimensional coordinates. While we describe the activations at each layer of a fully connected network as a vector, which works well for ordinary, one dimensional inputs, for ConvNets we jump to three dimensions to incorporate the geometry of the data (Figure 2.10). Colored image data is three dimensional - traveling in the third dimension (depth) corresponds to moving from one color space to another.

Working in this geometry, we now constrain the neuron inputs to only have local connections in width and height, but not in depth. For instance, we may restrict each neuron of the first hidden layer to be connected only to a 5x5 neighborhood, for all color channels. Each neuron in the first hidden layer will have 5x5x3=75 inputs, as opposed to hundreds of thousands. Except for the sides of the image, there is a one to one correspondence between a neuron in the next layer, and one in the previous layer, which happens to be the center of its neighborhood.

This local, sparse connectivity approach is not just a result of extreme computational

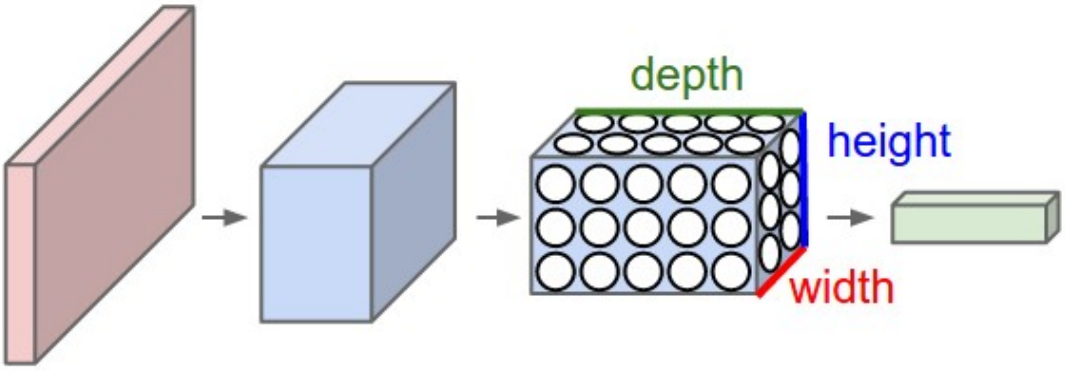

Figure 2.10: Layers of a ConvNet are volumes, not vectors. The first volume is the input image, whose third dimension corresponds to different color spaces. Taken from http://cs231n.stanford.edu

costs of fully connected networks; it also introduces a sensible prior: transformations of a small region of an image describe well the low level properties of an image, which can then be pooled into high level representations. Pooled local representations based on features such as HOG [18] and SIFT [53], and ConvNets share this prior.

ConvNets and pooled local representations also have the same parameter sharing property: SIFT is applied to every single patch of image the same way regardless of its location within the image. Similarly, sets of neurons in a convolutional layer apply the same transformation to their respective regions because their weights are shared across the set. Variance of low level representations is achieved by increasing the depth of the layer: a convolutional layer of depth D will have D sets of neurons with shared weights, but connected to different locales in the image. Parameter sharing introduces another prior, namely that local features ought to be invariant to translation. With this prior, each set of neurons tells us of the locations that satisfy some property described by the weights shared by those neurons. Moreover, the number of parameters per layer is reduced even further: For a $W * H * D$ layer connected to a previous layer of $w * h * d$ width, height and depth, we only have $N * M * d * D$ parameters, where N and M are the number of local connections in width and height. For the first hidden layer, if $N = M = 3$, $d = 3$ (i.e. input images are described in a tricolor space), $w = h = 200$ (input images are 200x200), and $D = 10$; the number of parameters would be 270, which is quite reasonable. For comparison, a fully connected layer of the same size would have $whdWHD = 48,000,000,000$

different parameters!

Deeper convolutional layers in a network create a hierarchy of increasing abstraction in representations. While neurons in deeper layers are still locally connected, they are indirectly affected by a larger and larger number of neurons from previous layers. Even networks with every neuron locally connected a very small region, at some depth the output of neurons will start to depend on all inputs. The herd of volumes on which the output of a neuron depends is called the *receptive field* of a neuron (Figure 2.11).

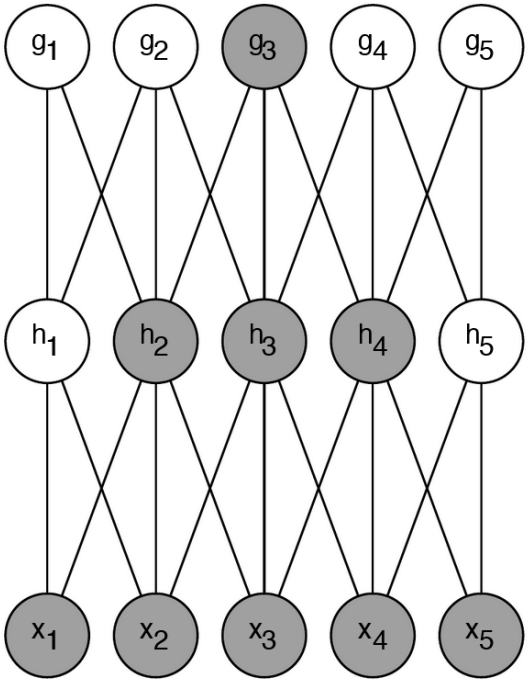

Figure 2.11: Receptive field of a neuron in a sparse, locally connected network. Taken from [6].

2.4.3.2 Layers of a ConvNet

Typically, a ConvNet consists of a number of convolutional layers followed by a few fully connected layers. The fully connected portion can be considered as a separate feedforward ANN which takes the output volume of the final convolutional layer as its input - the volume is flattened into a vector to facilitate this transition. The fully connected part is fairly standard, consisting of ReLU's.

The convolutional layers are more complex, and perhaps better understood in stages

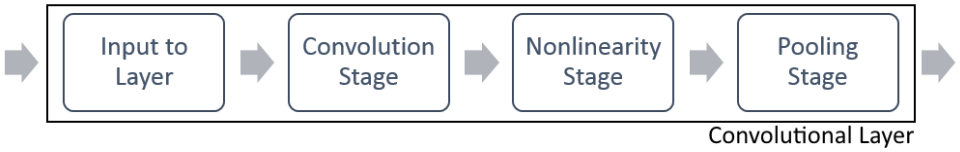

Figure 2.12: A convolutional layer can be understood as a collection of simple layers.

of several simple layers (Figure 2.12): a convolution stage, followed by a nonlinearity, often followed by a pooling layer. Type of nonlinearity and existence/type of pooling operation are predetermined as part of the network architecture; only convolution weights are optimized, again typically via backpropagation based mini-batch gradient descent. The duo of convolution followed by nonlinearity is in essence the same as the typical artificial neuron; but local connectivity combined with parameter sharing has a significant mathematical implication: the operation becomes a two dimensional convolution, as in Eqn. 2.35.

$$S[x, y] = (I * H)[x, y] = \sum_{l=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} I[x - k, y - l]H[k, l]. \tag{2.35}$$

If each neuron is locally connected to a $N * M$ area, the filter $H[x, y]$ is nonzero only in an $N * M$ area. In almost all cases $N = M$, and is typically 3, 5 or 7. So, the first convolutional layer corresponds to passing the image through a variety of small linear, translation invariant filters.As usual, the following nonlinearity is often rectification.

The goal of the pooling stage is to shrink the size of the output volume of the layer by pooling the outputs of several neurons into a single value. This keeps the size of the network as a whole, and consequently the number of parameters, at a manageable level while the network depth increases in terms of the number of convolutional layers. The pooling method of choice is universally max-pooling:

$$P[x, y] = \max_{k,l} S[Mk : Mk + M - 1, Nl : Nl + N - 1]. \tag{2.36}$$

Usually $M = N = 2$, so the output volume of the nonlinear activation is sliced to $2*2$ portions and the non-maximum values of each slice are removed from the final output volume (Figure 2.13). **No** pooling in depth, i.e. across separate filters, is performed.

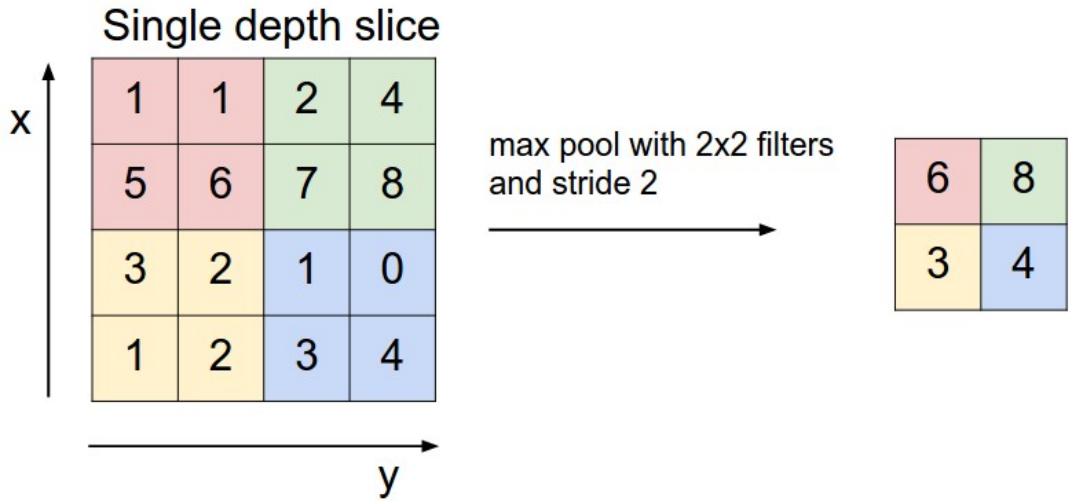

Figure 2.13: Demonstration of max-pooling. Taken from http://cs231n.stanford.edu

2.4.4 Final Remarks

The success of ConvNets is evident: from the tiniest, 32x32 images of the CIFAR-10 dataset [51] to the colossal size of over million images of the ImageNet dataset [69], ConvNet based methods reportedly hold the records for every single one. It is, currently, the state of the art for visual recognition. This may make it convenient to think that the similarity of neural networks to the human visual system makes them a good choice for computer vision related problems. However, one must not forget that the neural network idea was largely abandoned for a long time [67] due to its underwhelming performance compared to other algorithms at the time. It is hard to argue that this similarity offers an advantage over competing algorithms, when we still do not understand the intricacies of how even the most studied parts of the brain works [57]. Ultimately, results drive research - no researcher wants to waste time on a dead-end path, and people are flocking towards the path of deep learning as evidenced by the publications in recent conferences in Computer Vision [1] [28].

Nevertheless, it would be wrong to claim that the similarity is insignificant, as it is not unheard of that a neuroscientific finding can predict the direction of artificial neural network research. A study from year 2000 showed that neurons from auditory processing region of the brain could, when rewired to the visual input, retrain themselves to process visual signals [77]. Nowadays, similar ANN architectures are used to process very different inputs to solve a variety of problems.

Another prediction can be made in terms of the depth of a neural network: there are bounds on computational capacity of the human brain, such as the firing rate of neurons, and we can measure the time taken between seeing an object and recognizing it. By simple multiplication, an estimate on the depth of the human brain's object recognition network can be made. For a firing rate of 200Hz [33], and a recognition time of 0.1 seconds, the depth turns out to be 20, which is close to the depth of the state of the art ConvNets used in image classification and object detection.

CHAPTER 3

HANDCRAFTED VERSUS LEARNED REPRESENTATIONS FOR CLASSIFICATION

In the previous chapter, we established the theoretical framework for different schools of image representation. We identified two distinct approaches in obtaining representations from encodings of handcrafted local features, versus approaches purely based on learning. In this chapter, we compare the two approaches by taking a flagship method from both sides in settings that are as fair as possible for both sides.

Designing the experiments to allow for a fair comparison is no easy task. Attention must be given to a lot of details that make up the algorithms. We have checked our implementations by first reproducing the original experiments done using the methods, achieving similar levels of accuracy. Any deviations from the original results may be due to randomness that is inherent to many parts of the algorithms. As far as we know, no implementation details in the original works are missed in our own implementations. Parts of algorithms were implemented via off-the-shelf packages, which diminishes the possibility of bugs immensely.

Several comparison works of this fashion already exist in machine learning literature, such as [10], [11]. One takeaway from the study of these works is that as the comparison focuses on a specific part instead of a whole classification framework, the experiment setting becomes very clear: fix the rest of the framework in a way that does not favor any of the competing methods, then compare the end results of the whole system [10]. We may be able to come up with some evaluation metric directly for the outputs of the compared methods, but scoring well in such a metric might not necessarily translate to classification accuracy. For two very distinct algorithms, how-

ever, can be much harder to compare in this manner. If two algorithms have nothing in common, all we can do is fix the inputs: we must use the same training and testing data for both algorithms; if we use some sort of data augmentation for one algorithm, we must use the same augmented data for the other.

3.1 Experiment Setup

The final experimental setup consists of input data and labels for training and for testing, and two model definitions. We do not compare several handcrafted algorithms with each other; nor do we compare different ConvNet architectures. For the final comparison, the only variance is in the amount of training data made available to both models. However, in order to have a fair comparison, both models have to well constructed, both with respect to each other and compared to other possible models within their own paradigms. The models must be:

- **Representative of their paradigm** - they must consist of parts that are commonly used in other methods. In other words, we should not use some obscure variant that doubles the test execution time while improving the accuracy by some minuscule amount.
- **Optimal in their hyperparameter selection** - any suboptimal setting on one side but not the other will tip the scales, leading us to wrong conclusions.
- **Comparable in any way possible to each other** - having similar representation size, training time etc.

To satisfy the representativeness condition, we look to the more well-known methods from each paradigm. For handcrafted methods, Fisher Vector based approaches are seemingly still very popular within the community as evidenced by recent conference submissions [61, 21, 46], hence we focus on optimizing a Fisher Vector based framework. For the ConvNet model, we have chosen CaffeNet, a modified version of AlexNet, which has large availability, a reasonable size and very competitive performance.

The methods are compared on two levels of training: at the first level, both models are trained from scratch with the VOC2012 training data comprised of 11,530 images [23]. At the second level, we also make the ILSVRC2014 training data available for both methods. The ILSVRC2014 dataset consists of 150,000 images of 1,000 classes that are different from the 20 VOC classes [66]. During testing, we always use the VOC2012 test data. The test results are uploaded to the VOC2012 evaluation server for official evaluation.

3.2 Implementation Details of Compared Frameworks

3.2.1 The Fisher Vector Framework

An overview of the Fisher Vector classification framework is given in Figures 3.1 and 3.2. Parts of the system that merit some explanation will be discussed here.

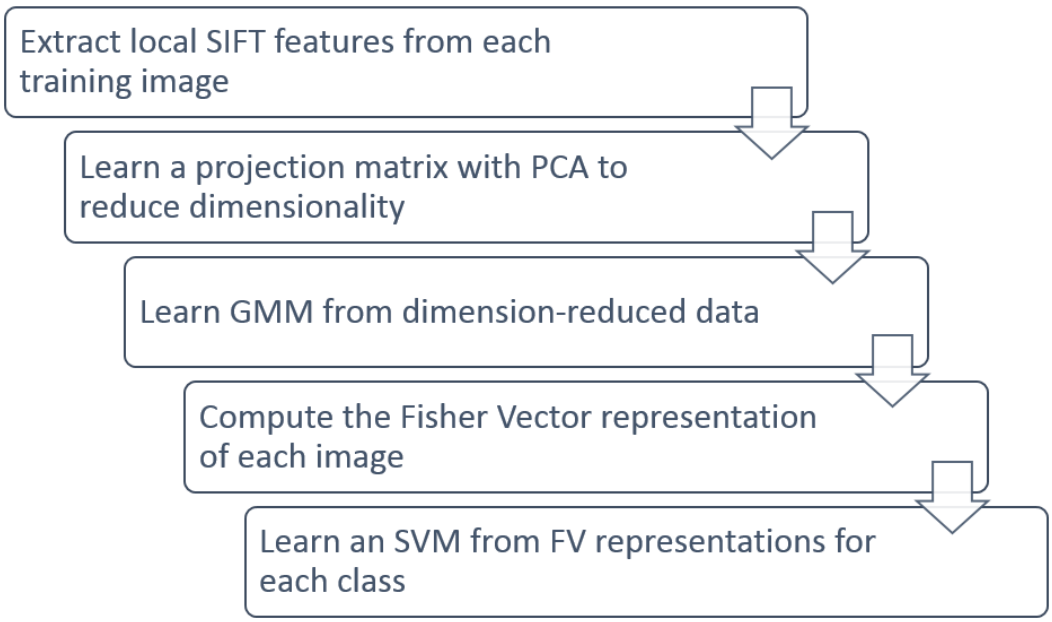

Figure 3.1: The Fisher Vector classification training framework.

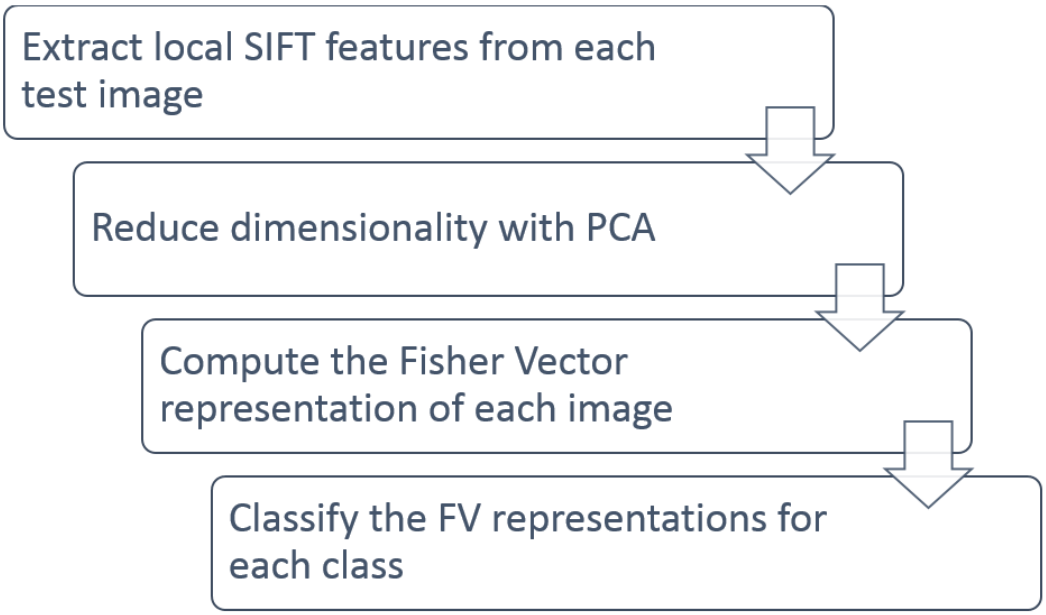

Figure 3.2: Test framework for Fisher Vector classification.

3.2.1.1 Extraction of Local Features

For local feature extraction, there are two main points of contention. The first one is the type of feature to be extracted, whereas the second one is the strategy enforced in the sampling of features.

As discussed in Section 2.2, there are several types of local features that are used in object recognition, with the most prevalent one being SIFT and its variants [53].

The SIFT descriptor of a point describes the local shape around that point using edge orientation histograms. To achieve scale invariance, the image may be rescaled to the scale of the detected keypoint; however, depending on the point sampling strategy, no keypoint detection might be applied. In that case, the image is rescaled to several predetermined sizes. At the desired scale, the image is smoothed with a Gaussian filter, then the SIFT descriptor, a 3-D spatial histogram of the image gradients around the point, is computed.

One serious issue with the SIFT descriptor is the lack of color information - the original SIFT formulation [53] is only for grayscale images. For this reason, some classification systems that make use of the SIFT descriptor supplement it with a color

descriptor. Another approach is to use one of the variants of the original SIFT descriptor, one that incorporates color information without supplement. In [73], colored SIFT variants as well as other color descriptors were empirically compared; it was found that the OpponentSIFT variant performed the best overall.

The OpponentSIFT descriptor first converts the RGB image to the Opponent color space as follows:

$$\begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} = \begin{bmatrix} \frac{R-G}{\sqrt{2}} \\ \frac{R+G-2B}{\sqrt{6}} \\ \frac{R+G+B}{\sqrt{3}} \end{bmatrix}. \tag{3.1}$$

Then, SIFT descriptors are computed on each of the three Opponent color channels. The concatenation of those three descriptors gives us the OpponentSIFT descriptor, which is the one we use throughout this comparison study.

It is shown that for both the original SIFT descriptor and its variants, square rooting of descriptor values and descriptor normalization are both strategies that improve performance [3]. They are simple to implement and have negligible computation cost; so they are used in our implementation.

The second issue of feature extraction is the sampling strategy. The original SIFT methodology includes a keypoint detection stage [53]; however, recent works commonly opt for no keypoint detection. Instead, they uniformly sample points, with a predetermined step size, on several differently rescaled versions of the image. The descriptor is computed for each sample point; then low contrast points are removed due to their unstability. This strategy is known as dSIFT, shorthand for dense SIFT, first used in [8].

3.2.1.2 Principal Component Analysis

The OpponentSIFT descriptors are 3 times as large as the SIFT descriptor; and many of the 384 dimensions are strongly correlated. This is not very desirable, since we assume in our Gaussian Mixture Model that the data dimensions are uncorre-

lated (Readers are referred back to Subsection 2.3.2 for detailed discussion). Moreover, the Fisher Vector space is very high dimensional. The ordinary FV has $2 * numberOfGaussians * descriptorLength$ dimensions. If a $3 * 1$ spatial pyramid[1] scheme is implemented, the dimensionality further increases fourfold. Dimension reduction with Principal Component Analysis(PCA) solves both of these problems together, at a reduced information loss.

With PCA, the data is described by a reduced number of orthogonal basis vectors. The vectors are chosen to be the eigenvectors of the covariance matrix of the data, so that after the projection to the directions of the new basis vectors, the covariance matrix becomes diagonal, i.e. the data becomes uncorrelated. The eigenvectors that correspond to the larger eigenvalues are selected as the basis vectors of the projection, meaning that only the dimensions that describe a smaller portion of variance in the data are eliminated.

3.2.2 The Convolutional Neural Net Framework

An overview of the simpler ConvNet framework is given in Figures 3.3 and 3.4. Nevertheless, some tricks in implementation are vital in achieving good performance with deep networks, which will be discussed here.

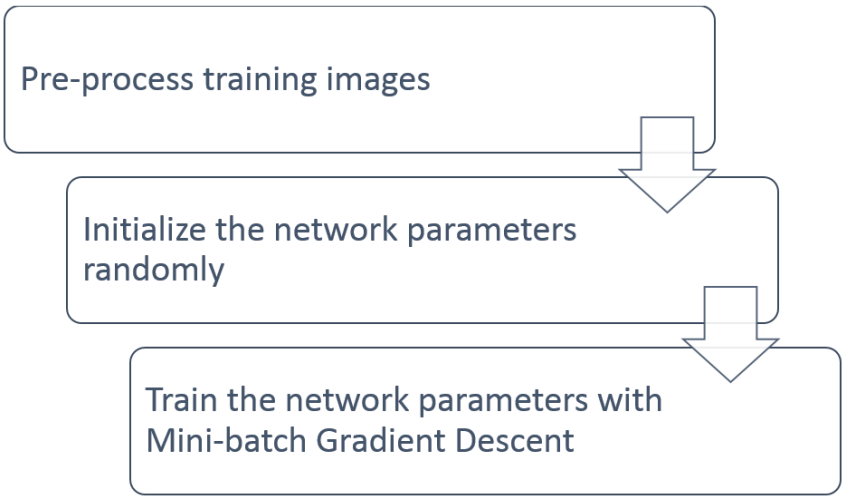

Figure 3.3: Training framework for Convolutional Neural Network classifier.

[1] $In a 3*1 spatial pyramid, the image is divided horizontally into 3 equal parts. The representation of the top, middle and bottom areas a$

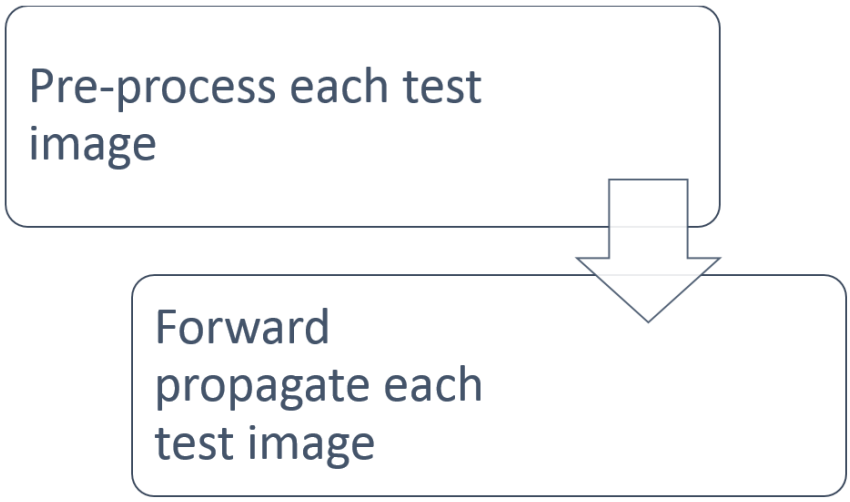

Figure 3.4: Test framework for ConvNet classification.

3.2.2.1 Preprocessing Images

ConvNets only accept images of a predetermined size as inputs. While rescaling an image is a trivial process, the aspect ratio, which varies significantly from image to image, is an issue. The default solution is to discard the original aspect ratio and simply resize the image to the required input size for the network; however, this necessarily causes the image to stretch. Instead of learning the characteristics of an object class from stretched examples, one may first rescale the image such that the shorter side has the same size as the desired input size, then crop off the sides in the other dimension symmetrically to bring the image to the desired size. This strategy opts to give up the information at the sides of the image in favor of the information dependent on keeping the aspect ratio. Experimentally, it has been found that cropping works better than resizing for image classification [11].

It is well established that neural networks work much better with zero-mean data that is appropriately scaled in all dimensions, i.e. if values in one dimension vary between $[-1; 1]$ while another dimension values are in the range of $[-500,000; 500,000]$, we would have a serious problem. Thankfully, image data is limited to a strict range of possible pixel values, e.g. $[0; 255]$ for uint8 images; so we only need to move the images to zero mean. To achieve this, we construct the mean image from by averaging the resized and cropped training images, then subtract this mean image from each training image. During test time, we subtract the same mean image obtained from

training data to the test images.

3.2.2.2 Network Initialization

Initialization of network parameters is a touchy subject. First of all, since network layers are symmetrical constructs, the only way we can break symmetry is by initializing parameters in a layer to different values. In practice, this is done with random initialization, with the initial values of the parameters of the same layer drawn from the same probability distribution. For the actual distribution, we follow the suggestion of [36], using a zero-mean Gaussian distribution with variance $2/n$, where n is the number of weights in a layer.

For the bias parameters, randomness is not necessary since the symmetry is already broken with the random initialization of weight parameters. We initialize all biases to a deterministic small, positive value, which is due to the use of ReLU's in the network used. If the bias is negative, the probability of activating the neuron initially with random weights is smaller than 1/2; and if the neuron is not activated, the gradient cannot be propagated back through that neuron [37].

3.3 Analysis of the Results

3.3.1 Optimization of Both Methods with Validation Data

Before moving on to the grand comparison of handcrafted versus learned representation, we did small empirical studies on how to best handle some details. One such detail is incorporating spatial information into FV's. The OpponentSIFT descriptor by itself encodes no spatial information: a local feature should be described the same way regardless of its location within the image. However, strict reliance on the local feature will deter us from making use of the discriminative information held by the general location of an object in an image, as well as the relative positioning of related local features.

We tested two methods of incorporating spatial information against the standard FV

formulation with no spatial information. One approach tested was to augment the dimension-reduced descriptor the XY coordinates of each point feature. The coordinates were adjusted such that the center of the image was $(0, 0)$ and both the width and length of the image would be equal to 1. The other method tested was a 3x1 *spatial pyramid*, as explained in Subsection 2.3.3. We tested all 3 methods on the validation data of VOC2012, and decided to use the 3x1 spatial pyramid for future experiments due to its higher performance on validation data 3.1.

Table 3.1: Performance comparison of different methods of incorporating spatial information for FV's.

Strategy Name	Description	mAP
FV	Spatial information is discarded.	0.5912
FV_AugmentXY	XY coordinates of point features are augmented to the descriptor.	0.6025
FV_SpatialPyramid	FV's are extracted in a 3x1 spatial pyramid.	0.6223

A similar study was done on the convolutional network, which was selected as CaffeNet due to its immense popularity as well as good performance for its reasonable depth. There was a concern that the size of the VOC2012 dataset might be too small to train an 8-layer network, to a point that it might be actually beneficial to reduce the number of layers in the network. We implemented an incremental reduction in the network depth by removing layers one by one, at each step retraining the network from scratch. Again, we tested the networks on the VOC2012 validation data. We observed that the reduction in the number of convolutional layers to 3 actually caused a slight increase in performance (Table 3.2, Figure 3.5). We used both the full-sized CaffeNet and the 3+3-layered version for the big test.

Table 3.2: Performance comparison of different number of Convolutional and Fully Connected layers for the ConvNet.

Conv Layers	FC Layers	mAP
5	3	0.3654
3	3	0.3687
3	2	0.3628
2	2	0.3435
2	1	0.2773

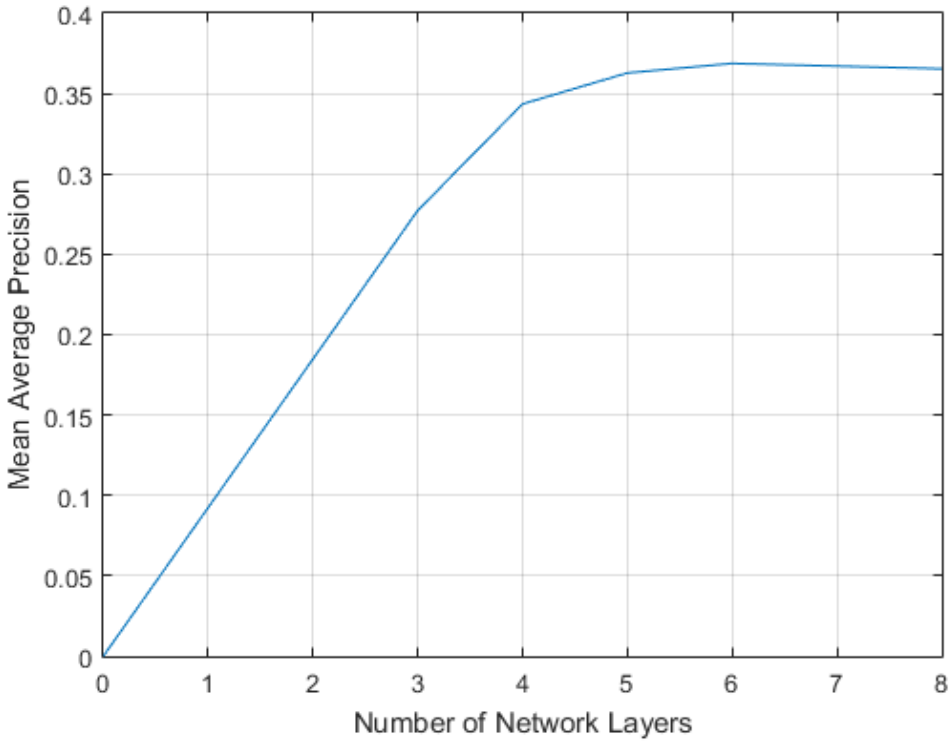

Figure 3.5: The effect of the depth of a network on its performance.

3.3.2 Comparison of FV Classifier versus CaffeNet

The VOC2012 dataset, with only 10,000 images, is tiny by today's standards - several times more photos are uploaded to Facebook every **minute**. The increase in dataset sizes may be the main cause of the rise of Deep Learning, but it might also be the case that better practices, such as the use of ReLU's coincided with the arrival of Big Data. By limiting training for both sides to only use the VOC2012 data, we aim to isolate the effect of dataset size on performance. The Average Precision results for each class, as well as the mean of all 20 classes are given in Table 3.3.

Looking at Table 3.3, we see a perhaps unsurprising result: even with a couple of years of improvements, the neural network with insufficient data performs much worse than the handcrafted method. Using some sort of data augmentation, even if applied to both methods, might help the ConvNet gain back some ground against the FV classifier, but the gap between the methods is simply too large to be covered up in that manner.

Table 3.3: Performance comparison of the FV based classifier and two ConvNet variants, trained only with VOC2012 data.

Class	FV_SpatialPyramid	CaffeNet_Full	CaffeNet_3+3
Aeroplane	0.87000	0.65700	0.65860
Bicycle	0.56370	0.29940	0.30720
Bird	0.60510	0.26010	0.25310
Boat	0.65720	0.34130	0.32510
Bottle	0.28160	0.13630	0.14010
Bus	0.75870	0.44330	0.46520
Car	0.57950	0.38540	0.35810
Cat	0.64470	0.37640	0.32970
Chair	0.50570	0.32770	0.29510
Cow	0.41500	0.17650	0.13430
Dining Table	0.47230	0.22830	0.20760
Dog	0.52040	0.33130	0.31480
Horse	0.60380	0.28830	0.31500
Motorbike	0.65070	0.39440	0.38820
Person	0.81100	0.66880	0.63880
Potted Plant	0.32210	0.11320	0.09900
Sheep	0.50980	0.26950	0.26640
Sofa	0.46650	0.20830	0.17700
Train	0.81060	0.41960	0.45370
Tv / Monitor	0.60150	0.31520	0.28660
Mean	0.58250	0.33200	0.32070

Comparing the per-class AP values of both deep networks studied, we see no statistically significant difference, indicating that the capacity introduced by the additional convolutional layers is left untapped. Going back to Table 3.2, we might remember that with the validation data performance really only started to fall off when the depth was reduced all the way to 3 from the original 8. The obvious takeaway is that deep networks cannot be utilized to its full capacity without having enough training examples. At the same time, as we increase the number of training examples, the performance should eventually get bounded by the depth of the network.

Moving onto the larger scale, we first have to note that the additional ImageNet data, due to having an entirely different set of classes, cannot be used to train a classifier that tries to learn to classify the 20 VOC classes from a given representation. The unsupervised training portion of Fisher Vectors, i.e. learning the PCA projection and

the GMM, does not improve by throwing in more data; it would actually hurt to add the ImageNet data into the mix during training as it would only work in lowering the amount of data, that is more directly related to the test data, made available to the learner. In order to provide a fair platform, we decouple the training of the ConvNet into two stages as well: the first stage learns a discriminative representation from the ImageNet data - while it discriminates between a different set of classes, there are certainly similarities in between, so it incidentally learns how to differentiate between VOC classes as well. For the second stage, we forward propagate the VOC2012 data through the network - with its final, class specific layer trimmed out - and obtain deep representations for each image to be classified. Then we train a classifier for these deep representations the same way we do with FV representations: we use an SVM. Another perk of this approach is that it allows us to directly compare the representations, since the classifier part is the same. The resulting classifiers are compared in Table 3.4.

From Table 3.4, we see that pretraining with a large dataset improved the ConvNet representation immensely. We also confirm our speculation that after some point, the performance is not bounded by the amount of data, but rather the depth of representation. The difference in performance between the 8-layer and the 7-layer representation is definitely significant. Also, 8 layers seems to mark a critical threshold where the learned representation starts to get ahead of its handcrafted counterpart by a healthy margin.

Speculatively, it could be said that discriminative power of a representation is not class- or problem-specific, owing to the fact that we just trained a representation that had no access to the class labels of the actual problem it was tested against; instead it was subjected to a different set of problems in training. Surely one might argue that the sets of classes have enough in common that such a generalization is not really applicable; however, [65] reports that off-the-shelf use of ConvNet representations beat the state-of-the-art in a considerably large variety of recognition tasks.

The perks of using ReLU type neurons cannot be forgotten: with proper weight initialization, the network will reliably settle in some point that is very close to optimal almost every time, making a statistical fluke very unlikely. In that sense both method-

Table 3.4: Performance comparison of the FV based classifier and two ConvNet variants, trained with ImageNet data in addition to VOC2012.

Class	FV_SpatialPyramid	CaffeNet_Full	CaffeNet_5+2
Aeroplane	0.87000	0.90190	0.86060
Bicycle	0.56370	0.65470	0.58030
Bird	0.60510	0.78020	0.71970
Boat	0.65720	0.70880	0.62460
Bottle	0.28160	0.37970	0.29290
Bus	0.75870	0.77490	0.71060
Car	0.57950	0.60470	0.52180
Cat	0.64470	0.81320	0.74710
Chair	0.50570	0.44390	0.45040
Cow	0.41500	0.51580	0.45010
Dining Table	0.47230	0.48700	0.45610
Dog	0.52040	0.76910	0.70470
Horse	0.60380	0.73070	0.62420
Motorbike	0.65070	0.70930	0.67710
Person	0.81100	0.87180	0.87570
Potted Plant	0.32210	0.30890	0.28680
Sheep	0.50980	0.64760	0.55200
Sofa	0.46650	0.33550	0.37900
Train	0.81060	0.83860	0.76440
Tv / Monitor	0.60150	0.59660	0.55730
Mean	0.58250	0.64360	0.59180

ologies are similar: even though there is a lot of inherent randomness in training of both models - for the FV, constructing the generative GMM; for the ConvNet, randomization of the order of images and random weight initialization - the results are quite reliably within a small range of values.

As a final remark, we see that the handcrafted methods still are not too far behind, even beating the full CaffeNet representation on a few VOC class challenges. Can we do better by combining the two, or would it be too costly? Would an ensemble of several deep representations be more useful instead? While we do not comprehensively tackle this area of ensemble learning, the experiments on Subsection 4.4.2 deal with this topic.

CHAPTER 4

OBJECT DETECTION: SELECTING THE CORRECT REGION

In the prior chapters, we have established that deep ConvNets are able to perform image classification at a significantly high accuracy. On the other hand, the comparatively harder problem of object detection has not seen the same level of success. In this chapter, we give a line of reasoning for why the state of the art ConvNets, originally used for classification, form the backbone of leading object detection systems; then we describe the methods used in those systems to generate candidate region proposals, which are required to complete the detection framework in a manner that reduces it to making a choice between proposed regions according to their classification score. We also propose an enhancement to one of the existing approaches that aims to improve detection accuracy at no additional computational cost. Finally, we share the results of some experiments that 1) compare the original and enhanced methods, and 2) explore a couple of ensemble strategies that combine handcrafted and learned representation techniques for object detection.

4.1 Motivation

Do we have to use a classifier in an object detection system? Yes, since the problem of object detection, according to the definition used throughout this thesis, requires for a successful detection to correctly answer these two questions:

- Where is the object?
- What is the class of the object?

Our solution will depend on the order in which we approach these two questions. If we decide to first tackle the 'Where?' problem, we have to first look for regions of interest in which an object resides, in accordance with some, possibly learned, *objectness* criterion. Then we can answer the 'What?' question by obtaining the representation for a region found to contain an object and classifying it. Alternatively, we may decide to first try to classify the whole image, answering the 'What?' question; then attempt to localize that object within the image. In practice, neither of the approaches are great: classifying first might be useful only if the image contains a single object, and trying to obtain the location without considering the type of the object is clearly suboptimal. One example situation where such a strategy fails is when parts of an object is identified as the objects in the image, instead of the target object itself; such as the head and the shirt of a person instead of the person as a whole [71]. The practical approaches try to handle these tasks together, taking the classification scores into account while deciding on the location of the object.

Let us imagine that we have the perfect classifier, that is, one that predicts the class of every image correctly, and whose confidence in its prediction drops when the object covers a smaller part of the image, or when part of the object is occluded or cropped out of the image . In other words, as we better localize the object, the classifier score increases, and the object resides at the local maximum of the space of possible object localizations. If we have a perfect classifier, we can also achieve perfect detection, provided that we can afford to run the classifier through each possible object localization exhaustively. While the state of the art in classification is by no means perfect, it is good enough that such an approach becomes viable.

It is quite common in object detection to output the result as a rectangular box that encapsulates the detected object, called the bounding box. This also allows detection algorithms to ignore arbitrarily shaped regions, and work only with rectangular regions. If an exact segmentation of the detected object is necessary, it can be extracted by a separate segmentation algorithm afterwards. This approach reduces the search space for possible object regions to a much more reasonable level, making brute force

methods such as Sliding Window (Section 4.2.1) viable. However, problems may arise when the object of interest is not rectangular shaped, which is demonstrated in Figure 4.1. In such cases, the true bounding box will encompass a significant amount of background clutter; which may cause a smaller bounding box that encapsulates only parts of the object to have a better classification score, if it includes the most discriminative features of the object.

Figure 4.1: The problem with restricting regions to be rectangular. The blue bounding box, while true, includes a greater percentage of background clutter compared to the red one. Arbitrarily shaped regions alleviate this problem.

4.2 Related Work

4.2.1 The Sliding Window

The exhaustive search method is a brute force way of performing object detection with classification. In exhaustive search, the detector blindly moves through the search space in all directions, trying to classify every possible region. The method offloads all decision making to the classifier, ensuring that the region with the best classification score is detected with certainty.

The main problem with the exhaustive search is that the amount of classifications required to complete the search on a single image is too high, even when only rectangular regions are taken into account. The search space is actually 4 dimensional, since a rectangular region can be defined by the coordinates of two opposite side corners. To make an estimate, we may also define it by the coordinates of its center, its size and aspect ratio. For a 640x480 image, there are:

- 300,000 candidate center points,
- 250 possible scales for each center (on average),
- 200 possible aspect ratios for each scale (on average).

That comes up to $300,000 * 250 * 200 = 15$ *billion* possible bounding boxes per a single image! This is obviously infeasible, so typically a few constraints are introduced to reduce this number [72]. One is to increase the step size, that is, skip every few pixels while going through the search space. Another is to fix the size and/or aspect ratio of the search window, which provides immense computational relief by reducing the dimensionality of the problem; but requires a prior to be set for the system about the shape and size of the objects to be found, which is fine if most objects to be detected are of similar size. The **Sliding Window**, used in [76], [17], [34], [25], [24] among many others, is not a purely exhaustive search method, but one with such constraints.

The Sliding Window is demonstrated in Figure 4.2. When the window size and aspect ratio are fixed, searching through the image can be implemented as sliding a bounding box from left to right, going one step down when the border is reached. In most object detection tasks, sizes and shapes of objects have a large variety, so a pure Sliding Window approach is not very useful. Using a number of windows with varying scales has been proposed by [76], but smarter approaches that reduce the search space to a tractable number of candidate windows(object proposals) without adding too much computational overhead are also used. These methods can be divided into two groups: *objectness*, and *similarity* based methods per [79]; or *window-scoring* and *grouping* based methods per [40]. In this work we adopt the former (objectness vs. similarity)

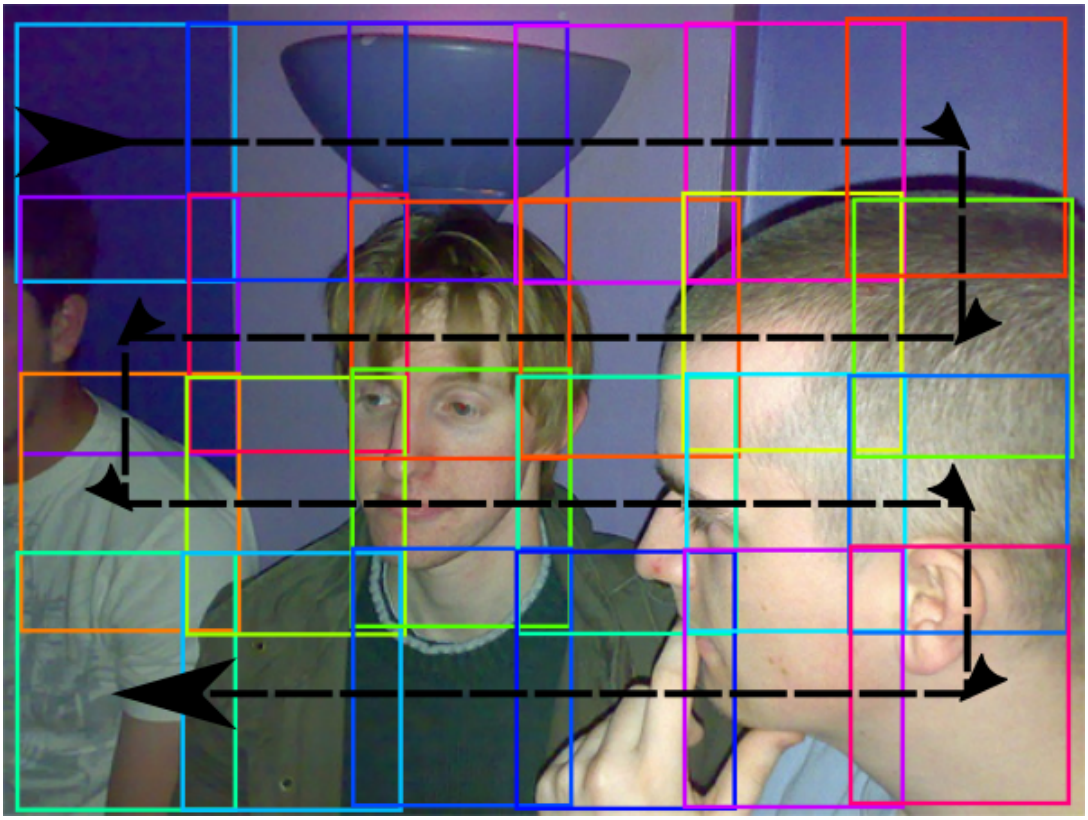

Figure 4.2: The sliding window detection, applied to an image.

because it fits well with our focal point on the topic, which is the question of what constitutes a good basis for evaluating the candidacy of a region.

4.2.2 Objectness Based Region Proposal Methods

Objectness based approaches reduce the number of proposals via some sort of *objectness* measurement - a measure of how much an image region resembles an object. They can be considered as a preliminary classification stage that divides the search space into windows that might contain objects, and those that definitely do not. This classifier is tuned to have very high recall; any missed object at this stage will never be seen by the final, more accurate classifier. The objectness classifier trades off precision in favor of very low computation cost per region, but not recall, which makes it feasible to apply to more object windows.

Objectness classifiers make use of a large variety of easily computable features, from color contrast to saliency to symmetry [64]. As the computational complexity of fea-

ture extraction grows, the portion of the search space that can be explored by the algorithm gets lower to accommodate. Even with a very simple classification algorithm, it is still impractical to process all possible windows. Typically, a uniformly distributed sample of all windows are subjected to the objectness classifier, meaning that it depends on random chance whether any sample window aligns well with the actual object boundary. Therefore, it may not be possible to have great localization with this approach.

4.2.2.1 Binarized Normed Gradients (BING)

The Binarized Normed Gradients algorithm is an extraordinarily fast, objectness-based proposal generation technique [13]. Compared to other state-of-the-art algorithms for the same task which require a few seconds to generate proposals for a single image [2][72], BING can process the same image on the order of milliseconds.

The BING algorithm (Figure 4.3) is, in essence, a sliding window detector. At the first stage of the algorithm, the input image is reshaped to different scales and aspect ratios. A fixed size (8x8 pixels) sliding window is scrolled through each of the rescaled versions of the image. This is equivalent to using several windows of a variety of sizes and aspect ratios; but is more efficient in implementation. Each window is represented by its normed gradients(NG), which is easy to compute while remaining discriminative towards objects and non-objects.

The Sobel operator [22] is a well-known method used to compute the gradient image. The horizontal directional gradient of an image I can be computed as:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{I}, \tag{4.1}$$

Similarly, for the vertical gradient:

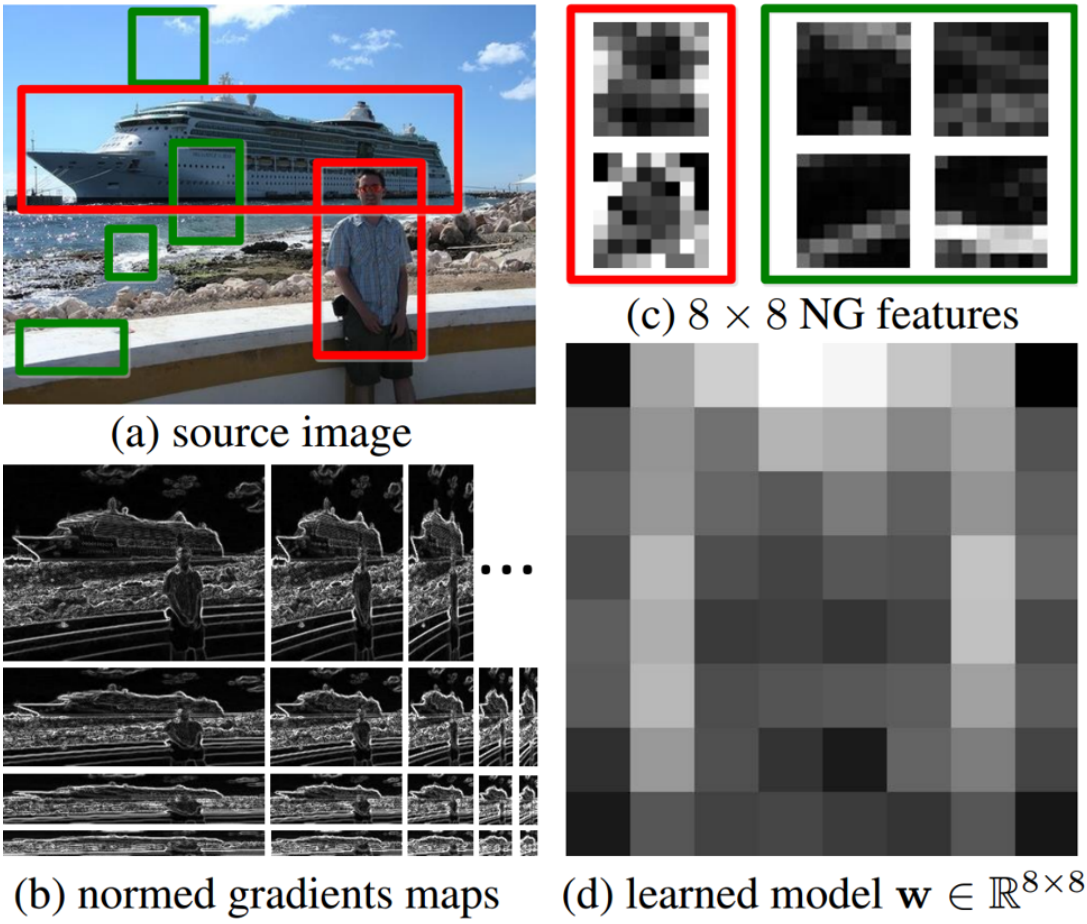

Figure 4.3: The BING Algorithm. (a) Red bounding boxes indicate objects, green ones are non-object boxes; (b) The image is reshaped to different scales and aspect ratios; normed gradient(NG) maps are extracted at each scale; (c) NG features for the bounding boxes in (a); (d) The linear model weights used to classify the NG features. Taken from [13].

$$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{I}, \tag{4.2}$$

The magnitude of the gradient, or the normed gradient at some point P on the image, is the L2 norm of $[G_{x,P} G_{y,P}]^T$:

$$\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2}, \tag{4.3}$$

where each operation is done elementwise.

The NG representation is discriminative for objectness, since there tends to be a strong boundary between an object and the background. The obtained features are translation, scale and aspect ratio invariant. Each bounding box is represented by its 8x8 NG map, which is flattened to a 64 dimensional vector for classification.

The classifier is a linear SVM [15], learned with ground truth object windows and randomly sampled background windows. The learned model is shown in Figure 4.3. The objectness score of a window is the inner product of its NG representation with this model. To further refine the objectness scores, a linear adjustment, with learned weights, is made to the score depending on its scale and aspect ratio. The motivation is that bounding boxes with some sizes and aspect ratios are more likely to contain objects compared to others. So, the final objectness score o_l of a window l, with a NG representation $\vec{g_l}$ and a scale/aspect ratio identifier i, is given as:

$$o_l = v_i < \vec{w}, \vec{g_l} > + b_i, \tag{4.4}$$

where (v_i, b_i) are the learned adjustment terms for the corresponding scale and aspect ratio. This main algorithm is sped up immensely by approximating w as a weighted sum of binary vectors, and binarizing the NG representations.

4.2.3 Similarity Based Region Proposal Methods

Similarity based methods try to construct object regions, instead of trying to eliminate a large amount of windows from a huge set of candidate regions. In place of classification, they perform segmentations of the image into areas that possibly correspond to objects. Multiple segmentations of the same image are used to generate a large variety of proposals, in order to ensure that all objects are included in the set of proposals output to the final classifier stage. These methods can generally achieve better localization, since regions are generated from actual segmentations of an image as opposed to random sampling.

There are different segmentation strategies used; one common approach is hierarchi-

cal merging where starting from an oversegmentation of the image, neighbor regions are merged together according to some criteria to form larger regions. At each step, the newly created regions are added to the pool of object window proposals. The criteria for merging regions tend to be not objectness-related, but rather measures of similarity of regions such as color and texture. One problem with similarity based methods is that the algorithms do not differentiate between object regions and other regions.

4.2.3.1 Selective Search

Selective Search is a proposal generating algorithm that generates proposals from hierarchical, similarity based segmentations [72]. It is arguably the most well-known algorithm of its kind thanks to its use in many state-of-the-art detection frameworks such as R-CNN (Regions with Convolutional Neural Network Features) [32]. The method uses a hierarchical, bottom-up grouping process to generate regions of varying sizes, starting from an oversegmentation, i.e. superpixels as described in [26]. Diversification in proposals is achieved in three ways: one way is to run the algorithm in several, complementary color spaces; another is to vary the similarity measurement formula, and the final one is to vary the oversegmentation hyperparameter which changes the initial region map.

The algorithm starts with obtaining the superpixel segmentation which generates the initial regions in the hierarchy. A graph is created from the region map, on which the nodes are the regions, and nodes corresponding to neighboring regions are connected. The edge weights relate to the similarity between the regions. Iteratively, pairs of regions are merged one at a time. At every step, the pair of nodes with the largest edge weight is merged together, and the similarity weights between the newly generated region and its neighbors are calculated from scratch. Each newly constructed region in hierarchy is added to the list of candidate windows. The process continues until the whole image is merged into a single region. The algorithm is repeated with the other combinations of color spaces with different values of the oversegmentation hyperparameter and different similarity measures.

The similarity measure is made up of a combination of color similarity, texture simi-

larity, size, and the fill measure. The color similarity is computed from the color distributions of regions. For a pair of regions (r^i, r^j) with normalized color histograms $(\vec{c^i}, \vec{c^j})$, the color similarity is computed as:

$$s_{color}(r^i, r^j) = \sum_k \min(c_k^i, c_k^j), \tag{4.5}$$

where each c_k^i corresponds to a bin count of the histogram. The normalized color histogram of the merger region is the average of the merged regions, weighted according to their relative size. After computing histograms of the initial regions, the rest of color similarity computations becomes a much simpler task.

The texture similarity is computed in a closely related manner. Texture histograms $\vec{t^i}$ are obtained for each region using a fast, SIFT-like method. The histograms are compared in the same way:

$$s_{texture}(r^i, r^j) = \sum_k \min(\vec{t_k^i}, \vec{t_k^j}). \tag{4.6}$$

The size metric is a way to enforce merging of smaller regions first; therefore it decreases as the size of the region pair increases:

$$s_{size}(r^i, r^j) = 1 - \frac{size(r^i) + size(r^j)}{size(im)}. \tag{4.7}$$

Finally, the fill measure is used to encourage the algorithm to fill the holes. The motivation here is that a region enclosed by another has a good chance to belong to the same object, even if they share no color or texture similarity. Wheels of a car are enclosed by its chassis, and they are parts of the same object even though they are nothing alike. The fill measure is calculated as:

$$s_{fill}(r^i, r^j) = 1 - \frac{size(B^{i,j}) - size(r^i) - size(r^j)}{size(im)}, \tag{4.8}$$

where $size(B^{i,j})$ is the size of the bounding box that contains both regions. The final similarity measure can be the sum of any combination of these 4 measures. If size

and fill are used together, the measure reduces to the bounding box size of the merger relative to the size of the image.

In the end, the diversification strategies help to create a very long list of object proposals. If only a select quantity of proposals is desired, these proposals need to be ordered in some way according to their likelihood of containing an object. Since no objectness measure is used, the list is reverse sorted according to the place of each region in the hierarchy, except randomized to avoid having only the larger regions in the final list. Actual object regions are expected to be found in many of the diversification strategies; so it becomes quite likely that at least one of those regions makes it through the randomization stage to the final list of proposals.

4.3 Fisher-Selective Search

The Fisher-Selective Search (FSS) algorithm is our extension to the standard Selective Search algorithm. A drawback of all algorithms described in Section 4.2 is that there is no interaction between the proposal generation stage and the classifier stage. If some part of the region classification process can be reused in the region proposal generation part of the framework, not doing so would be a waste of resources. And since the goals of both stages are not far apart from each other, we should be able to find some smart approach to make the algorithms more similar, so that we can compute the common part of both stages in one go.

An argument against such an approach is the generalization angle: if we take a proposal generating algorithm and infuse it with parts of our specific classification algorithm, it does not suddenly become unusable with any other classification strategy, but in that case it certainly loses its computational advantages. Even so, the fusion algorithm might still be preferred over the original one if it provides enough additional accuracy to justify the trade-off in computation time. Such an approach is similar to using ensembles of classifiers in the sense that it combines more than one classification method to improve overall accuracy. Ensemble methods are quite prevalent in machine learning and classification literature [29] [30]; while they improve accuracy, the improvement is not linear with the number of classifiers to be combined.

In Fisher-Selective Search, we take advantage of the fact that Fisher encoding is additive in the new feature space. This means that in a merging strategy, the Fisher Vector representation of the merger region is the weighted average of FV's of the merged regions. Therefore, after computing the FV's of the initial oversegmentation regions, propagating the FV's throughout the merging process comes at virtually no cost. It is important to note that the Improved Fisher Vector(IFV) formulation does not have this additive property, but can be derived from the original FV easily (with Eqn. 2.17) when needed. Equation 4.9 describes the mathematical relation between the FV's of two regions and the FV representation of the merged region:

$$\hat{\mathbf{\Phi}}_t = \frac{c_i\hat{\mathbf{\Phi}}_i + c_j\hat{\mathbf{\Phi}}_j}{c_i + c_j}, \tag{4.9}$$

where c_i, c_j are the number of local features contained in regions r^i, r^j.

4.3.1 Fisher-Selective Metrics

The Fisher-Selective Search proposes two new decision metrics to be used as part of the merging strategy: one describing the similarity between two regions in terms of their FV representations and another that tries to construct high scoring regions.

The FV similarity metric has its basis in the fact that distance between two FV's is a good indicator of similarity between the regions they represent. Linear distances in the FV feature space is meaningful in the sense that they are equivalent to computing dissimilarity with the Fisher Kernel. Thus, we propose the FV similarity metric as the inverse of Euclidean distance between two FV's:

$$s_{FV}(r^i, r^j) = \frac{1}{1 + \sqrt{\sum_k (\Phi_k(\mathbf{X}_i) - \Phi_k(\mathbf{X}_j))^2}}. \tag{4.10}$$

Here, $\mathbf{X}_i$ corresponds to the local features contained in r^i, and $\mathbf{\Phi}(\mathbf{X}_i)$ is the **Improved** FV representation of region r^i. This operation maps the L2 norm of the distance vector between two FV's to a value between 1 and 0, which is important since all other Selective Search metrics have the same property which allows us to combine

them into more robust similarity metrics appropriately. The function is monotonically decreasing: similarity always decreases as distance increases.

The second metric is a problem-specific metric that prioritizes merging of high-scoring regions with some classifier. Since many object recognition problems are multi-class, we propose a strategy of defining the score of a potential merge as the maximum of its scores on all tasks:

$$s_{FishObjectness}(r^i, r^j) = \frac{1}{1 + \exp[-\max(\mathbf{y}(\mathbf{\Phi}_t))]}, r^t = r^i \cup r^j, \tag{4.11}$$

where $\mathbf{y}(\mathbf{\Phi}_t)$ is the multiclass classification score vector for the IFV of the merged region. Again, the scores are mapped to between 0 and 1, this time with the monotonically increasing sigmoid function. It should be noted that this metric is, for all intents and purposes, an objectness metric for a given problem. In some sense, the inclusion of this metric bridges the gap between the similarity-based and objectness-based proposal generation methods.

4.3.2 Detection With Fisher-Selective Search

The inclusion of FSS-specific metrics defined earlier allows us to skip having a separate classification stage; the outputted bounding boxes come with already computed classification scores. This results in a very compact detection framework. The process as a whole is described in Algorithm 1.

4.4 Experiments

4.4.1 Comparison of Fisher-Selective Search with Vanilla Selective Search

In this section, we put our Fisher-Selective Search algorithm to the test: we check the performance of the method on the VOC2012 detection task. We directly compare the results with the results of the same experiment conducted with the vanilla Selective Search algorithm.

Algorithm 1: The Fisher-Selective Search object detection algorithm.

Input: Image, Local features with locations, Generative model parameters, Classifiers

Output: Set of bounding box-corresponding classification score pairs (B, Y)

Obtain initial regions $R = r^1, r^2, ..., r^n$ with oversegmentation;

for $i \leftarrow 2$ **to** l **do**

- Count and store the number of features in r^i as $C = c^1, c^2, ..., c^n$;
- Compute the FV $\hat{\mathbf{\Phi}}_i$ of r^i;
- Compute the corresponding IFV $\mathbf{\Phi}_i$;

Initialize similarity set $S \leftarrow \emptyset$;

for *each neighboring pair* (r^i, r^j) **do**

- Calculate similarity $s(r^i, r^j)$;
- $S \leftarrow S \cup s(r^i, r^j)$;

while $S \neq \emptyset$ **do**

- Get $s(r^i, r^j) = \max S$;
- Merge $r^t \leftarrow r^i \cup r^j$;
- Compute the new FV $\hat{\mathbf{\Phi}}_t$ using merged region FV's and feature counts;
- Compute the corresponding IFV $\mathbf{\Phi}_i$;
- Remove old similarities: $S \leftarrow S - (s(r^i, r^*) \cup s(r^j, r^*))$;
- Calculate new similarities $S_t \leftarrow s(r^t, r^k)$ for each neighbor r_k of r_t;
- $S \leftarrow S \cup S_t, R \leftarrow R \cup r^t$;

Extract bounding boxes B from each region R;

Compute corresponding classification scores Y from FV's.

We construct two different versions of both methods: one full and one lite version for each. The full versions try out more merging strategies and work in more color spaces than their lite counterparts; therefore, the full versions are richer in the number of proposals. The differences between each version are explained in Table 4.1.

During testing, we found that fixing the maximum number of guesses per class per image to a finite value increases performance. After all, almost all images contain only a reasonable number of object from each class, a number that does not exceed 1 in many cases. We consider this value to be a critical hyperparameter that should

Table 4.1: Differences between each test scenario.

Acronym	Merge Strategies	Color Types	k Values	Nr. of Hierarchies
SS_full	Color+Texture+Size+Fill, Texture+Size+Fill, Size, Fill	HSV, Lab, RGI, H, Intensity	100, 200	$4 * 5 * 2 = 40$
FSS_full	FV Objectness, FV Similarity, Color+Texture+Size+Fill, Texture+Size+Fill	HSV, Lab, RGI, H	100, 200	$4 * 4 * 2 = 32$
SS_lite	Color+Texture+Size+Fill, Texture+Size+Fill	HSV, Lab	50, 150	$2 * 2 * 2 = 8$
FSS_lite	FV Objectness, FV Similarity	HSV, Lab	50, 150	$2 * 2 * 2 = 8$

be optimized with respect to the accuracy with the validation data. However, due to time constraints, we do not make such an optimization beforehand. Here, we instead submit the variation of average precision scores for different values of this hyperparameter.

We also note that all of the classifiers used in these algorithms were trained using SS proposals. Specifically, we use the ground truth bounding boxes as the positive examples, and produce negative examples by running the SS algorithm on the training data and selecting the proposals that have a small overlap with one or more of the ground truth bounding boxes. Since the hardest test cases are regions that have some overlap with the object but have bad localization, training specifically with these cases is expected to improve performance. [32] proposes running the first iteration of the classifier through the training images a second time, and adding misclassified regions to the training examples for the final classifier. We skip this, since we only care about the relative performance of the algorithms, and since the classifier is common in all compared methods, its quality has no effect on the relative performance. We also point out that learning with SS bounding boxes should favor the SS methods slightly; training a separate classifier with FSS proposals for FSS algorithms may improve their performance.

The results for each of the 4 variants are given in Table 4.2. Comparing the full

versions, we see a good amount of improvement even with a slightly reduced number of merging strategies, i.e. hierarchies. Where the FSS algorithm really shines is the comparison of the lite versions: discarding all of the merging strategies used in the original SS algorithm still leaves us with two very strong merging strategies, namely the Fisher similarity score and the Fisher objectness metric. The reduction in average precision is very low, and the lite version still beats the full SS algorithm by a healthy margin. By contrast, the lite version of the original SS algorithm falls of quite a bit compared to the full version.

Table 4.2: Comparison of methods in terms of the average precision of classification.

	Average Precision			
Maximum Objects Per Image	SS_full	FSS_full	SS_lite	FSS_lite
1	0.110	0.135	0.077	**0.144**
2	0.132	0.149	**0.092**	0.142
3	**0.137**	**0.154**	0.091	0.136
4	0.136	0.153	0.089	0.134
5	0.133	0.151	0.089	0.132

The results show that incorporating a discriminative representation into the merging strategy removes the need for much variation in proposal generation. Instead, we can stick with few strong strategies to construct the proposals and achieve better recall in a reduced number of proposals (Figure 4.4). The magic of Fisher-Selective Search is in its additive formulation and applicability to arbitrarily shaped regions. These properties allow us to compute the FV for the whole hierarchy in linear time, so we may even be able to combine it with other classifiers to obtain even better results. The next set of tests explore this possibility.

4.4.2 Comparison of Ensemble Strategies

A question lingers in our minds from Chapter 3: can we perform better by combining the best handcrafted method with the best learned representation? It would be a waste to throw away decades of experience gained from working with handcrafted features, but how can we use it efficiently? In the recent years, attempts were made to combine Fisher Vector based methods with deep networks [68, 61], which were successful to some extent; however, winning entries to the ILSVRC competitions are virtually all

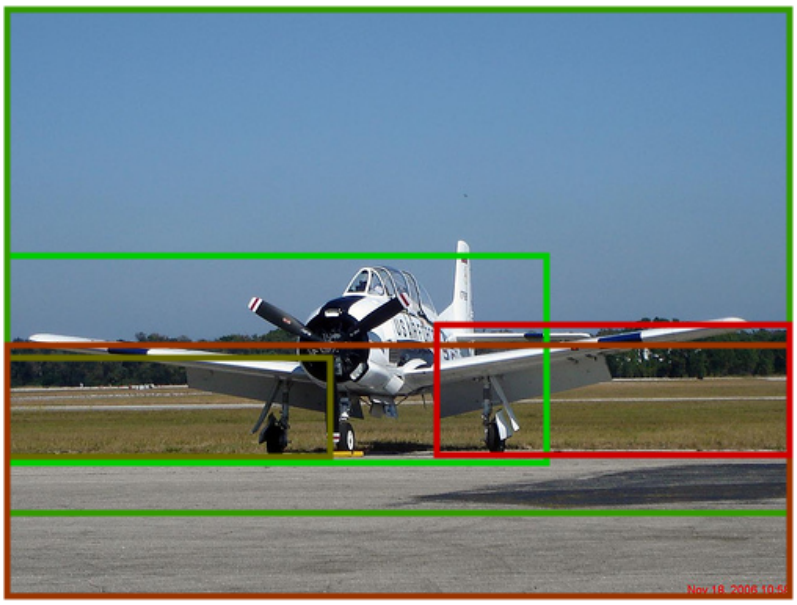

Figure 4.4: Airplane detection results with the lite FSS region proposals. The proposals are color-coded in a green-red spectrum, with a higher classification score corresponding to a greener bounding box. Only the top 5 proposals are shown.

purely based on deep network architectures [66].

In this section, we propose using FSS as a way of incorporating the power of hand-crafted representations in an otherwise purely deep framework. Instead of using the Fisher Vector based classifier from the earlier section, we have trained another classifier from the second-to-last layer of CaffeNet representations of regions. Additionally, we also trained an ensemble classifier from the concatenation of the FV and the CaffeNet representations. This time, we compare 3 different scenarios: the first one is the classification of FV representations of FSS region proposals, the leading algorithm tested in the earlier section. The competing algorithms are both ensembles of FV's and deep representations. The third strategy (Selective Search proposals, classified with ensemble classifier trained with both FV and ConvNet features) is a fairly standard way of combining different techniques into a single, better, classifier; our main goal is to test whether the novel approach of using the FV representation strictly to produce better region proposals, then using a purely deep representation in the

classifier portion of the framework, gives comparable results to the standard method. The ConvNet representations here are computed from CaffeNet, described and used for classification in Chapter 3. The methods are matched to their acronyms in Table 4.3, and the average precision results are given in Table 4.4.

Table 4.3: Description of which detection framework each acronym represents.

Acronym	Description
FSS_FV	Fisher-Selective Search proposals, classified with Fisher Vectors only
FSS_ConvNet	Fisher-Selective Search proposals, classified with ConvNet features only
SS_FVConvNet	Selective Search proposals, classified with ensemble classifier (trained with both FV and ConvNet features)

Table 4.4: Comparison of ensemble methods in terms of the average precision of classification.

	Average Precision		
Max Objects/Image	FSS_FV	FSS_ConvNet	SS_FVConvNet
1	0.135	0.258	0.274
2	0.149	0.293	0.295
3	**0.154**	**0.299**	**0.296**
4	0.153	0.297	0.294
5	0.151	0.296	0.295

From the tables, we see that the introduction of deep convolutional representations to our framework has an immensely positive effect on performance: the Average Precision is doubled in both ensemble strategies. It is also very promising to see that implicitly formulating the information contained in Fisher Vectors in the form of better region proposals is practically equivalent to an explicit formulation within the ensemble classifier.

CHAPTER 5

CONCLUSIONS

Throughout this thesis work, we explored the area of visual, automatic object recognition with a tailored focus on the specific problems of image classification and object detection. We started off with the claim that the representation is the cornerstone of any object recognition framework, that the stepping stones in the field have always been a jump in the capability of the state-of-the-art representation.

A lot of time was spent on handcrafted representations, in particular the Fisher Vector representation, to understand how and why the paradigm shift in Computer Vision happened so suddenly; and we tried to identify the point where deep representations surpassed their handcrafted counterparts. We found that the availability in very large, labeled datasets allowed us to learn a discriminative representation from scratch and saw that it performed better than handcrafted approaches; even when a problem involving classification is not accompanied by a large dataset, deep representations trained in other, large datasets still performed very well due to their immense discriminative strength.

We devoted Chapter 4 to exploring the relation between the classification and detection problems. We mentioned some popular proposal generation methods and talked about their advantages as well as limitations. Furthermore, we proposed an extension to the widely used Selective Search algorithm, called the Fisher-Selective Search, that showed great promise in experimental results, beating the vanilla Selective Search algorithm by a healthy margin. We argued that the extension adds no additional cost to the standard *proposal generation followed by classification* framework, as long as a Fisher Vector classifier is used.

Finally, we proposed a novel, implicitly ensemble detection framework that makes use of discriminative information supplied by the Fisher Vector at the proposal generation stage, followed by a deep representation classifier. We envision that going into the future, such approaches may help the experiences gained through years of research with handcrafted features live on.

As future work, the hypothesis that performance of a Convolutional Neural Network is either bounded by the size of the training set or the depth of the network must be tested thoroughly. Also, an empirical study of the computational cost of Fisher-Selective Search compared to the vanilla Selective Search may be considered. Finally, for a classifier-independent evaluation of Fisher-Selective Search, the recall versus number of proposal windows curve of the algorithm could be studied.

REFERENCES

[1] *{IEEE} Conference on Computer Vision and Pattern Recognition*. IEEE, 2015.

[2] B. Alexe, T. Deselaers, and V. Ferrari. Measuring the objectness of image windows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2189–2202, 2012.

[3] R. Arandjelovic and A. Zisserman. Three things everyone should know to improve object retrieval. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2911–2918. IEEE, June 2012.

[4] F. A. C. Azevedo, L. R. B. Carvalho, L. T. Grinberg, J. M. Farfel, R. E. L. Ferretti, R. E. P. Leite, W. Jacob Filho, R. Lent, and S. Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *The Journal of comparative neurology*, 513(5):532–41, Apr. 2009.

[5] L. J. Ba and R. Caruana. Do Deep Nets Really Need to be Deep? *CoRR*, abs/1312.6, Dec. 2013.

[6] Y. Bengio, I. J. Goodfellow, and A. Courville. Deep Learning. 2015.

[7] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a {CPU} and {GPU} Math Expression Compiler. In *Proceedings of the Python for Scientific Computing Conference ({SciPy})*, 2010.

[8] A. Bosch, A. Zisserman, and X. Muñoz. Scene classification via pLSA. In *Computer Vision–ECCV 2006*, pages 517–530. Springer, 2006.

[9] Y.-L. Boureau, N. Le Roux, F. Bach, J. Ponce, and Y. LeCun. Ask the locals: Multi-way local pooling for image recognition. In *2011 International Conference on Computer Vision*, pages 2651–2658. IEEE, Nov. 2011.

[10] K. Chatfield, V. Lempitsky, A. Vedaldi, and A. Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *British Machine Vision Conference*, 2011.

[11] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. In *British Machine Vision Conference*, 2014.

[12] K. Chellapilla, S. Puri, and P. Simard. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

[13] M.-M. Cheng, Z. Zhang, W.-Y. Lin, and P. H. S. Torr. {BING}: Binarized Normed Gradients for Objectness Estimation at 300fps. In *IEEE CVPR*, 2014.

[14] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The Loss Surfaces of Multilayer Networks. Nov. 2014.

[15] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sept. 1995.

[16] A. Cotter, O. Shamir, N. Srebro, and K. Sridharan. Better Mini-Batch Algorithms via Accelerated Gradient Methods. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1647–1655. Curran Associates, Inc., 2011.

[17] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[18] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[19] Y. Dauphin, R. Pascanu, c. Gülçehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *CoRR*, abs/1406.2, 2014.

[20] Y. N. Dauphin and Y. Bengio. Big Neural Networks Waste Capacity. *CoRR*, abs/1301.3, Jan. 2013.

[21] M. Dixit, S. Chen, D. Gao, N. Rasiwasia, N. Vasconcelos, W. Li, and N. Vasconcelos. Scene Classification with Semantic Fisher Vectors. *Vision Research*, 50(22):2295–2307, 2010.

[22] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*, volume 7. 1973.

[23] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*, 111(1):98–136, June 2014.

[24] S. S. Farfade, M. Saberian, and L.-J. Li. Multi-view Face Detection Using Deep Convolutional Neural Networks. *CoRR*, abs/1502.0, 2015.

[25] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–45, Sept. 2010.

[26] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[27] E. Fiesler. Neural Network Topologies. 1996.

[28] D. J. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors. *Computer Vision - {ECCV} 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part {I}*, volume 8689 of *Lecture Notes in Computer Science*. Springer, 2014.

[29] Y. Freund and R. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. *Computational learning theory*, 55(1):119–139, 1995.

[30] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting, 2000.

[31] J. C. Gemert, J.-M. Geusebroek, C. J. Veenman, and A. W. Smeulders. Kernel Codebooks for Scene Categorization. In *Proceedings of the 10th European Conference on Computer Vision: Part III*, ECCV '08, pages 696–709, Berlin, Heidelberg, 2008. Springer-Verlag.

[32] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[33] A. H. Gittis, S. H. Moghadam, and S. du Lac. Mechanisms of sustained high firing rates in two classes of vestibular nucleus neurons: differential contributions of resurgent Na, Kv3, and BK currents. *Journal of neurophysiology*, 104(3):1625–34, Sept. 2010.

[34] H. Harzallah, F. Jurie, and C. Schmid. Combining efficient object localization and image classification. In *2009 IEEE 12th International Conference on Computer Vision*, pages 237–244. IEEE, Sept. 2009.

[35] H. Harzallah, F. Jurie, and C. Schmid. Combining efficient object localization and image classification. In *2009 IEEE 12th International Conference on Computer Vision*, pages 237–244. IEEE, Sept. 2009.

[36] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR*, abs/1502.0, Feb. 2015.

[37] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv: 1207.0580*, pages 1–18, 2012.

[38] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies. 2001.

[39] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, Jan. 1989.

[40] J. H. Hosang, R. Benenson, P. Dollár, and B. Schiele. What makes for effective detection proposals? *CoRR*, abs/1502.0, 2015.

[41] G. Hughes. On the mean accuracy of statistical pattern recognizers. *Information Theory, IEEE Transactions on*, 14(1):55–63, 1968.

[42] T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. *Advances in neural information processing systems*, pages 487–493, 1999.

[43] Y. Jia. Caffe: An open source convolutional architecture for fast feature embedding, 2013.

[44] A. Karpathy. Lessons learned from manually classifying CIFAR-10.

[45] A. Karpathy. What I learned from competing against a ConvNet on ImageNet, 2014.

[46] B. Klein, G. Lev, G. Sadeh, and L. Wolf. Associating Neural Word Embeddings with Deep Image Representations using Fisher Vectors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4437–4446, 2015.

[47] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[49] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 2169–2178, 2006.

[50] T. Le Paine, P. Khorrami, W. Han, and T. S. Huang. An Analysis of Unsupervised Pre-training in Light of Recent Advances. *ArXiv e-prints*, 2014.

[51] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-Supervised Nets. *ArXiv e-prints*, 2014.

[52] T. S. Lim, W. Y. Loh, and Y. S. Shih. Comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*, 40(3):203–228, 2000.

[53] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.

[54] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., Mar. 1997.

[55] V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*, (3):807–814, 2010.

[56] M. H. Nguyen, L. Torresani, L. de la Torre, and C. Rother. Weakly supervised discriminative localization and classification: a joint learning process. In *{IEEE} 12th International Conference on Computer Vision, {ICCV} 2009, Kyoto, Japan, September 27 - October 4, 2009*, pages 1925–1932. IEEE, 2009.

[57] B. A. Olshausen and D. J. Field. How close are we to understanding V1? *Neural computation*, 17(8):1665–1699, 2005.

[58] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 1717–1724, Washington, DC, USA, 2014. IEEE Computer Society.

[59] A. R. Parker. On the origin of optics. *Optics & Laser Technology*, 43(2):323–329, Mar. 2011.

[60] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[61] F. Perronnin and D. Larlus. Fisher Vectors Meet Neural Networks: A Hybrid Classification Architecture. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3743–3752, 2015.

[62] F. Perronnin, J. Sánchez, and T. Mensink. Improving the Fisher Kernel for Large-scale Image Classification. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, pages 143–156, Berlin, Heidelberg, 2010. Springer-Verlag.

[63] L. Prechelt. Early Stopping-But When? In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 55–69, London, UK, UK, 1998. Springer-Verlag.

[64] E. Rahtu, J. Kannala, and M. Blaschko. Learning a category independent object detection cascade. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1052–1059, 2011.

[65] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN Features off-the-shelf: an Astounding Baseline for Recognition. Mar. 2014.

[66] 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. page 43, Sept. 2014.

[67] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, page 88, Oct. 2014.

[68] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep Fisher Networks for Large-Scale Image Classification. In *Advances in Neural Information Processing Systems*, pages 163–171, 2013.

[69] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv e-prints*, 2014.

[70] A. Torralba, K. P. Murphy, W. T. Freeman, and M. A. Rubin. Context-based vision system for place and object recognition. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 273–280 vol.1, 2003.

[71] Z. Tu, X. Chen, A. L. Yuille, and S. C. Zhu. Image parsing: Unifying segmentation, detection, and recognition. *International Journal of Computer Vision*, 63(2):113–140, 2005.

[72] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective Search for Object Recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

[73] K. E. A. van de Sande, T. Gevers, and C. G. M. Snoek. Evaluating Color Descriptors for Object and Scene Recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1582–1596, 2010.

[74] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *The Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[75] A. Vedaldi and B. Fulkerson. Vlfeat: An Open and Portable Library of Computer Vision Algorithms. In *Proceedings of the International Conference on Multimedia*, MM '10, pages 1469–1472, New York, NY, USA, 2010. ACM.

[76] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–511–I–518. IEEE Comput. Soc, 2001.

[77] L. von Melchner, S. L. Pallas, and M. Sur. Visual behaviour mediated by retinal projections directed to the auditory pathway. *Nature*, 404(6780):871–6, Apr. 2000.

[78] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-constrained Linear Coding for image classification. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3360–3367. IEEE, June 2010.

[79] X. C. H. M. X. Wang and Z. Zhao. Improving Object Proposals with Multi-Thresholding Straddling Expansion. 2015.

[80] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun. Deep Image: Scaling up Image Recognition. Jan. 2015.

[81] B. Xu, N. Wang, T. Chen, and M. Li. Empirical Evaluation of Rectified Activations in Convolutional Network. 2015.

[82] S. Zheng, C. Qiang, H. Zhongyang, H. Yang, and Y. Shuicheng. Contextualizing object detection and classification. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1585–1592, 2011.

[83] X. Zhou, K. Yu, T. Zhang, and T. S. Huang. Image Classification Using Supervector Coding of Local Image Descriptors. In *Proceedings of the 11th European Conference on Computer Vision: Part V*, ECCV'10, pages 141–154, Berlin, Heidelberg, 2010. Springer-Verlag.