FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Pre-trained Convolutional Networks and generative statistical models: a study in semi-supervised learning

John Michael Salgado Cebola

MSc DISSERTATION

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Master in Electrical and Computers Engineering

Supervisor: Luís Filipe Pinto de Almeida Teixeira

July, 2016

Resumo

Este estudo explorou a viabilidade de modelos pré-treinados de redes convolucionais e métodos generativos (nomeadamente Latent Semantic Analysis, Latent Dirichlet Allocation, Sparse Coding e Fisher Vectors) num contexto de aprendizagem semi-supervisionada aplicado a tarefas de classificação de imagem. Este objetivo foi conseguido de duas maneiras distintas: primeiro, utilizando métodos generativos na criação de um vocabulário visual aplicado a um algoritmo de Spatial Pyramid Matching, a ser utilizado por Support Vector Machines para classificação de imagens e em segundo, através do estudo do comportamento de ConvNets pré-treinadas para estimar a classificação de imagens não categorizadas. Todos os testes foram realizados em dois conjuntos de dados, um binário constituído por 25000 imagens de cães e gatos obtidos do Kaggle, que foi posteriormente expandido para ter 5 categorias, adicionando as classes de peixe, baleia e galáxia (dados obtidos também do Kaggle e do conjunto de imagens ImageNet). Os modelos pré-treinados foram obtidos do Model Zoo da toolbox Caffe, tendo sido previamente treinados para o conjunto de dados Imagenet.
Foi realizado um estudo comparativo e qualitativo destes métodos generativos para os conjuntos de dados descritos. Os modelos pré-treinados das ConvNets foram também utilizados para guiar o processo de treino semi-supervisionado, estimando provisoriamente as categorias das imagens sem etiqueta. As classificações atribuídas a cada imagem pela ConvNet foram também utilizadas como estimadores da confiança para cada etiqueta provisória.
Mostrou-se que os Fisher Vectors atingiram os melhores resultados dos métodos generativos (0.83216 para os dados binários 0.91 para os dados de 5 categorias), apesar de se terem observado precisão comparável nos outros métodos. Para os dados do conjunto binário, a ConvNet atingiu a melhor performance. No conjunto de dados de 5 categorias, as soluções que utilizaram os modelos generativos conseguiram inicialmente atingir melhores resultados, apesar de se ter mostrado que com um aumento dos dados etiquetados e maior tempo de treino, o modelo da rede pode eventualmente voltar a ter melhor performance. Além disto, mostrou-se que kernels lineares, quando utilizados com os mencionados métodos generativos, conseguem atingir melhor performance com menos tempo de treino e utilizando menos recursos computacionais. Este resultado reveste-se de importância adicional quando comparado com as necessidades intensivas de tempo e recursos computacionais das ConvNets, que requerem computação em GPUs e vários Gigabytes de memória RAM.
Os resultados do estudo esclarecem a viabilidade de métodos generativos como uma forma de redução de dimensionalidade de dados, e como os modelos estatísticos fornecem informação mais rica que, ao ser combinada com modelos de ConvNets pré-treinados, conseguem atingir resultados aceitáveis em tarefas de classificação. Estes resultados abrem caminho para continuar a exploração destas soluções para encontrar bons resultados em conjuntos de dados onde imagens etiquetadas são escassas.

Abstract

This study explored the viability of out-the-box, pre-trained ConvNet models and multiple generative methods (namely probabilistic Latent Semantic Analysis, Latent Dirichlet Allocation, Sparse Coding and Fisher Vectors) in a semi-supervised learning context for image classification tasks. This is done in two distinct manners: first, by utilizing semi-supervised generative models in the creation of a visual vocabulary in a Spatial Pyramid Matching formulation, to be used by a Support Vector Machine for image classification and secondly by studying if a pre-trained ConvNet model can be used to estimate category labels of unlabelled images. All experiments pertaining to this study were carried out over two distinct datasets, a two-class set comprised of 25000 images of cats and dogs obtained from Kaggle, which was later expanded into a 5-category dataset with additional fish, whale and galaxy classes (this additional data being again obtained from Kaggle, but also from the ImageNet dataset). The pre-trained models used were obtained from the Caffe Model Zoo, and were trained for the ImageNet dataset challenge.
A comparative and qualitative study of all generative methods was realized through multiple trials over each dataset. Furthermore, the pre-trained ConvNet was utilized to guide the semi-supervised training by estimating the class of each unlabelled image. The class score attributed by the ConvNet to each unlabelled image was also used as an estimator of the confidence of the tentative labels.
It was shown that Fisher Vectors were able to achieve the best accuracy scores out of all generative methods (0.83216 for the binary dataset and 0.91 for the 5-category dataset), although all tested methods achieved comparable results. For the binary case, none of the generative ensembles managed to match the accuracy of the pre-trained ConvNet model. For the 5-category dataset, the ensembles showed competitive results when compared to the ConvNet model, although after a more intensive training schedule using a surplus of labelled data, the ConvNet once again topped all results. Additionally, it was shown that linear kernels perform competitively when utilized in conjunction with these generative models, allowing for faster training times for the classifiers, while also utilizing less computational resources, requiring only ROM and usage of the processor. This was seen as especially relevant when compared to the ConvNets, which require some days of training even when utilizing 16 Gigabytes of RAM and multiple Nvidia graphic cards for computations.
The results of this study lend insight on the viability of these generative methods as a form of feature reduction, and how the rich statistical modelling they provide can be combined with pre-trained models to quickly achieve acceptable accuracies in classification tasks, paving the way for future exploration using these methods to allow convergence to well-performing solutions quickly with datasets scarce in labelled data.

Agradecimentos

I'd like to thank professor Luís for sharing his knowledge and insight, guiding this project through its rockiest stages all the way to its destination. I'd also like to thank my brother, whose assistance in numerous matters- from struggles with installing software, to lending me his personal computer to gather further computing power for the numerous tests, to helping me write and polish this document- proved invaluable. He was the stallion that often carried this project to its fruitful destination. Furthermore, I'd like to thank my good friend Meira, whose humour and advice over a pleasant cup of alcohol helped keep my mind intact. To everyone who was there to give me that extra push as the iron bar weighted on my shoulders at the gym- to you all, I'd like to thank you for motivating me to take care of my body and helping me put my mind off work when the stress became crushing. And, lastly, I'd like to Ria for patiently and diligently helping me with innumerable tasks as my soul was slowly sucked dry by many a institution as I worked. Truly the work of a perfect and elegant maid.

John Michael

*“Congratulations.
You managed to complete this absolutely meaningless test.”*

GLaDOS

Contents

List of Figures

List of Tables

Abbreviations

BoC	Bag of Colours
BoF	Bag of Features
BoW	Bag of Words
CCV	Colour Coherence Vector
ConvNet	Convolutional Network
CVPR	Computer Vision and Pattern Recognition conference
CUDA	Compute Unified Device Architecture
DoG	Difference of Gaussians
EM	Expectation-Maximization
FLANN	Fast Library for Approximated Nearest Neighbour search
FV	Fisher Vector
GB	Gigabyte
GMM	Gaussian Mixture Model
GHz	GigaHertz
GPU	Graphic Processor Unit
HMM	Hidden Markov Model
HoG	Histogram of Gradients
Hr	Hours
HSV	Hue, Saturation, Value
I/O	Input/Output
ILSVRC	ImageNet Large Scale Visual Recognition Competition
K-NN	K Nearest Neighbours
LDA	Latent Dirichlet Allocation
LoG	Laplacian of Gaussian
LSA	Latent Semantic Analysis
LSE	Least-Squares Error
MAP	Maximum A Posteriori
MLE	Maximum Likelihood Estimation
PCA	Principal Component Analysis
PDF	Probability Density Function
RAM	Random Access Memory
RGB	Red Green Blue
SC	Sparse Coding
SIFT	Scale-Invariant Feature Transform
SGD	Stochastic Gradient Descent
SPM	Spatial Pyramid Matching
SURF	Speeded Up Robust Features
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TSVM	Transductive Support Vector Machine
VQ	Vector Quantization

Chapter 1

Introduction

1.1 Problem contextualization and motivation

Image classification is a central problem of Computer Vision and Machine Learning, which has been the focus of much research in recent years. It's a vital task in numerous applications of either field, allowing for automation of multiple tasks in engineering, identification of patterns or analysis of visual data. With the boon of the Internet, vast amounts of data from multiple fields and for multiple applications have become readily available. Thus, it's of crucial importance to design robust algorithms and methods that allow identification and classification of this wealth of data, enabling its use, considering the value of both properly classified data and common unsorted data.

One of the greatest obstacles faced when handling these large volumes of images and video is tied to the fact that, more often than not, the visual data is unlabelled. Whilst unlabelled data is readily available and easy to extract, labelled data is scarce and quite costly to obtain. The labelling process is often an onerous, manual task, done by human inspection. It's therefore of utmost importance to find reliable methods which minimize the need for labelled data. Unfortunately, most state-of-the-art methods rely heavily on labelled data and methods to artificially expand this labelled dataset (for instance, the ubiquitous convolution networks require hundreds of thousands of labelled training examples to achieve their unmatched performance). Indeed, most popular and successful methods for image classification are supervised methods, that is, algorithms which require labelled training sets. Since information captured within a visual medium is often rich and complex, it's easy to understand the need for this guided training.

Sadly, completely unsupervised learning- that is, methods focused on using solely unlabelled data - has very limited applications and success. Very few such methods exist, most being forms of clustering or Gaussian mixture models, and neither has, to date, achieve comparable performance to their supervised counterparts. A possible compromise arises in the form of semi-supervised learning; that is, ensembles of classifiers which utilize a set of labelled and unlabelled data. Such methods could tap into the "best of both worlds", making use of the large volume of easy to obtain unlabelled data and a small subset of labelled data. It's been shown that, in various applications (especially in environments where the amount of labelled data available is limited), augmenting the dataset with unlabelled data results in increased performance when compared to utilizing only the labelled dataset. A similar argument could be drawn towards using labelled data from categories or applications which are sufficiently similar to the desired one, albeit not exactly the same. This sort of "sufficiently similar" dataset can, ultimately, be interpreted as a form of "noisy data" from another dataset. This can be done indirectly by utilizing pre-trained models as estimators for the class of images from the target dataset. Such a usage effectively represents a form of transfer learning, where learning by a model from a given dataset is applied to another different dataset. Exploring these various options and comparing their performance on multiple datasets can help understand how the hurdles posed by the scarcity of labelled data can be overcome.

1.2 Main Objectives

The primary goal for this project is to explore the viability of a number of semi-supervised ensembles in the task of image classification. The most relevant questions this investigation aims to answer can be summarized as follows:

1. how viable are pre-trained, out-the-box ConvNet models in assisting semi-supervised learning?
2. how accurate are class scores from these models as estimators for the confidence of classes from different datasets if they are sufficiently similar? How to gauge such similarity?
3. how viable are generative, probabilistic models in assisting semi-supervised learning?
4. does the usage of artificial data generated from the aforementioned models increase the overall performance of the model?
5. how can these different approaches be combined in a single architecture? Is there any increase in performance?

Initially, after some theoretical considerations, an overall scheme for the ensemble was drafted and implemented. This includes the feature descriptors, the feature clustering and extraction processes, the different variations of the final image representation and the classification algorithms. Various parameters pertaining to multiple parts of this pipeline were tested and optimized to both highlight their influence on the overall behaviour of the model and achieve acceptable results. Two primary datasets were used- an initial binary dataset (which acts as a form of "corner case", exemplifying the type of classes and datasets which provide the greatest challenge in terms of accuracy) and a larger, five class dataset. The pictures for either case were obtained from multiple Kaggle competitions.
The ultimate goal for this project was to both carry out a comparative study of some prominent available options and achieve an ensemble which utilizes these that's capable of producing competitive results using as little labelled data as possible for both datasets.

1.3 Document Structure

The subsequent document is divided into three main chapters. Firstly, an overview of both the existing literary work, current investigation trends and theoretical groundwork is presented in section 2. This section is divided in three subsections; the first, which is also divided in multiple segments, lays theoretical details regarding various topics regarding features, feature extraction and description, distance and accuracy metrics and brief mathematical foundations for the different classifiers. The second explores in greater detail the various algorithms used, highlighting their merits and demerits in the context of this investigation and contextualizing them in light of recent trends and state-of-the-art benchmarks. Lastly, the third section will enumerate some of the most recent methods and their results, to be further explored and used as comparison in succeeding sections.

The second main chapter (3) will illustrate and more thoroughly explain the methodology used over the course of this investigation. This includes both a description of the algorithms and ensembles implemented, going into detail about their inner workings, but also a thorough explanation about how their performance is evaluated, and the more relevant hyperparameters which were optimized. This chapter shall also expand upon particular properties of the dataset and how they potentially relate to the aforementioned parameters and overall performance of each model. Preprocessing and multiple techniques used to expand the dataset shall also be covered here.

The final chapter (4) brings it all together by presenting results from the multiple tests, justifying variations in performance (both between different models and/or ensembles, but also within each individual model through variation of the multiple parameters). A comparison to several state-of-the-art solutions is drawn, highlighting any positives or negatives of the achieved solution in this context.

Chapter 2

Background and literature review

2.1 Image model and overview of classification tasks

There are multiple ways to acquire digital images: digital cameras, scanners or magnetic resonances, to name a few. However, when transforming an image to a digital device, what's effectively recorded are numeric values for each point in the image. These points are called pixels. They can be defined as the smallest addressable element of the image (and, by extension, the smallest element that can be manipulated). An image is typically comprised of many channels, each composed of a number of pixels. Therefore, mathematically, an image can be understood as a set of matrices of pixels. The way the information of the image is encoded in these matrices is defined by a colour model and its representation [1]. The two most common are:

- RGB representation, in which the three matrices of an image i, (R_i, G_i, B_i) hold the intensity value of each of the three primary colours for each pixel;
- HSV and HSL (Hue, Saturation, Value/Lightness), in which the three matrices of an image i, (H_i, S_i, V_i) result from geometric transformation of the RGB representation, separating colour information (hue) from brightness (value).

Manipulating an image can be understood as simply as applying a number of functions to these input matrices, producing new, altered output matrices. Image classification tasks can be described as the process of labelling an image; that is, to assign some of many predetermined classes to each new image on the dataset. This is ultimately a learning task. Due to the matrix representation of images in computer vision, a number of machine learning techniques can be readily exploited in this context. There are multiple methods to tackle problems of this nature, and solutions are often comprised of multiple steps: initial image preprocessing, feature extraction, feature clustering and the classification task proper.

2.2 Preprocessing

Usually, in order to both remove noise and simplify the task of feature detection and extraction, images are subjected to some initial transformations. This is called the "preprocessing step". Operations at this stage mostly involve the application of filters and changing details about the image representation (for instance, changing between a RBG space representation to HSV, or simply turning a colour image to a grayscale image). Most of the operations used in the preprocessing stage during this project are described below.

2.2.1 Filters

Since, as seen, an input image is represented by one or more two-dimensional matrices, most filters are applied using multiple matrix products (a filter is, in practice, a bidimensional matrix) locally on the image (filter matrices- or kernels- are of typically small dimension). This simplifies filtering operations computationally and, in the case of separable filters, these matrices can be combined by multiplication into a single matrix to be applied all at once [1]. While filtering removes noise from an image (smooths values across a neighbourhood of pixels, toning down outliers), excessive filtering can tamper with innate properties of the image (edges or colour), so these operations must be used in moderation. Some common filters include:

- Gaussian filters, where each entry is drawn from some normal distribution and is naturally symmetrical around the centre of the kernel, as seen below in equation 2.1 a 3×3 filter kernel example (which would need to be iteratively applied over the entire image, as exemplified in figure 2.1):

$$\begin{bmatrix} 2 & 4 & 2 \\ 4 & 9 & 4 \\ 2 & 4 & 2 \end{bmatrix} \tag{2.1}$$

- Average and median filters, each computing the average or median value on a $k \times k$ region (k odd), and replacing the pixel at the centre by the calculated average/median value. It should be noted that no linear mapping can be drawn for these filters. Thus, they cannot be explicitly approximated by a matrix representation. Whilst this incurs and increased computational cost, it should also be noted that a median filter usually results in an image with crispier edges after noise removal, compared to other filters (an appealing quality if one can spare the associated computational cost).

- Kuwahara and Nagao-Matsuyama filters, both searching for regions of smallest variance in the $k \times k$ window and applying average filtering on that region.

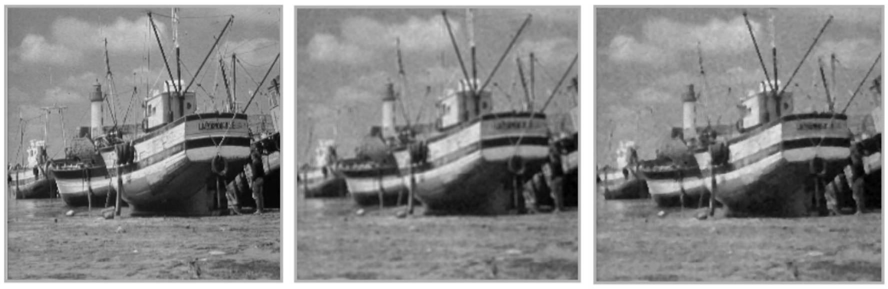


Figure 2.1: Result of the application of a Gaussian filter (middle) and median filter (right). Original image on the left [1]

2.2.2 Gradient calculation and edge detection

The concept of edge comes almost inherently in simple daily life situations, associated with notions of borders or boundaries between regions. However, to properly harness this potential application in segmenting an image into parts, a more careful definition is required. Edges are structures characterized by high variation in one or more directions, and are often located and estimated through calculations of the gradient [1]. While edges as a primary feature for both learning and classification tasks have fallen out of favour next to more abstract feature representations which capture richer sets of information from the input pictures, these still hold a niche and can have merit supporting other features in some applications. Furthermore, as mentioned, edges are useful in roughly segmenting a picture where distinct zones are present. This makes edge detection important for image preprocessing, either to accentuate the edges for further feature detection or to locate regions of interest in the image, to reduce the size of the image to compute in subsequent steps, and at a comparably small computational cost. Edges and corners as features is a topic to be further expanded upon on 2.3. As for estimating edges, as mentioned, computing the gradient over the image and locating regions where it has high values is usually a straightforward and effective solution. Thus, taking the gradient operator as:

$$\nabla I(x,y) = \frac{\partial I}{\partial x} + \frac{\partial I}{\partial y} \tag{2.2}$$

its value is often estimated by application of a mask, similar to what's commonplace with filtering, applied locally and iteratively to the input image through matrix multiplication. The result allows an acceptable estimation of the magnitude and orientation of the gradient and, therefore, of edges themselves. Some of the most common edge detection kernels are:

- The Sobel operator computes the gradient estimation based on applying the following operation:

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.3}$$

Where G represents the gradient, and the two terms $G_{x,y}$ terms are obtained by applying to a 3×3 neighbourhood of the input image the two following masks, respectively:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.4}$$

and

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.5}$$

- The Prewitt operator follows a similar calculation, but uses instead the two following masks:

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.6}$$

and

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.7}$$

- The Canny edge detector is a slightly more sophisticated edge detection technique [2]. This edge detection technique first applies Gaussian filtering (for instance, as exemplified in 2.1) and afterwards draws gradients applying masks similar to those of Sobel or Prewitt operators, rotated in one of eight directions. It then applies non-maximum suppression (that is, entries which are not a maximum are eliminated) and thresholding below one value and above the other (that is, values above an upper bound and below a lower bound are set to white and black, respectively). The surviving values undergo hysteresis thresholding which, in essence, uses information about a neighbourhood to decide whether a pixel is set to white or black (pixels near strong edge values are set to white). Canny detectors take a number of hyperparameters (threshold values, Gaussian parameters for the filters) and are especially common in edge-detection tasks. An example of the output of the Canny edge detector can be seen in figure 2.2.

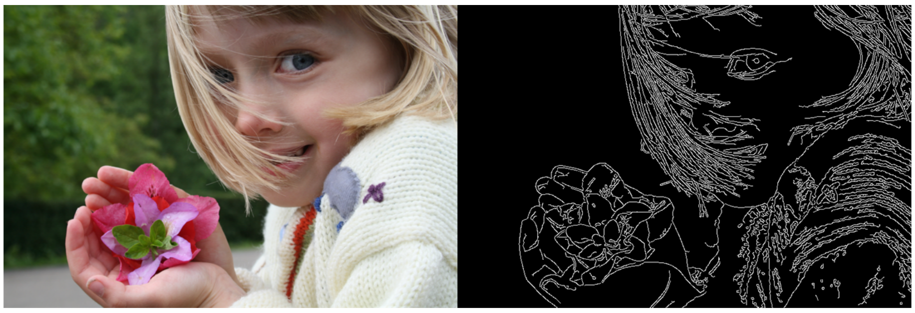


Figure 2.2: Result of the application of the Canny edge detector [1]

2.2.3 Hough transform

A feature detection technique closely related to edge extraction is the Hough transform, which attempts to form imperfect instances of objects or regions within a certain class of shapes through a voting system [3]. A Hough transform can, given a fragmented set of edges, reform them into lines or curves, in particular. This is done by re-parametrizing the input edge objects into a new, more appropriate space, and using accumulators for a multitude of potential final shapes to store their respective votes. Voting can be done based on a number of criteria which is set appropriately for the shape; for instance, in the original formulation for lines, the Hough algorithm would parametrize the input edges into the space of the parameters (Θ, ρ) and vote based on deviation of the distance each point had towards a perpendicular line crossing the origin. The parameters (Θ, ρ) are descriptive of the polar representation $\rho = x \cdot cos(\Theta) + y \cdot sin(\Theta)$ of the line object. The shape with the most votes is the ultimate output of the Hough transform. The results of such a process for a mock example are presented in the figure 2.3. This process is quite important because edge detectors might deliver arbitrarily fragmented edge objects which, on their own, might not have any particular significance and might be undesirable as features. It should be noted that more abstract regions and shapes might not be properly captured by a Hough transform (or may be captured wrongly, due to limitations in the voting kernel).

2.3 Feature learning, feature descriptors and feature extraction

There is no universal consensus for the definition of feature in computer vision, the details being largely dependant on the desired application [4]. However, a general notion is that a feature is a "point of interest" of the image. In classification tasks, these points are uniquely tied to the class of the image- they are distinct from features from another class, and appear across the vast majority of images within the same category. A way to model and understand these desired properties of inter-class uniqueness and intra-class repeatability is linking features to latent variables. While features can be described in a multitude of ways- as simple edges and corners or levels of colour intensity and as complex as an histogram of the frequency of these low-level features- they are often ultimately represented in vectorial or matrix form. Often, features may be of difficult

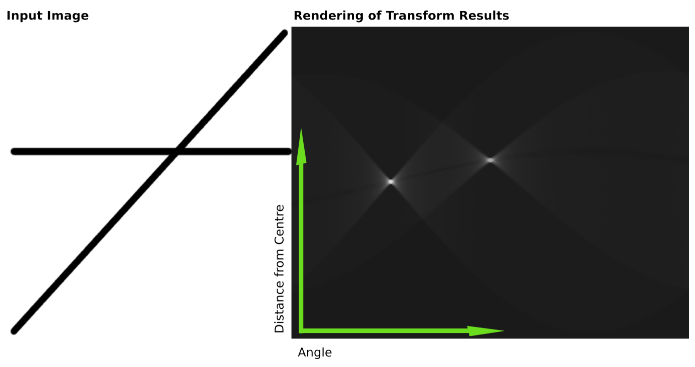


Figure 2.3: Graphic representation of the accumulators for the multiple line objects which attempt to represent the two lines on the left. Note how two white points stand out (the objects which resemble the original lines the most). Image from [55]

interpretation when visualized on their own, and can only be understood in this mathematical formulation. Selecting an appropriate number of features is crucial for any learning or classification task- an insufficient number of features will result in information too scarce to properly model the image class ("underfitting"), whilst an over-abundance of features results in an equally adverse effect of "overfitting" (modelling noise or undesirable factors in a set of pictures which do not accurately represent the image category) [5, 6, 7]. Proper dimensioning of the features also reduces needless computational complexity and costs. It's therefore crucial to identify relevant features and to properly extract them from input images.

2.3.1 Corner Detection

Much as was the case with edges, the notion of corners is relatively intuitive. A way to look at a corner is as the intersection between edges; a zone where two edges "meet". A more formal definition of corner requires the introduction of a more rigorous concept [1]. Letting M be a structure tensor for some portion of an image, I, that is, a window with some area $w(u,v)$:

$$\begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix} \tag{2.8}$$

Where $I_{x,y}$ represents the first order Taylor series approximation of the partial derivatives of $I(u+x, v+y)$, used in the calculation of a sum of squared differences between the patch capture in the original (u,v) window and another patch captured in a shifted window, $w_{shift}(x,y)$. This weighted sum is directly correlated with the M tensor- the eigenvalues of a structure tensor naturally capturing variation of the gradient in a direction associated with the partial derivatives that compose it. Therefore, a corner, which is represented by sharp variations in both directions, can be estimated by values of high and comparatively similar value; that is:

$$(\lambda_1, \lambda_2) : \frac{\lambda_1}{\lambda_2} \approx 1 \tag{2.9}$$

Most corner detectors (such as the Harris corner detector) use this approach to estimate which points do indeed represent corners. While often insufficient on their own to provide a rich enough description of an image for proper learning or classification, corners do facilitate picture matching and are often desirable features in many applications.

2.3.2 Blob detectors

Blobs are a form of feature which comprise a small region which differs in some properties (such as brightness or colour) compared to its surroundings; it can be viewed as an abstraction of a corner. Within the blob, these unique characteristics are mostly constant and uniform, that is, within a blob all points are similar, in some sense [1] (this can be seen in 2.5). Blob detection can be made to be scale-invariant (that is, blobs can be matched between rescaled images) using scale hyperparameters in blob-detection techniques.
Much like corners, blobs are often insufficient as features to characterize an image, but are often used in conjunction with other forms of features for classification tasks. There are three primary methods of blob detection. These are:

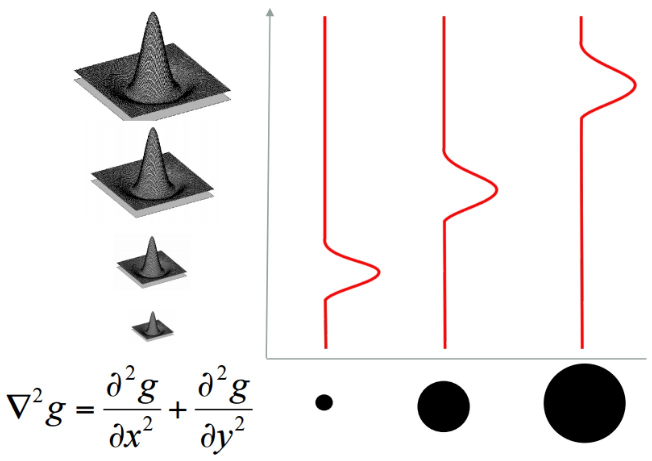
- The Laplacian of Gaussian (LoG) is one of the most common blob detectors. Given an input image, $f(x,y)$, this technique first computes a convolution of said image with a Gaussian kernel:

$$L(x,y|t) = g(x,y|t) * f(x,y) \tag{2.10}$$

Where t represents the aforementioned scale hyperparameter. Afterwards it applies the Laplacian operator, normalized to scale, resulting in:

$$\nabla^2_{norm} L(x,y|t) = t(L_{xx} + L_{yy}) \tag{2.11}$$

and detect local maxima or minima (with respect to space but also scale). Due to this normalization, if one keeps track of the scale parameter, it's possible to detect matching blobs in multiple scales (thus achieving scale-invariant features). The result of the convolution operation can be seen as a scale-space volume $L(x,y|t)$ which is analogous to a scale-space pyramid representation. Noticing that explicit calculation of the Laplacian is computationally expensive and that the scale-space representation $L(x,y|t)$ necessarily satisfies a diffusion equation such as:



$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$

Figure 2.4: Blob detector applied to different resolutions of the same picture. The maxima of the LoG operator shifts with the parameter σ for different scale space representations of the same image

$$\nabla^2_{norm} L(x,y|t) = 2\partial_t L \tag{2.12}$$

- The Laplacian can also be approximated by a difference between to Gaussian-smoothed images (the Difference of Gaussian operator, DoG), as:

$$\nabla^2_{norm} L(x,y|t) \approx \frac{t}{\Delta t}(L(x,y|tp\Delta t) - L(x,y|t)) \tag{2.13}$$

  Which is a sufficient approximation with far less restricting computational requirements. This second technique often provides comparable results. In a scale space representation, the parameter σ of the Gaussian functions ordinarily represents the scale factor.

- A third alternative is to utilize the determinant of the Hessian to estimate these blobs (which has better scale selection properties than the Laplacian operator under affine and geometric operations, compare to the Laplacian operator, even though it incurs a higher computational cost). It should be noted that, much like corners, blobs aren't usually a sufficiently rich feature description for learning, although they can reinforce the process and promote matching accuracy.

2.3.3 Principal Component Analysis

Principal Component Analysis (PCA) is a method that applies an orthogonal transformation to a set of potentially correlated variables, resulting in a set of linearly uncorrelated variables (the

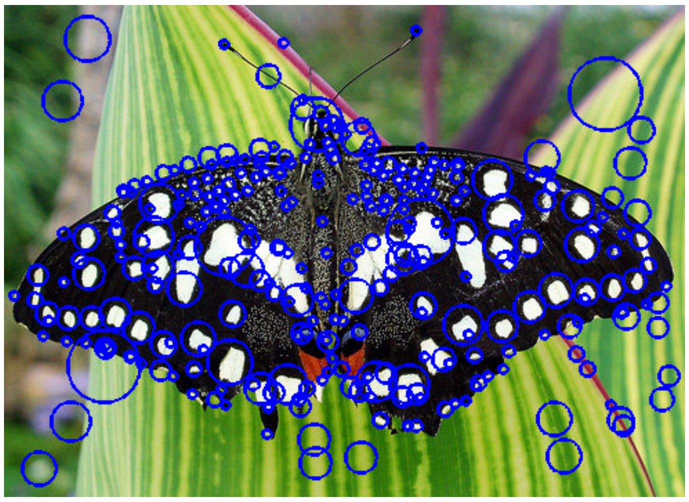


Figure 2.5: Result of the application of a LoG blob detector. Blue circles and ellipses represent the output blobs

eponymous principal components) [8, 4, 9]. Mathematically, if X is the $n \times p$ input data matrix, we wish to find the p-dimensional vector of weights $w_{(k)} = (w_1, ..., w_p)_{(k)}$ such that:

$$t_{k(i)} = x_{(i)} \cdot w_{(k)} \tag{2.14}$$

Where $x_{(i)}$ represents the ith row vector of X, and $t_{(i)}$ the row vector of its corresponding principal scores. The method is such that the first component has the highest value, and each succeeding component takes the largest possible value under the aforementioned restraint of orthogonality. In practice, this means the former components express more "variance" pertaining to the input space. It is also customary to constrain the vector w to have unitary norm. PCA can be achieved rather simply by applying singular value decomposition to the covariance matrix of the input features (or, if possible, eigenvalue decomposition). Letting Q represent the sample covariance between the different principal components:

$$Q \propto X^T X = W \Lambda W^T \tag{2.15}$$

Which, rewritten, yields:

$$W^T Q W \propto W^T W \Lambda W^T W = \Lambda \tag{2.16}$$

Where in Λ each component is represented by a normalized eigenvalue of $X^T X$. PCA can be used to reduce the dimensionality of the input space (for instance, by discarding features which do not represent the variance significantly, and are therefore unlikely to be important towards learning or classification), but can also be used in a preprocessing step to reinforce major features (for instance, to enhance the colour components which are more unique to each feature). One of the most crucial uses of PCA is to reinforce dominant colour components while discarding comparably unimportant colour information in images. If this process is done with some randomness, it can allow to artificially expand labelled datasets, a particularly useful technique in training ConvNets.

2.3.4 Detectors based on histogram representations of gradients and orientations

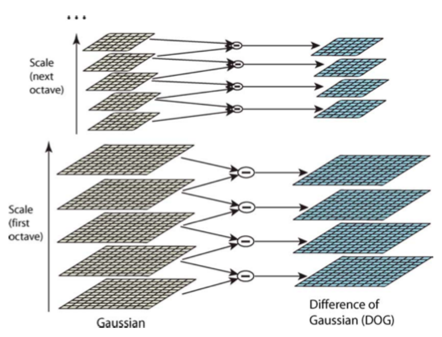
Some feature detection techniques aim to achieve invariance to scale, geometric operations (such as rotations, translations and shearing) and/or photometric changes (such as changes in brightness, for example). This can be partially achieved by histogram representations over a local cell of some size (which attempt to capture local trends and relations instead). The three more common examples of such techniques are:

- SIFT (scale-invariant feature transform, [10]);
- SURF (speeded up robust features, [11]);
- HOG (histogram of gradients, [12]);

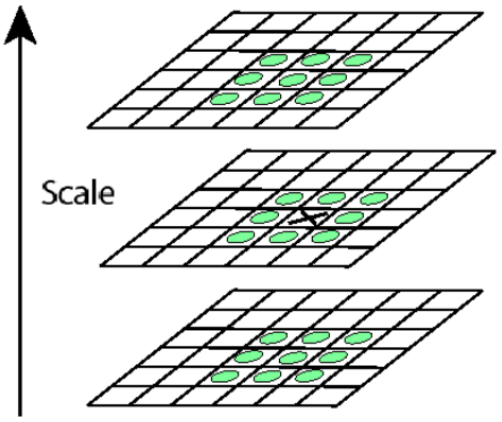
While these differ in the nature of the cell over which the gradients on multiple orientations are captured, or on other intermediate steps to achieve invariance to photometric modifications, all these feature descriptors mostly utilize a similar principle: capture the gradient on multiple directions locally, keep the most representative values and, after carrying this over a cell of some predetermined size, compute the histogram of surviving orientations. The features generated by these algorithms are often impossible to coherently interpret by direct human examination (as exemplified in 2.7) but, computationally speaking, they allow for classification and learning invariant to the aforementioned transformations, which may be desirable. Of the three, SIFT is both the most commonly used and the algorithm which presents the best results in the multiple approaches used in this project. SIFT is comprised, primarily, of the following steps:

1. **Scale-invariant feature detection:** initially, key points are estimated by minima and maxima of a difference of Gaussian kernel applied in scale space to a series of smoothed and resampled replicas of the original image. This step is a scale space analogue to what was described for blob detection in 2.3.2;

2. **Keypoint localization:** the following step filters the set of previously obtained keypoints, both discarding points poorly located along edges (which, as seen, exhibit strong response to the DoG operator) and points which are especially susceptible to noise. By taking the second order Taylor expansion of the DoG operator centred in each keypoint, their accurate location and scale are estimated;



(a) Representation of the DoG kernel applied to a series of Gaussian-smoothed images



(b) Visualization of the search for a match through the scale space, similarly to what's done in blob detection

3. **Orientation assignment:** a key step in achieving rotation invariance, this step computes at the keypoint's scale, σ, the magnitude and orientation of the gradient on a neighbourhood around that keypoint's location using pixel differences, that is:

$$\begin{aligned} m(x,y) &= \sqrt{(L(x+1,y)-L(x-1,y))^2+(L(x,y+1)-L(x,y-1))^2} \\ \Theta(x,y) &= atan2(L(x,y+1)-L(x,y-1), L(x+1,y)-L(x-1,y)) \end{aligned} \tag{2.17}$$

These are then placed in bins for an histogram representation (normally 36 bins, each representing 10^{o}), which are further weighted by a Gaussian of parameter 1.5σ. The highest peaks recorded in this histogram are assigned to that specific keypoint.

4. **Keypoint descriptor:** Lastly, to further bolster the invariance of the keypoints and also make them even more distinctive, histogram representations over small cells are computed on a window around the keypoint location (originally 8 bins for 4×4 cell regions over a 16×16 pixel window). These are then weighted by a Gaussian function of zero mean and 1.5σ. The result is a 128-dimensional descriptor (8 bins per cell, for a total of $4\times4\times8=128$ bins), the typical SIFT descriptor.

PCA-SIFT introduces an additional PCA feature reduction applied on the normalized gradient patch, on the descriptor step which, as seen, can reduce computational costs and maximize information yield in a smaller vector. It should be noted that construction of such descriptors can be computationally expensive. Furthermore, these descriptors typically do not capture well some overall properties of the input pictures (seeing as they apply gradients locally to generate the histogram representation) which might be useful and desirable (for instance, the overall shape or colour of the object). While these are obviously scale-variant or rotation-variant properties, they might still be desirable given the context of the application (although they might naturally be

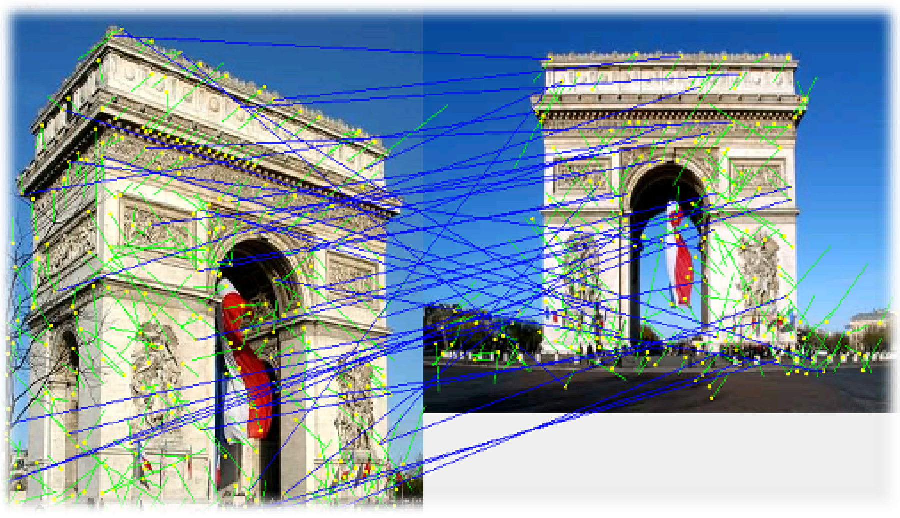


Figure 2.7: Features captured using the SIFT descriptor and matching made with the same scene on a modified picture

extracted by some other method and used alongside these invariant feature descriptors). It's often useful combining SIFT descriptors with other representations that better capture these properties.

2.3.5 Colour histograms

Colour histograms are amongst the most immediate and simple ways to extract colour information as a features [13]. Like any histogram representation, it reflects the frequency of the various colour components in the image. It doesn't necessarily require any form of image segmentation, as it can be applied to the image globally, although one can draw colour histograms for various image patches. Colour histogram are orderless features- within the window in which they're applied, spatial information pertaining to these features is discarded. Furthermore, colour histograms are especially sensitive to noise (any global perturbation in the image will be captured). Colour histograms can be applied to different colour spaces; depending on the space, given K bins, the generated histogram representation can either be K-dimensional (HSV space) or 3K-dimensional (RGB space). Bins usually count pixels that fall within their range, but can easily be construed in more sophisticated way (capture median colours in small pixel ranges, for instance). While mostly invariant to rotation and translation, and easy to match by some metric of distance (for instance, Euclidean distance), colour histograms are quite poor features on their own (it's common for many completely different natural, unadulterated images to yield very similar colour histograms). That said, they are useful and interesting complementary features to most feature descriptors which generally discard most colour information (such as SIFT, for example).

2.3.6 Colour Coherence Vector

Colour Coherence Vectors (CCV) attempt to measure to which extent do pixels with certain colour characteristics are clustered into congruent, coherent regions. The CCV algorithm assigns scores based on the area of regions which hold pixels with similar colour [14] (or, an alternative interpretation, how many pixels with similar colour characteristics exist close together- the eponymous "coherent pixels"). To add resistance to noise and enrich the overall performance of the algorithm, CCV utilizes a "colour resolution range" when determining whether or not a pixel is coherent. That is, CCV checks if pixels are within a certain colour range when comparing it with its neighbours (comparable, to an extent, with bins present in an histogram representation). The overall CCV score of the image- which can be used to measure similarity or dissimilarity for matching and classification purposes- is computed by comparing these histogram-like representation, comparing the coherent and incoherent pixels in each bin. CCV provide a finer set of features than colour histograms (noting the previous claim that it's entirely possible for two different images to have the same colour histogram, this situation is less likely with CCV, mostly due to its enhanced resistance to noise).

2.4 Intermediate feature models for classification

Low-level features and feature descriptors, like those previously described, are often insufficient on their own, either due to their high dimensionality or due to their inability to capture global relation between features in an image. It becomes of interest to find models which encode additional information about the features and their relations and frequencies that also reduce dimensionality and overall computational costs. Some of these models will be explored in this section.

2.4.1 Bag of Visual Words

Bag of Visual Words (BoVW) or Bag of Features (BoF) approaches are characterized by the use of an orderless collection of image features, lacking any structure or spatial information. Due to its simplicity and performance, these approaches have become well-established in the field of computer vision [15, 16]. The name comes from the analogy with the Bag of Words representation used in textual information retrieval. In the text application, distinctive words (in the same sense a visual feature is distinctive: it can be used to classify documents of the same category with some confidence) are clustered into "bags" (using multiple clustering techniques discussed in [83]), which are then subsequently treated as features and used for classification. The primary concept in the visual variant is to consider that an image is composed by "visual words". A visual word is a local segment in an image, defined either by a region (image patch or blob) or by a reference point with its neighbourhood. Feature descriptors like those utilized in the SIFT or SURF algorithm provide good, distinctive features which can be taken as "visual words" after clustering. Further analysis of the frequency of different visual word allows to create a histogram representation the content of each image. A general method for implementing this model is as follows:

1. Computing low-level image descriptors directly from the image (such as the SIFT descriptors);

2. Quantizing the descriptors into clusters, reducing the number of visual words to the centroids of each cluster. These are the final visual words used to form the vocabulary;

3. Representing each image by a vector of frequencies of visual words (analogous to an histogram representation).

Typically, feature vectors yielded by this last step are sparse in nature (especially considering the large dimensionality of most vocabularies). This problem can be mitigated by usage of intermediary techniques as sparse coding, latent Dirichlet analysis or probability latent semantic analysis. Furthermore, overly common features may poorly represent the distinctive characteristics of each class, and therefore it often becomes desirable to penalize these common features. A common weighting technique used in this context is "term frequency-inverse document frequency", or tf-idf. This method applies to each element of a feature vector $F_n = (f_1, ..., f_k)_n^T$, derived from a vocabulary of K words, the weighting term given by:

$$t_i = \frac{n_{id}}{n_d} \log \frac{N}{n_i} \tag{2.18}$$

Where n_{id} represents the number of occurrences of the ith word in image d, n_d the number of features present in that same image and n_i the total amount of occurrences of the ith word in all images of the N images corpus. The BoW model has three essential hyperparameters: the type of feature descriptor, size of the patch in which its applied and the number of visual words present in the final vocabulary (the number of clusters estimated in the clustering step in which the vocabulary is formed).

2.4.2 Spacial Pyramid Matching

Spacial Pyramid Matching (SPM) is an extension of the original, simpler BoW model [17]. SPM aims at capturing spatial information about features (overall location and correlation between feature frequency and location) whilst maintaining the locally orderless architecture of the BoW model. This is achieved by partitioning the image into increasingly fine sub-regions and computing histograms of local features found inside each subregion. In practical terms, at each resolution level, a BoW model is built for each partition. Compared to the original BoW model, SPM achieves significantly better performance, albeit at a heavier computational cost. It should be underlined that, in principle, all information contained by simple BoW model is contained in the

SPM extension, as the 0th level of the SPM model corresponds simply to a BoW model applied to the image as a whole. SPM utilizes a "pyramid matching kernel" defined as:

$$\begin{aligned} k^L(X,Y) &= I^L + \sum_{l=0}^{L-1} \frac{1}{2^{L-l}}(I^l - I^{l+1}) \\ &= \frac{1}{2^L} I^0 + \sum_{l=1}^{L} \frac{1}{2^{L-l+1}} I^l \end{aligned} \tag{2.19}$$

Where l is a resolution level from the overall L resolutions and X and Y are two sets of d-dimensional feature vectors. As stated, each level divides the original image into cells. SPM divides the original image into 2^l cells at each d resolution level, resulting in dimensionality $D = 2^{dl}$ when considering all feature vectors. The function I^l is a histogram intersection function which is applied to the histogram representations of vectors X and Y at resolution level l as follows:

$$I^l = I(H_X^l, H_Y^l) = \sum_{i=1}^{D} min(H_X^l(i), H_Y^l(i)) \tag{2.20}$$

Where the sum is taken across all features at resolution l (so, the overall number of features detected across all cells), yielding the overall number of features present in both histogram representations of the feature vectors X and Y, H_X^l, H_Y^l respectively. It's important to understand that the features matches detected at a level l include all matches at the finer level $l+1$, an argument which can be applied recursively all the way to the finest level L (refer to the scheme in 2.8 for clarity). This justifies the term $I^l - I^{l+1}$ in 2.19, where repeated matches are removed. Furthermore, the weight term $\frac{1}{2^{L-l}}$ simply penalizes matches at coarser levels when compared to matches at higher levels. Intuitively, this means that features at coarser levels are viewed as more "dissimilar" than those at finer levels, although depending on the nature of the problem, this weighting term can be viewed as a hyperparameter of the problem. If the assumption that only similar features can be matched is made, by extension of the BoW model, and if the feature vectors are quantized into one of M types (typically through clustering), the overall pyramid kernel becomes the sum of M individual kernels for each m feature type:

$$K^L(X,Y) = \sum_{m=1}^{M} k^L(X_m, Y_m) \tag{2.21}$$

Like in the BoW model, SPM can also be combined with numerous methods for feature reduction and quantization at this last step, opening interesting possibilities for semi-supervised approaches.

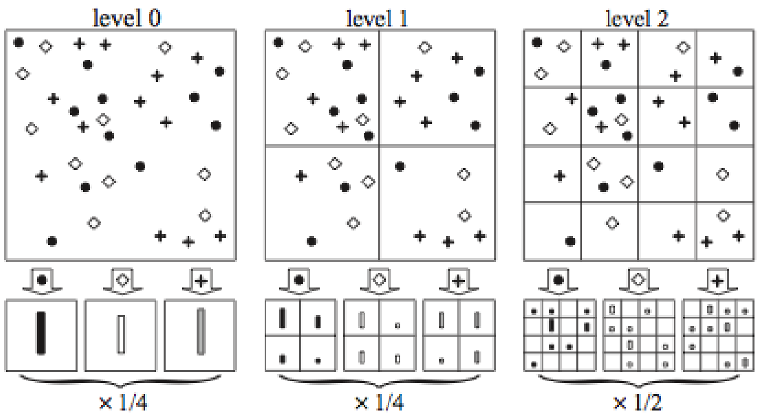


Figure 2.8: Visual representation of the SPM model for 3 pyramid layers with three features, represented by crosses, circumferences and circles

2.4.3 Bag of Colours

The Bag-of-Colors (BoC) model is similar in design to the Bag of Words model, but aims at exploiting color information instead [13]. Instead of utilizing invariant feature descriptors like SIFT, it creates a colour dictionary utilizing information extracted from histogram representations of different colour component in image patches. The output is a histogram of frequency of colours, analogous to the histogram representation of visual words on the BoW model. The model is comprised, generically, of the following steps:

1. Convert the image to an appropriate colour space (often to take advantage of its geometric properties);
2. Compute the most frequent colour components in each patch (16×16 pixel patches in the original formulation), yielding a three dimensional histogram representation;
3. Generate the colour vocabulary through the usage of clustering algorithms (such as K-means);
4. For each image, build a histogram representation of the frequency of "colour words" from the previous determined dictionary;

Much like in the BoW model, the primary hyperparameters are the area and number of patches and the size of the colour vocabulary. BoC can either greatly increase performance or have minimal effect, depending on how valuable colour information is for the classification task, and how regular in colour the category is.

2.4.4 Fisher Vectors

Fisher Vectors (FV) is a method for image representation based on the Fisher Kernel [18, 19]. Summarily, this method first extracts local descriptors from the images and then it fits a GMM to the sampled data. Lastly, it computes statistics regarding each descriptor in pertaining to the GMM. This yields a wealth of information about the structure of the descriptors of the images. It can be seen as a generalization of the BoW model. One of its main attractions is that the GMM fit acts as a form of feature reduction, utilizing comparatively smaller vocabularies and reducing computational complexity. It performs well even with simple linear classifiers. The EM algorithm 2.5.3 is used to optimize the MLE criteria 2.5.2 of the GMM fit. The number of Gaussians used, k, is a hyperparameter of the FV model.

Additionally, FV is often paired with PCA, applied to the local descriptors. This enables further dimensionality reduction. Compared to the BoF model, the Fisher Vector offers a more complete representation of the dataset, as it encodes not only the count of occurrences but also higher order statistics related to its distribution. This richer information captured by the FV model translates into a more efficient representation, since much smaller vocabularies are required in order to achieve comparable performance. Recent experiments also show an improved performance compared to the BoF model in terms of classification accuracy.

2.5 Classification and learning

Learning tasks usually come in one of two primary flavours- supervised or unsupervised learning [20, 7, 21, 4, 6]. These pertain to the type of dataset used for training: supervised learning makes use of labelled data and unsupervised learning makes use of unlabelled data. Generally speaking, the former tends to outperform the latter, primarily due to the richer nature of the learning set. This comes at a price, however - labelling is primarily done through human action, and can be both terribly costly and inefficient. Hence the attractiveness in unsupervised learning, shifting part of that cost to potentially worse performance, but ultimately sparing the pricey, onerous task of manually labelling large datasets. By nature, unsupervised learning is often associated with clustering, although these are not synonymous. A third type of learning is semi-supervised learning- standing in-between these, semi-supervised approaches tend to use a small, labelled set reinforced with a wealth of unlabelled, cheap data. Semi-supervised approaches tend to outperform either purely supervised or unsupervised learning (in virtue of making use of a larger dataset by usage of unlabelled examples, but also taking advantage of the guidance provided by properly labelled samples during the learning process), which further makes these solutions attractive. It's common to assume labelled samples are independent and identically distributed (i.i.d.), a supposition which holds for the majority of applications. In a Bayesian interpretation of the semi-supervised approach, for some probability density function, we have:

$$f_\Theta(X) = \arg\max_\Theta log(P((X_i,Y_i)^l|\Theta) + \lambda P((X_j)^{l+u}|\Theta)) \tag{2.22}$$

Where λ is a parameter which represents the weight (that is, overall relevance) attributed to the unlabelled data in the learning task, Θ the vector with the parameters of the model and (X_i,Y_i)

a feature vector with its respective label. This generic approach to generative modelling is quite important when fitting cost functions in semi-supervised approaches.

2.5.1 Generative vs non-generative models

For a classification task, given a feature vector, X_i, there are two primary learning models: generative and discriminative models [4, 9, 22]. Generically, given X_i, we wish to estimate:

$$P(C_k|X_i) = \frac{P(X_i|C_k) \cdot P(C_k)}{P(X_i)} = \frac{P(X_i,C_k)}{P(X_i)} \tag{2.23}$$

Depending on the methodology employed, one can either estimate $P(C_k|X_i)$ directly from the given data or attempt to infer $P(X_i,C_k)$, the joint probability density function of latent variables expressed by the extracted features. The former are called "discriminative methods" whilst the latter are called "generative models". Generative models attempt to model and learn the underlying density function of the class. These capture a much broader breadth of relations between the various present variables and allow to paint a mathematically richer formulation of the categories as a whole (indeed, "generative models" owe their name to their ability to generate samples due to inferred knowledge about the probabilistic nature of the phenomena they model). This comes at a cost, however, as this information is often harder to infer (due to inherent complexity and simply due to being less readily available, especially in regards to training examples) and computationally more expensive. Another issue which further compounds these difficulties is that in the field of computer vision, there's often a lack of a proper "negative sample". To exemplify, if one considers the problem of binary classification, one would evidently need labelled training examples for each class to effectively learn an underlying probabilistic model. But in the field of computer vision, this concept is often blurred- if one wishes to learn a shape (for instance, human faces), the concept of "negative sample" isn't clear. This contrasts heavily to non-probabilistic models. In their simplest interpretation, discriminative models are concerned with drawing a decision boundary which allows them to satisfactorily classify unlabelled input feature vectors into one of many classes. While this approach abstracts underlying statistical models, it's been shown to be more readily applicable and, in fact, have a comparatively higher success rate in terms of classification tasks. However, since little additional knowledge about the classified object is drawn, these approaches can behave poorly in other relevant aspects of computer vision, such as transfer learning (that is, these discriminative models often are problem-specific and offer very little help to other, potentially related problems, or even any form of scene-interpretation). Using both approaches together to ultimately draw a fuller model for a scene is a challenging but interesting approach which has seen some success in recent years, compared to purely discriminative or generative solutions.

2.5.2 MLE and MAP estimation

Maximum Likelihood Estimation (MLE) and Maximum A Posteriori probability estimation (MAP) are the two more common methods for estimating the parameters of a statistical model given data [4, 9]. Given a population of $(x_1,...,x_n)$ i.i.d. elements, MLE defines a likelihood function for the model Θ as:

$$L(\Theta;x_1,...,x_n) = f(x_1,...,x_n|\Theta) = \prod_{i=1}^{n} f(x_i|\Theta) \tag{2.24}$$

Maximization of the logarithm of this function yields a MLE estimator of the parameter Θ as $\hat{\Theta}_{ML}$. Noting that the logarithm is a monotonically increasing function, computations can be further simplified by replacing products with sums by maximizing the logarithm:

$$L(\Theta;X) = \sum_{i=1}^{n} \log f(x_i|\Theta) \tag{2.25}$$

The canonical form of the likelihood function. Often, a closed-formed solution for this estimator can be found as an explicit function of the observed data, which is a desirable property (though this needn't be the case, and numerical approximations might be required). MLE is usually algebraically straightforward and computationally simple. A MLE estimator is also consistent (arbitrary precision can be achieved through an also arbitrary increase in the number of samples), thought it's not necessarily unbiased. The MAP estimation runs on a similar algorithm, but further enhances this estimation by utilizing information pertaining to the prior probabilities, $g(\Theta)$, as:

$$\hat{\Theta}_{MAP} = \arg\max_{\Theta} f(x|\Theta)g(\Theta) \tag{2.26}$$

This method is typically preferred if there's available information about $g(\Theta)$ (which can often be estimated if there's a large enough sample size, assuming a i.i.d. population). A MAP estimator is also derived from a Bayesian interpretation of the model, being specifically useful for generative models based on a similar formulation.

2.5.3 EM algorithm

The Expectation-Maximization algorithm (EM algorithm) is a common iterative method for finding both MLE and MAP estimations of parameters in statistical models [23]. Simply put, the algorithm firstly computes an estimation of the log-likelihood function (the E-step), evaluated using the current approximation of the parameters Θ, followed by a maximization step (M-step), which recalculates the Θ model parameters using the previously estimated log-likelihood function. Mathematically, taking the log-likelihood function (as defined in 2.25) $L(\Theta, X) = \sum_{i=1}^{n} p(X,Z|\Theta)$, where X represents the observed data, Z the latent variables or unobserved data and Θ the model parameters, we have:

- **E-Step**: $Q(\Theta|\Theta^{(t)}) = E_{Z|X,\Theta'}\left[\log L(\Theta;X,Z)\right]$
- **M-Step**: $\arg\max_{\Theta} Q(\Theta,\Theta^{(t)})$

Where Q is some distribution that's proportional to $p(\Theta;X,Z)$, that is: $Q \propto p(\Theta;X,Z)$. This two step process is repeated until the derivatives of the likelihood function are arbitrarily close to zero, indicating (potentially local) maxima (this resolution parameter being used as the stop condition for the algorithm). The EM algorithm can effectively find approximations of the MLE and MAP parameter estimations where explicitly solving the associated equations is not possible due to unknown latent variables or lack of a closed-form solution (for example, in a mixture model, it's often assumed that each data point as at least one hidden variable which associates it with a certain class). The EM algorithm is therefore especially useful for generative learning. The algorithm, whilst both simple and useful can return nonsense maxima (in the same example of a mixture model, a maxima which corresponds to a case of zero covariance).This is a result of the underlying MLE process in the EM algorith. This requires special care when deciding upon estimators. Different initialization of the algorithm may result in convergence to different maxima, and therefore a sweep of these hyperparameters may be conducted if desired.

2.5.4 Naive Bayes hypothesis

Naive Bayes classifiers are a family of learning algorithms which are based on applying the Bayes' theorem assuming conditional independence between variables- the Naive Bayes hypothesis [4, 9, 21]. The Naive Bayes hypothesis ultimately approximates the probability density $P(C_k;x_1,...,x_n)$ as:

$$P(C_k;x_1,...,x_n) = P(C_k) \cdot \prod_{i=1}^{n} P(C_k|x_i) \tag{2.27}$$

The Naive Bayes hypothesis can be used in conjunction with other learning methods, allowing to group multiple variables into a joint probability distribution, abstracting it and allowing it to be treated as one single PDF. It often simplifies intermediate calculations, allowing to reduce the dimensionality of the problem (as a concrete example, for normal distributions, the covariance matrix becomes merely a diagonal $N \times N$ matrix, thus holding N elements). The conditional independence assumption, while often useful and providing sufficient approximation, must be used carefully; this is a powerful assumption, which, if misused, may cause important loss of information at best, and induce errors downright at worst.

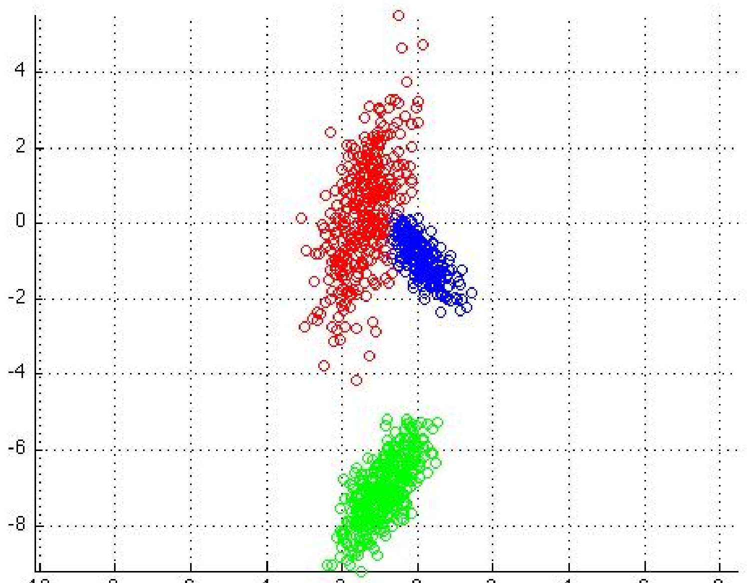


Figure 2.9: Subpopulations captured through a Gaussian mixture model utilizing the EM algorithm

2.5.5 Mixture Models

A mixture model is a model for representing the presence of subpopulations within a large dataset, without necessarily requiring information identifying to which of these subpopulations the data points belong [24] (see 2.9). The nature of such a "subpopulation" can be abstracted to any type of class within a given context. Mixture models attempt to model each of these subpopulations to a probability density function of a given family and are therefore generative models. Common density functions include Gaussians, multivariate Gaussians, multinomial and exponential distributions. Mixture models take a large number of hyperparameters, such as (K,N), the number of mixture components and observations, respectively, and (Θ_k,Φ_l), the parameters of each distribution for each k probability density function and its respective weight. Mixture models may be reformulated as a form of clustering, in which unsupervised learning is carried out, although this is not necessary. While limited in the fact that mixture models cannot correctly model subpopulations which follow different probability density functions, they're still frequently used in cases where one has some confidence about the uniformity of the nature of the subpopulations.

2.5.6 Hidden Markov Models

Hidden Markov Models (HMMs) is an extension of a Markov Model where it's assumed a number of hidden variables which represent the state of the system control the emission of output tokens (a generic state diagram with the corresponding emissions can be seen in 2.10). It can be viewed as an extension of a mixture model which does not assume variables to be independent; instead,

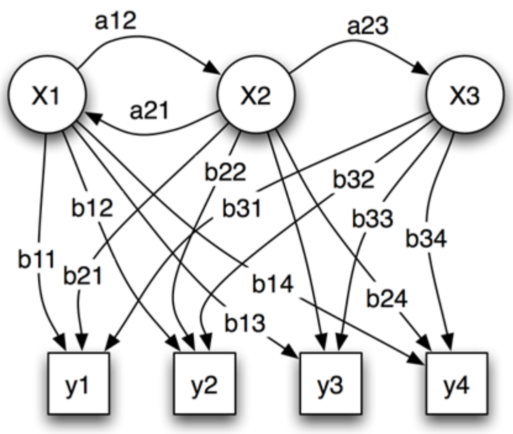


Figure 2.10: Visual representation of a chain modelled by a HMM

these are related by a Markov process. HMMs require, in essence, two main matrices of hyper-parameters, a matrix A and B, which hold in each entry the probabilities for the emission of an output token (thus, the "emission matrix") and the probabilities of changing between states (the "state transition matrix"). HMM learning is usually compounded with usage of the EM algorithm to derive ML iterative estimations of the various parameters, updated with each new observation of output tokens. HMMs are incredibly versatile; despite their original formulation in the context of filtering, these can translate multiple relations between multiple observations (including temporal ones) [21, 25, 26].

2.5.7 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is a generative statistical model which attempts to explain sets of similar observations through unobserved groups. It's typically utilized in semi-supervised or unsupervised contexts [27, 28, 29]. The input dataset is grouped in different "topics" which attempt to model similarities within the observed variables. For example, applying LDA to a set of feature vectors extracted from images of cats and dogs could cluster them into two topics, "dog related" and "cat related". An alternative way of interpreting the model (its dual formulation) is understanding that each topic has some probability of generating the different features (a "cat-related" topic would be more likely to generate features associated with cats in comparison to dogs). Topics are generally poorly defined entities, and quite similar to the "bags of words" in the BoW model previously introduced. LDA assumes that priors follow a Dirichlet distribution. It takes a series of parameters, such as:

1. α, parameter of the Dirichlet distribution prior on the per-document topic distributions;

2. β, parameter of the Dirichlet prior on the per-topic word distribution;

3. Θ_i, the topic distribution of the ith document in the corpus;

4. ϕ_k, the word distribution of the kth topic;

5. $z_{i,j}$, the topic for the jth word in the ith document;

The LDA model has the various $w_{i,j}$ words as observables variables- all other variables are latent (like the topics themselves). The LDA generative process for a corpus of M documents with length N_i, D, assumes that each document is represented as a random mixture of topics, and can summarily be described as:

1. Choose

2. $\Theta_i\ Dir(\alpha)$ for $i \in 1,...,M$;

3. Choose

4. $\phi_k\ Dir(\eta)$ for $k \in 1,...,K$;

5. For $w_{i,j}$, with $i \in 1,...,M$ and $j \in 1,...,N$:

   - Choose a random topic $z_{i,j}\ Multinomial(\Theta_i)$;
   - Choose a random word $w_{i,j}\ Multinomial(\phi_{z_{i,j}})$;

Lengths are considered independent variables during this process. Learning the multinomial distributions is done through the EM algorithm and Gibbs sampling. LDA is often used as form of intermediary feature reduction (for instance, tied with a BoW model). However, with access to the multinomial distribution, artificial topic vectors may be generated by random draws.

2.5.7.1 Probabilistic Latent Semantic Analysis

Probabilistic Latent Semantic Analysis (pLSA) is a particular case of LDA for a uniform Dirichlet distribution, based on latent class modelling. Like LDA, it can be used as a form of feature reduction and also yields a multinomial topical distribution [30, 27, 31]. Considering observations in the form of co-occurrences, (w,d), described as above, the model can be summarily presented as:

$$P(w,d) = \sum_z P(z)P(d|z)P(w|z) = P(d)\sum_z P(z|d)P(w|z) \tag{2.28}$$

With the former formulation being called the symmetrical formulation ((w,d) generated by z) and the latter the asymmetrical formulation. The asymmetrical formulation has an interesting interpretation: for each document d, the most likely latent topic z is chosen, which in turn is reflected on the words w generated by that topic. PLSA has been used in conjunction with Fisher's kernels or SPMs in semi-supervised discriminative settings to some success.

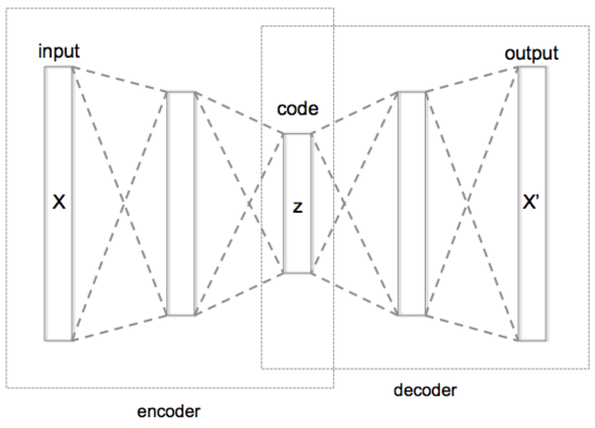


Figure 2.11: Visual representation of a three hidden layer auto-encoder and decoder. Note the resemblance to a multi-layer perceptron net

2.5.8 Sparse Coding

Sparse coding is an unsupervised method of feature remapping and reduction. It shares similarities with a multi-layer perceptron net. It changes the representation of the input layer into a sparse representation that has lower dimensionality and results in increased accuracy [32], as visualized in 2.11. Sparse encoders try to minimize the loss function:

$$L_{sc} = ||WH - X||_2^2 + \lambda ||H||_1 \tag{2.29}$$

Where, with W being a matrix of bases and H a matrix of codes, the former term is called the reconstruction term and the latter the sparsity term (which, respectively, handle optimization of the code itself and penalize excessive sparseness). In practice, this problem can be quite challenging to solve. However, satisfactory approximations to the solution can be obtained the auto-encoder cost function:

$$L_{sc} = ||W_{\sigma}(W^T X) - X|^2 \tag{2.30}$$

Which is much cheaper to handle computationally.

2.5.9 Clustering

Clustering is a semi-supervised of unsupervised task of grouping objects from a set into smaller, subsets in a way that objects in each of these subgroups display greater similarity compared to all the other objects. It's one of the most primary tasks in data mining, statistical analysis and machine learning [33, 34, 4, 35, 36]. Clustering isn't an algorithm in itself, but includes a family of methods to undertake the aforementioned clustering task, differing on how clusters are defined and how they are formed. Popular methods include clustering data points based on proximity using some distance metric, partitioning the input space into dense regions or section it according to probability density functions. Some types of clustering include:

- Centroid models, such as k-means clustering;
- Connectivity models, such as hierarchical clustering;
- Distribution models, such as the previously illustrated mixture models (2.5.5);

2.5.9.1 K-means clustering

K-means clustering is a method of vector quantization that aims to partition n observations into k clusters, each observation belonging to the cluster with the nearest mean, which ultimately identifies the cluster itself [37]. While in its original formulation, K-means is a NP-hard problem, heuristic methods can be employed to allow quick convergence to a local optimum. GMM models utilizing the EM algorithm are quite similar to some formulations of K-means clustering. K-means clustering is also relevant due to its close relation to the k-nearest neighbour classifier (classification of new points in K-means clustering can be used by searching which of the clusters is the nearest neighbour of that same point). The algorithm is quite simple. Considering a set of observations, $(x_1, ..., x_n)$, the objective function is a minimization of square error pertaining to each cluster, that is, the within-cluster sum of squares (WCSS), defined as:

$$\arg\min_S \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\| \tag{2.31}$$

Where S represents the set of $k < n$ clusters, each identified by their mean μ_i, the algorithm run two primary steps:

1. assign each data point to the nearest cluster through the WCSS as seen before. Since the sum of squares is the squared Euclidean distance, this is the metric typically used to decide which cluster is "nearest";
2. Update the means of each cluster based on the data points it contains:

$$m_i^{t+1} = \frac{1}{|S_i^t|} \sum_{x_j \in S_i^t} x_j \tag{2.32}$$

Noting that the arithmetic mean is also a least-squares estimator. The primary hyperparameter to consider when using K-means clustering is the number of clusters, k. Initial seeding for each cluster also warrants some study, as K-means can produce nonsense results due to convergence into local minima.

2.5.9.2 Hierarchical clustering

Hierarchical clustering seeks to build a hierarchy of clusters using one of two approaches:

- **Agglomerative**: a "bottom-up" approach, where multiple clusters are paired up and merged as one moves up the hierarchy;
- **Divise**: a "top-down" approach, where all observations start in one single cluster which is successfully divided into smaller clusters;

In order to decide which clusters are merged or split, their dissimilarity is measured. This is done by properly selecting a metric (as seen in), and using one of multiple criteria for linkage, such as maxima, minima or average of linkage clusterings.

2.5.9.3 Nearest Neighbour Search

Nearest neighbour search is an optimization problem which, as the name implies, aims at finding the closest points to some centre. "Closeness", in this context, is expressed through some generic dissimilarity function: for some metric space M, the dissimilarity function acts as a distance metric (not necessarily geometric, although these are also acceptable).

Of the many implementations of nearest neighbour search, FLANN (Fast Library for Approximate Nearest Neighbour) is often chosen due to its comparatively fast convergence to an acceptable solution, being computationally light (often being considered comparable to state-of-the-art for most applications).

2.5.10 Linear and Logistic regression

Out of all regression models, the most used in learning are linear and logistic regression models. Linear learning attempts to model a relation of the form:

$$Y = W^T X + W_0 \tag{2.33}$$

It can be shown (by computing the derivative of the LSE function applied to this decision boundary) that the optimal solution for the least-squared error loss function is:

$$W = (X^T X)^{-1} X^T Y \tag{2.34}$$

Linear regressions are rather versatile techniques [38]. They can be reformulated as a method for ML estimation or to make usage of a Bayesian formulation, despite their simplicity (the only costly operation being inversion of a potentially large matrix). In the case of logistic regression, the boundary is given by:

$$F(X) = \frac{1 + exp(W_0^1 + W^1 X)}{1 + exp(W_0^0 + W^0 X)} \tag{2.35}$$

Either regression captures relations between hidden variables of varying formulations. Whilst relatively limited in usage and effectiveness, regression models are versatile and can compliment other, more complex model ensembles.

2.5.11 Support Vector Machines

Support Vector Machines (SVMs) are an extension of simple linear regression models by attempting to find the most noise-resistant and robust model [39, 7, 40, 6]. Let us consider a binary classification task with labels $(y_0, y_1) = (-1, 1)$. Defining a generic decision boundary with some bias term b:

$$\gamma_[i] = y_i(w^T x_i) + b \tag{2.36}$$

One should notice that multiplying the boundary by the class label y_i simplifies interpretation, as a positive product reflects a properly labelled class if the coordinates are shifted as to make boundary surface the line $y = 0$. SVM achieves robustness by optimizing a margin between the decision boundary and the data points. The concept of "margin" is fairly intuitive, but carefully defining it with proper mathematically formulation is of utmost importance. Let us assume, for the sake of brevity and with no loss of generality, that the bias term $b = 0$. We can identify a functional margin as:

$$\hat{\gamma}^i = y_i(w^T x_i) \tag{2.37}$$

It's obvious that to achieve a large margin we need $w^T x_i$ to have large magnitude and the same sign as y_i. A problem with this formulation arises that prevents it from accurately representing the distance of each x_i to the decision hyperplane: the distance metric is poorly defined. One could arbitrarily change the magnitude $||w||$ without influence on the classification itself. Therefore, to obtain a metric that's invariant to these multiplicative factors, we move away from functional margins towards a "geometric margin", defined as:

$$\hat{\gamma}^i = y_i(\frac{w}{||w||}^T x_i) \tag{2.38}$$

Furthermore, we're interested in finding the smallest geometric margin for the whole dataset (intuitively, this smallest margin is the limit to th noise resistance of the model):

$$\gamma = \min \tilde{\gamma}^i \tag{2.39}$$

We can now tackle the maximization problem of finding the decision hyperplane which yields the maximum geometric margin. What is actually done, however, is converting this maximization problem to its dual (if the set is convex, although this is common in the usually binary features obtained from visual vocabularies), resulting in a minimization problem:

$$\begin{aligned} &\gamma := \min_{\tilde{\gamma},w} \frac{\|w^2\|}{2} \\ &Subject\ \ to: \\ &y_i(w^T x_i) \geq 1 \end{aligned} \tag{2.40}$$

This can be formulated as a minimization quadratic program problem with linear constraints. Since it's a minimization problem over a surface, the method of Lagrange multipliers can be used to derive a solution. This yields the Lagrangian function:

$$\min L(w,b,\alpha) = \|w^2\| - \sum_{i=1}^{n} \alpha_i(y_i(w^T x_i + b) - 1) \tag{2.41}$$

For a bias b, where α_i are the Lagrange multipliers. This is the classic linear SVM cost function, in dual formulation. Recent years brought slight variations to this formulation; particularly the sub-gradient descent SVM works directly with the formulation:

$$f(w,b) = \|w^2\| + \sum_{i=1}^{n} \big[\frac{1}{n} \max(0, 1 - \sum_{i=1}^{n} \alpha_i(y_i(w^T x_i + b)))\big] \tag{2.42}$$

Which utilizes gradient descent algorithms to speed up convergence towards a solution. Solving the SVM cost function requires calculations and inversions of large matrices, a potentially computationally costly process. Furthermore, implementing on-line learning with SVMs, whilst possible, is a complex affair. Despite these limitations, models using SVMs can achieve outstanding performances, having also the advantage of requiring only a small number of parameters to be stored (the so called "support vectors"). Classifying a data point is also a trivial affair, requiring only usage of said support vectors.

2.5.11.1 Slack variables and non-linearly separable datasets

SVMs can only achieve a solution in the previous formulation if the data is linearly separable (that is, if all points in the dataset can be classified correctly). Evidently, this is often impossible in practice. A method to overcome this is to introduce slack variables, ε_i, which attempt to quantify the cost of misclassifying a point compared to the overall robustness of the margin [41] (that is, how much is a larger margin worth versus a smaller number of misclassified points). Introducing these in the previous formulation yields:

$$\begin{aligned}
&\gamma := \min_{\tilde{\gamma},w} \|w^2\| - C\sum_{i=1}^{n}\varepsilon_i \\
&Subject\ \ to: \\
&\varepsilon_i > 1 \\
&y_i(w^T x_i) \geq 1 - \varepsilon_i
\end{aligned} \tag{2.43}$$

With C representing a multiplicative cost factor, reliant on the overall problem. This ultimately results in the following cost function:

$$\begin{aligned}
&\min L(w,\beta,\alpha) = \max\sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}\sum_{i=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j \\
&Subject\ \ to: \\
&0 \leq \alpha_i \leq C \\
&\sum_{i=1}^{n}\alpha_i y_i = 0
\end{aligned} \tag{2.44}$$

In complex models with multiple different methods, this cost parameter can be used to reflect estimations and certainties from different tiers of the overall model.

2.5.11.2 Non-linear classification and the "kernel trick"

More difficult datasets may result in very unsatisfactory solutions even with the introduction of slack variables. This often indicates that the dataset may not be well-modelled by a linear model. SVMs overcome this limitation by implementing the "kernel trick". that is, transforming the input feature space utilizing some form of non-linear kernel (such as a Gaussian kernel). This can easily improve performance on non-linearly separable datasets. Applying transforms the original linear discriminant function through the aforementioned kernel function:

$$\max g(x) = w^T x_i + b \tag{2.45}$$

Becoming:

$$\max g(x) = w^T \Phi(x_i) + b \tag{2.46}$$

Most common kernel functions include polynomial functions, Gaussian functions and χ^2 kernels.

2.5.11.3 Semi-supervised variations of SVMs

SVMs can be further reformulated to endorse a semi-supervised approach, typically called "S3VM", standing for Semi-Supervised Vector Machines. One of the more popular options is TSVM, Transductive Support Vector Machines [42, 43]. TSVMs utilize the same formulation and cost functions as normal, supervised SVMs, but apply transduction to an additional, unlabelled dataset together with the primary, labelled training set, D^*:

$$D^* = (x_i^* \| x_i^* \in R^p)_{i=1}^k \tag{2.47}$$

2.5.12 Deep Convolutional Networks

Currently the state of the art in many image recognition and classification tasks [44, 45, 46], deep Convolution Networks (ConvNets) implement a complex ensemble of linked perceptron layers built on top of a structure which learns abstract local features by applying convolution filters to windows of the input matrix (the so-called "receptive field" of the convolutional neuron). ConvNets are capable of learning a number of abstract features and even relations between features in their successive convolution layers, completely automating the feature description and extraction process. The final, dense layers apply successive transformations to the input feature space as to make it as linearly separable as possible, allowing for incredible accuracy [47] (the most recent architectures are on-par with human manual classification). Some of the more relevant ConvNet architectures include:

- LeNet, the first deep ConvNet successfully applied to vision and classification tasks;
- AlexNet [48], a 2012 architecture which built upon the original LeNet. It popularized ConvNets by reaching record performances in the ImageNet challenge (under 15% error rates), as present in the scheme in 2.12;
- ZFNet [47], a 2013 extension of AlexNet which optimized various hyperparameters to achieve superior accuracy;
- GoogLeNet [49], a 2014 ConvNet built on a computational budget, reaching competitive performance with limited computational resources by using a comparatively smaller model (less than one tenth of the parameters present original AlexNet) and a longer training schedule;
- VGGNet, another architecture starred in 2014 which explored the importance of the depth of the network as a hyperparameter. This large, very deep 100 million parameter featured

impressive performance, but required hefty computational resources and a comparatively long time to achieve accurate results;

- ResNet, a recent, 2015 architecture, which introduced skip connections and heavy use of batch normalization. Despite impressive results, very few public implementations of this architecture are available at the time of writing;

These models can ultimately be reduced to four main components (visual reference in 2.13a):

- Conv layers, which apply filters to the input layer, computing the output of the neurons connected to the images in the dataset. This layer tends to have the most tiers;
- ReLu layers, applying a non-linear activation function of the form $\max(0, x)$. This does not change the volume of the layers, and is typically used in-between Conv layers;
- Pooling layers, which apply downsampling to a previous layer. These reduce a $k \times k$ layer to a $(k-n) \times (k-n)$, for $n < k$;
- Fully-connected (or dense) layer, comprised of fully linked perceptrons which transform feature vectors output from the Convolutional portion of the networks into proper classes;

While unchallenged in respect to performance, ConvNets have a large number of associated, heavy costs [50]. Training ConvNets is an onerous and extremely expensive task- often requiring a number of high-end GPUs working in parallel and a very high amount of RAM memory (dozens of gigabytes). ConvNets are also massive models, with millions of parameters, requiring a large amount of resources to be stored and used. And lastly (and perhaps more worryingly), ConvNets are easily prone to overfitting, requiring extremely large datasets to be fully trained, or complex data-boosting techniques [51]. To mitigate these limitations, it's often a common approach to use one of various available, pre-trained models to accelerate the learning process and relax the high dataset requirements. It should be noted, however, that utilizing pre-trained models as a starting point incurs some risk. It can be shown, through statistical methods, that the more dissimilar the categories for the pre-trained and desired tasks are, the more likely is the ConvNet to under-perform. Furthermore, utilizing pre-trained models often instils constraints upon the ConvNet architecture. At the moment of writing, of all discriminative models, ConvNets present the best potential accuracy.

2.5.12.1 Backpropagation

Backpropagation is a gradient descent method utilized in most types of neural networks. Exploring a generic network with some number of hidden, intermediary layers and considering the LSE loss function:

$$E(\omega) = \frac{1}{2}(t-y)^2 \tag{2.48}$$

Where t is a value or label for the training point and y the corresponding output of the network (stressing that this value y is the actual output of the network, i.e.: a Real number, and not a label of any sort). Computation of the derivative of this cost function is necessary to estimate the gradient and update the weights given a learning rate η. Taking the output of a neuron L in a layer k to be $z(x,\omega) = z\left(\sum_{j=1}^{N} \omega_j h_j(x)\right) = z(\phi_j)$, for weight ω_j and neuron output $h_j(x)$, the derivative can be obtained by applying the chain rule twofold:

$$\frac{\partial E(\omega)}{\partial \omega_{ij}} = \frac{\partial E(\omega)}{\partial z_j} \frac{\partial z_j}{\partial \phi_j} \frac{\partial \phi_j}{\partial \omega_{ij}} \tag{2.49}$$

Remembering that the derivative of the sigmoid kernel can be easily reduced to:

$$\frac{\partial z(\omega x)}{\partial \phi_j} = z \cdot (1-z) \tag{2.50}$$

And letting, for the jth neuron, δ_j be:

$$\delta_j = \frac{\partial z_j}{\partial \phi_j} \cdot \sum_{i \in L} \omega_i \delta i \tag{2.51}$$

The following expressions are obtained:

$$\frac{\partial E(\omega)}{\partial \omega_{ij}} = \sum_{i \in L} \delta_l \omega_{il} \cdot z_i(1-z_i) \cdot \hat{z}_i \tag{2.52}$$

And:

$$\frac{\partial E(\omega)}{\partial \omega_{ij}} = -(t-y) \cdot \hat{z}_i = \delta_{output} \cdot \hat{z}_i \tag{2.53}$$

For the hidden and output layers, respectively, where $\hat{z}_i$ represents simultaneously the input of the neuron in the current layer and the output of the neurons in the preceding layer (equivalent to x_i if the layer in question is the hidden layer immediately succeeding the input layer). It should be noted that the derivative $\frac{\partial E(\omega)}{\partial z_j}$ cannot be trivially calculated for a neuron in an arbitrary hidden layer; it can, however, be iteratively calculated using the total derivative with respect to z_j if one notes that:

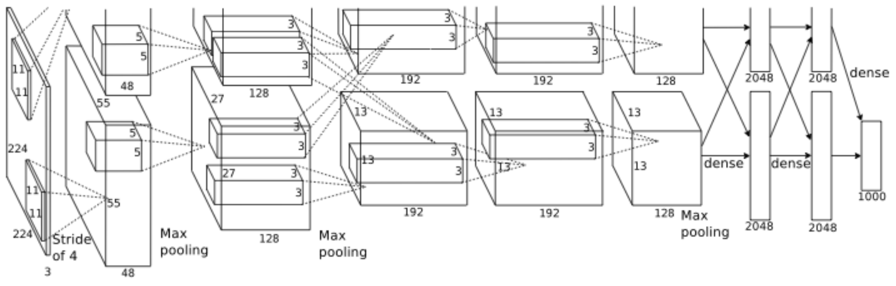


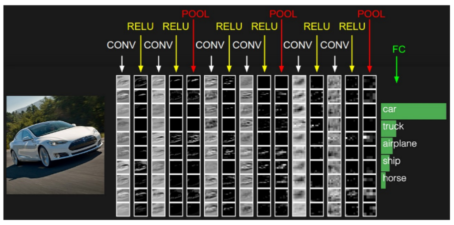
Figure 2.12: Visual representation of the popular AlexNet architecture for a deep ConvNet [48]

$$\begin{aligned}\frac{\partial E(z_j)}{\partial z_j} = \frac{\partial E(\phi_u,\dots,\phi_w)}{\partial z_j} = \\ \sum_{i\in L}\frac{\partial E(\phi i)}{\partial \phi i}\cdot\frac{\partial \phi_i}{\partial z_j}\end{aligned} \tag{2.54}$$

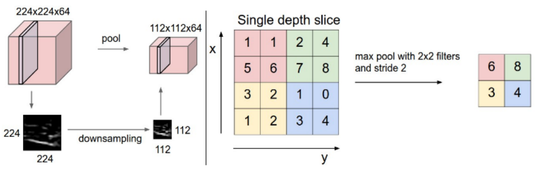
Which yields the aforementioned results of $\sum_{i\in L}\delta_l\omega_{il}$, by noting that $\frac{\partial \phi_i}{\partial z_j}$ yields the weight w_i for that connection since, as already mentioned, the output of a layer is the input of the succeeding layer. Likewise, the term $\frac{\partial E(\phi i)}{\partial \phi i}$ is simply the error factor δ_i for a neuron of the succeeding layer and the summation is merely across all neurons connected to the output of the current on the mesh L. It should noted that the partial derivative above described will evolve as to eventually collapse onto the aforementioned case which yields, simply, $(t-y)$, acting as a "stop condition" for the iterative method (even though this calculation is never explicitly made, as in actual implementations of backpropagation, the error factors δ_i of an immediately previous layer are always stored in memory). Backpropagtion, on its own, has limitations in very deep networks, as the multiple multiplicative factors "dillute" the primary sources of error through the network in the first layers. This forces longer, more extensive training which, in turn, can result in noticeable overfitting.

2.5.12.2 Dropout

In order to mitigate the issue of overfitting and the limitations of backpropagation mentioned above, it's customary to apply dropout during the training of the ConvNet. Dropout entails "shutting off" neurons during backpropagation with some probability, effectively ensuring no weight update is carried out [52, 51] (and, by extension, no learning). This means that, in each iteration, not all weights are simultaneously updated, reducing the threat of overfitting. How aggressive the dropout rate must be depends primarily on the size and nature of the dataset. Dropout is a crucial tool to combat overfitting, and it's required in any learning for typical ConvNet architectures.



(a) Visual representation of the layers for a generic deep ConvNet. Notice the ReLu and Pooling non-linearities between the convolution layers [50]



(b) Matrix representation of the filters for a generic deep ConvNet [50]

2.6 Recent research trends

Even superficial analysis of the proceedings of the most recent editions of the CPVR conference should unequivocally portray the dominance of applications related to ConvNets [53, 54]. Recent publications include the architectures which dominated the Imagenet challenge in the past three years, such as the AlexNet, the GoogLeNet, the ZFnet, the ResNet and the VGGNet. Other recent work relating to ConvNets has been about numerous layer optimizations (refer to 2.5.12) or attempting to achieve competitive performance on a computational budget. That said, the two greatest shortcomings of ConvNet approaches yet remain to be surpassed: their reliance on massive labelled datasets, and their vast computational requirements. In terms of generative applications, these have fallen out of favour ever since ConvNets have monopolized the limelight. However, some recent attempts have been made at increasing performance on scene classification tasks utilizing LDA and GMMs. A brief study on the usage of sparse coding in the vector quantization step of SPM approaches has also shown much promise, not only by improving accuracy, but also by allowing the computational costs associated with this expensive step to scale linearly, instead of exponentially. Fisher Vector approaches have seen some recent applications (such as [18] and [19]), but have predominantly been relegated towards assisting the train of ConvNet by applying feature reduction between layers [46]. Recent works have attempted to improve the performance of ConvNets by utilizing semi-supervised approaches, but no conclusive architecture has been developed as of the time of writing. Other works have argued against the usage of generative models as a form of feature reductions, claiming it results in loss of valuable information (as seen in [17]). These contradictory results can be interpreted as hints towards the fact that the appropriate methodology largely depends upon the context of the problem to which it's applied.

2.7 Computer Vision and Machine Learning toolboxes

This section presents the main computer vision and toolboxes used within the implementation of the project. These are:

- OpenCV 2.4, a computer vision toolbox compatible with C++ and Python with many state-of-the-art implementations of multiple algorithms, including functions for filtering, edge-detection, feature extraction and description and some classifiers, including SVMs [55];

- Caffe, a machine learning toolbox primarily used for ConvNets, including a number of pre-trained models in a "Model Zoo" package, utilized with C++ and Python [56];
- Theano, a Python machine learning toolbox which can be used in conjunction with Caffe pre-trained models, providing numerous classes to abstract lower-level implementation details [57];
- Pylearn2 and Scikit, two machine learning and computer vision toolboxes required due to Theano dependencies [58, 59];
- VLFeat, a machine learning toolbox written in C with implementations of numerous generative models like LDA, FV, GMMs and sparse coding [60];
- The matrix calculation and operation software Matlab, with all machine learning packages [61].

2.8 Additional mathematical concepts

This last section illustrates some simple, yet useful mathematical concepts applied throughout the project.

2.8.1 Distance metrics

Distance metrics are a number of proximity measures which gauge similarity (or dissimilarity) between two elements in a dataset. Letting d represent the distance and x_i, y_i elements of vectors X and Y, respectively, we have:

1. Euclidean distance:

$$d_e(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.55}$$

2. Weighted euclidean distance:

$$d_{we}(X,Y) = \sqrt{\sum_{i=1}^{n} w_i(x_i - y_i)^2} \tag{2.56}$$

3. Manhattan distance:

$$d_M(X,Y) = \sum_{i=1}^{n}(x_i - y_i)^2 \tag{2.57}$$

4. Chebyshev distance:

$$d_\infty(X,Y) = \max |x_i - y_i| \tag{2.58}$$

5. Mahalanobis distance (for a within class variance matrix B, which approximates the covariance of that same class):

$$d_m(X,Y) = \sqrt{(X-Y)^T B(X-Y)} \tag{2.59}$$

(It should be noted that the Mahalanobis distance can be reduced to a special case of the weighted Euclidean distance if the matrix B is diagonal, normalized by a scale factor pertaining to the class. This is especially useful in applications involving PCA.

6. Pearson's correlation coefficient:

$$s_{Pearson's}(X,Y) = \frac{X_d^T Y_d}{\|X_d\| \cdot \|Y_d\|} \tag{2.60}$$

Where $X_d = (x_1 - \mu_X, ..., x_n - \mu_X)$ (and similarly for Y_d). Pearson's correlation coefficient captures information about the orientation of the two vectors centred around their means (that is, a normalized measure of dissimilarity between the orientation of both vectors).

Appropriate choice of a distance metric can heavily influence analysis and judgement of the output of various learning and clustering algorithms.

2.8.2 Accuracy measures and performance metrics

Generically speaking, accuracy is a measure of the degree of closeness between a set of measurements or experimental values and their true value. In machine learning and classification tasks, there's a number of possible metrics to measure the performance of a system. Considering the self-explanatory True-Positive (TP), True-Negative (TN), False-Positive (FP) and False-Negative (FN) rates, we can define:

- Sensitivity: $TPR = \frac{TP}{TP+FN}$
- Specificity: $TNR = \frac{TN}{FP+TN}$
- Precision: $PPV = \frac{TP}{TP+FP}$
- Negative Prediction Value: $NPV = \frac{TN}{TN+FN}$
- Miss rate: $FNR = \frac{FN}{TP+FN}$
- Accuracy: $ACC = \frac{TP+TN}{TP+TN+FP+FN}$

In addition, a convenient way to condense this information is in the form of a confusion matrix. A confusion matrix (or error table) is a matrix CM which contains in its entries $cm_{(i,j)}$ the fraction of elements of class i classified as belonging to class j. Evidently, in entries $cm_{(i,j)}; i = j$ we have the rate of correctly labelled data points for that class. Furthermore, it's also obvious that $\sum_{i \neq j} c_{(i,j)}$ yields the rate of incorrectly labelled examples for the same class. A confusion matrix is a convenient and summary method to present information about the misclassification rates of all points for all classes.

2.8.2.1 Typical cost functions

Cost or risk functions are measures of the deviation between actual measurements and their real value. They typically represent the goal functions of most learning algorithms. A few of the more common ones include:

- **Mean Squared Error**: $MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$, one of the easiest cost functions to calculate, frequently used in regression, representing also an unbiased estimator which minimizes variance;
- **Root-mean-squared Deviation**: $RMSE = \sqrt{MSE(\hat{\Theta})}$, also unbiased and an extension of the MSE, minimizing the standard deviation;
- **Mean Squared Weighted Deviation**: $MSWD = \frac{1}{n-1}\sum_{i=1}^{n}\frac{(\hat{Y}_i - \bar{Y})^2}{\sigma_{Y_i}^2}$, derived from χ^2 distribution, which attempts to account for both internal and external reproducibility pertaining to the model;

2.8.3 Singular Value Decomposition

Singular value decomposition (SVD) is the factorization of a real or complex matrix which originated as an extension of the eigendecomposition of a positive semidefinite normal matrix. Letting M be a $n \times m$ matrix, SVD yields a decomposition of the form $U\Sigma V^*$ where:

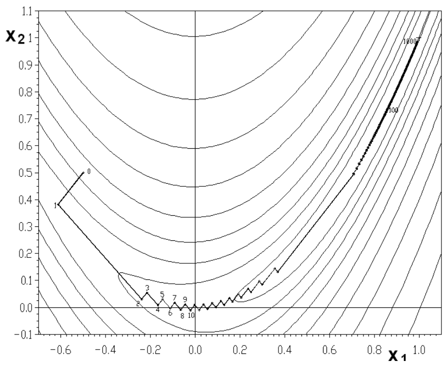
- U is a $m \times m$ unitary matrix;
- V is a $n \times n$ unitary matrix;
- Σ is a $m \times n$ rectangular diagonal matrix;

The entries of Σ are the eponymous singular values of M, which are related to its eigenvectors through $s_i = \lambda_i^2$. Furthermore, the left-singular vectors of M are a set of orthonormal eigenvectors of MM^* and the right-singular ones a set of orthonormal eigenvectors of M^*M.

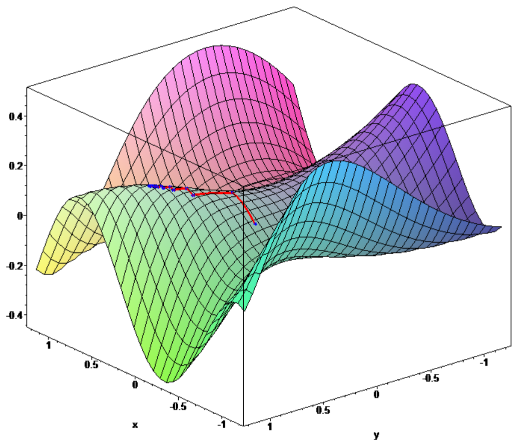
Computationally, SVD has a number of uses: speeding up computation of pseudo-inverses, solving systems of homogeneous linear equations, solving least-square optimization problems and in separable and mixture models, to name a few.

2.8.4 Gradient Descent

Gradient descent is a first order optimization algorithm used to find minima and maxima of a function (typically a goal function) by using the gradient to estimate the direction of maximum variation in each step. Assuming the multivariate function $F(X)$ is defined and differentiable, in the neighbourhood of point x_a, $F(X)$ decreases the fastest if one travels in the direction of the



(a) Visualization of the gradient descent algorithm in a level curve representation of the cost function. With each iteration, the algorithm "zig-zags" towards the minima of the goal function. Note how with each step the variation becomes progressively smaller



(b) 3D visualization of the SGD algorithm over the surface of some goal function. This particular representation is of gradient ascent, the formulation that works towards maxima

gradient, $-\nabla F(x_a)$. It follows that if $x_b = x_a - \eta \nabla F(x_a)$, for some learning rate η, then $F(x_b) \geq F(x_a)$. Applying this calculation iteratively, the algorithm should successfully approximate the minima x_m, until the difference between steps is smaller than some desired resolution (which can be seen in both figures 2.14a and 2.14b). For some properties of F (namely convexity) and particular choices for the rate η, convergence can be guaranteed [62]. In practice, to speed up gradient descent, approximations of the gradient are used. Gradient descent is almost omnipresent in machine learning, being used to approximate solutions to the various goal functions and being applied to learning tasks due to its simplicity and the small computational requirements of most approximated solutions.

2.8.4.1 Stochastic gradient descent and gradient descent with moments

Stochastic gradient descent (SGD) is an approximation to the original gradient descent formulation that uses stochastic methods to estimate the direction of the gradient [63]. It provides a sufficient approximation that can be computed much faster and is overall much lighter computationally. In order to diminish the chance of converging to a local minima, it's often customary moment term (in analogy to the moment in Newtonian mechanics) to "resist" quick variations in direction. This sort of method is often paired with many forms of gradient descent, including the aforementioned backpropagation algorithm in deep neural networks.

2.8.4.2 Nesterov's stochastic gradient descent

Nesterov's stochastic gradient descent is a variation of stochastic gradient descent using moments. In its original formulation, it requires evaluation at points other than those held in the current model values. However, it's been shown by Nicolas Boulanger [64] that it can be reformulated as:

$$
\begin{aligned}
v_{t+1} &= \mu v_t + \eta \nabla(\Theta_t + \mu v_t) \\
\Theta_{t+1} &= \Theta_t + v_{t+1}
\end{aligned}
\tag{2.61}
$$

Where μ, η are the momentum and learning rate parameters and v, f, Θ represent the update, the gradient and the model, respectively. This formulaton is completely analogous to SGD with classical momentum with the sole addition of a perturbation parameter on the gradient calculation. This approach was tested against other SGD implementations such as AdaDelta and AdaWing, marginally outperforming both.

Chapter 3

Methodology and experimental design

3.1 Introduction and project overview

The task of image classification through the multiple variations of SPM used- extensions of the BoW model- can often be broken down into a few key steps, these being:

1. Image preprocessing;
2. Feature extraction and description;
3. Intermediary feature processing (creation of vocabulary representations, feature reduction, feature encoding);
4. Classifier training and testing;
5. Cross-validation and acquisition of a final model ensemble;

This chapter is divided in sections which delve into further detail about these various steps, relating them to the overall design of the project. Summarily, the main topics covered in this section are:

1. Initially, the nature and properties of the datasets utilized in the different trials are highlighted. An initial discussion of methods used to manipulate and pre-process this data is also presented;
2. Secondly, a section describing the architecture of the pre-trained networks is presented. This includes the method to adapt this model to the training and test datasets, as well as multiple methods to manipulate the data in such a way that it does not harm the performance of the ConvNet, but increase the amount of data available for training;
3. The first of the remaining three sections- all pertaining to the overall SPM ensembles- explores low-level image descriptors used to extract features for the intermediary generative methods, and how these were combined to generate the final feature descriptor;

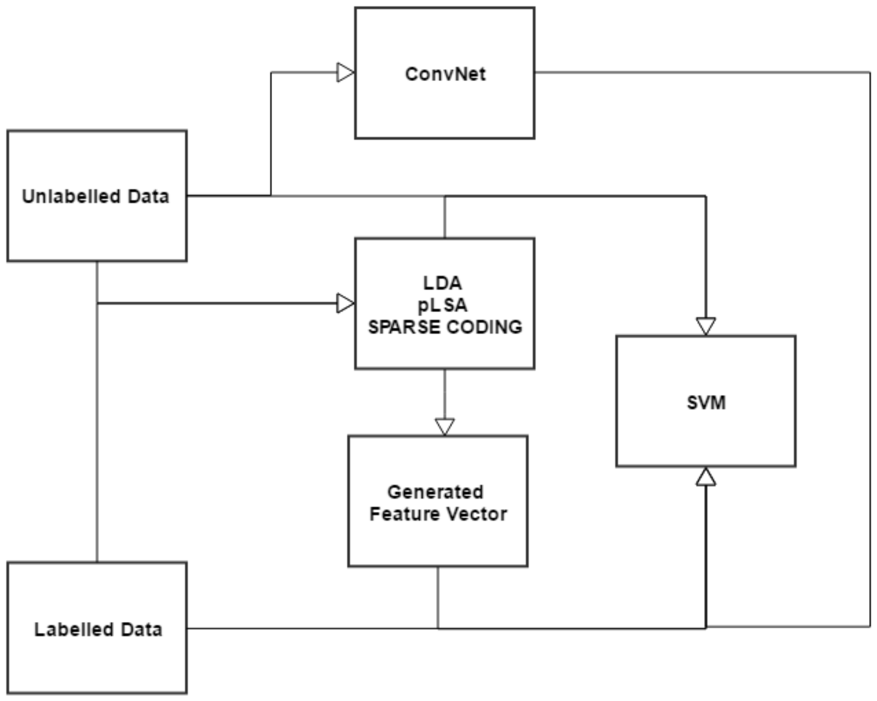


Figure 3.1: Schematic representing the overall ensemble for image classification

4. The fourth section explores generative and statistical methods built on top of or in place of K-NN clustering vector quantization: LDA, pLSA and sparse coding. It also explores the usage of Fisher Vectors;

5. The fifth section, pertaining to the classification step, explores how the typical soft-margin SVM formulation can be adapted to make use of the ConvNet class scores as an estimation for the cost penalty during non-linearly separable classification. It also delves onto the novel idea of "generated topic vectors" for the pLSA model;

6. The last section has a brief hardware overview, exploring what computational resources were available for the different tests;

Figure 3.1 presents a visualization of the overall architecture of the proposed solutions, highlighting the usage of both generative methods as a form of intermediary feature reduction, the usage of the high-accuracy ConvNets as a form to guide training using unlabelled examples and the usage of fictitious, generated topic vectors.

3.1.1 Similar studies and previous work

Previous work has been made in an attempt to utilize generative methods in image classification tasks. Firstly, a follow-up study to the original SPM study utilized sparse coding as a method of feature reduction [65]. The results of this study showed that sparse coding can be quite successful as a form to generate a distinct visual vocabulary. A second study utilized LDA purely as an intermediate means of feature reduction for a scene classification task [66]. Both of these, however, only tested the methodology on comparatively small datasets (for instance, the Caltech dataset utilized in only has, at most 120 images per category). These studies did not delve deeply onto the applications of either method in larger datasets, with a wealth of unlabelled data (comparatively, this study aims to apply them to thousands of images per class). Another study on the use of Fisher Vectors yielded encouraging results, showing their versatility and competitive accuracy even for large-scale image classification tasks [18]. These results were taken as a starting point for the application of Fisher Vectors in the context of this study. Overall, application of these generative methods in large-scale datasets remains comparatively uncommon and unexplored.

3.2 Dataset overview and preprocessing

Three primary datasets were used to test and compare the different methods. The first dataset is comprised of 25000 images, labelled as "cats" and "dogs". The data was extracted from a Kaggle competition [67, 68]. This binary dataset was chosen for a variety of reasons. Firstly, the classes are quite irregular, with elements of each class having multiple inconsistent characteristics (which may even overlap, such as similar colours or overall shapes), partly due to the variable backgrounds and positions for each cat or dog (as seen in 3.2). Secondly, these two classes are quite similar to other classes present in the Imagenet dataset, which was used to train the various ConvNet models acquired form Caffe's model zoo. Lastly, the classes are not only irregular but share some similarities between each other. This binary, simpler case served to test and validate the various proposed approaches. This original dataset was then expanded to hold five classes- adding the "whale", "fish" and "galaxy" classes. Images for these were obtained from Kaggle competitions and the ImageNet dataset [69, 70]. The galaxy and whale classes present the solutions with a more regular dataset (especially in terms of the background); however, it should be noted that edge information is quite poor in the galaxy and whale classes, and that colour information is quite similar between the fish and whale classes in some examples (predominant shades of blue in the background, for instance). Furthermore, the galaxy class has no direct analogue in the Imagenet dataset which was used to train the pre-trained ConvNet models [56]. This provides the opportunity to test the performance of these pre-trained models (both on their own and as a tool to assist training using unlabelled data) in tasks which are somewhat different from those for which the deep networks were originally trained for. The 5-category dataset was designed to both test the methods on a wider, more complex dataset, but also to compare these methods with the performance of the pre-trained ConvNet on more of an "even field", with a few classes for which the ConvNet did not have tens of thousands of labelled examples to be trained upon. The last dataset, not used directly, is the Imagenet dataset. This is the dataset on which the ConvNet models were trained. The Imagenet dataset spans over a million images across a thousand classes. It has, furthermore, an hierarchical tree structure for assigning subclasses, meaning the ConvNet is trained to pick up very distinctive, finer features even for irregular classes (for example, in the case of the "cat" class, the Imagenet dataset presents categories for different species of cat, meaning the ConvNet was trained to pick up distinctive features for multiple types and races of cat). It should be noted that despite the relatively large number of labelled images available for each dataset, only one thousand labelled examples were used for each category, at most, to simulate scarcity of this type of data. Both tests and validation sets were also comprised of these labelled examples, allowing the collection of numerous results, such as precise accuracy measures and confusion matrices for each algorithm. In vocabulary generation and encoding, methods utilized the aforementioned one thousand labelled images in conjunction with one thousand unlabelled images (for all methods permitting the usage of unlabelled data). The SVM training schedule utilized one thousand training examples, initially all labelled and then progressively replacing labelled data with unlabelled data.

3.2.1 Preprocessing

Images in all datasets were, if necessary, resized down to a 400×400 pixel area, to reduce computational costs. Initially, mean-colour corrections were used, but these proved to have either minimal or slightly adverse effects when local colour histogram descriptors were added as features. Due to the nature of the images from the whale and galaxy categories- the former populated by very large, high-resolution images, and the latter dominated by dark areas without information of interest- it became important to edit these images beyond simply resizing them. The main idea was to locate the regions of interest and cut down the excessive blue and black portions of each picture (in the case of the whale and galaxy categories, respectively). As seen in figure 3.3, edge information for these two specific categories is quite poor. The initial approach at edge detection followed by Hough transform proved ineffective in locating regions of interest in each picture. Instead, alluding to the colour properties of the pictures, each image was split into its three HSV channels. Sweeps over the Hue and Saturation channel were carried out- first, median and average Hue and Saturation over a small sample of ten images were computed. These were used to define a "typical background pixel", possible due to the regularity of the images in the dataset. Then, iteratively, each pixel that deviated from these values was considered as being potentially part of an "interest patch". Its neighbouring pixels were sampled and, if they were sufficiently similar (and, by extension, also dissimilar to the typical background pixels), were added to a congruent region. After iterating over the entire image, an approximate centre of the largest anomalous region was chosen as the centre of the region of interest. A 400×400 pixel area around it was extracted as the trimmed, final image. This was crucial due to computational costs, but also so features of interest weren't located in a small image region, thus making a large portion of the image essentially noise, beyond potentially useful colour information contained in the aforementioned predictable background.

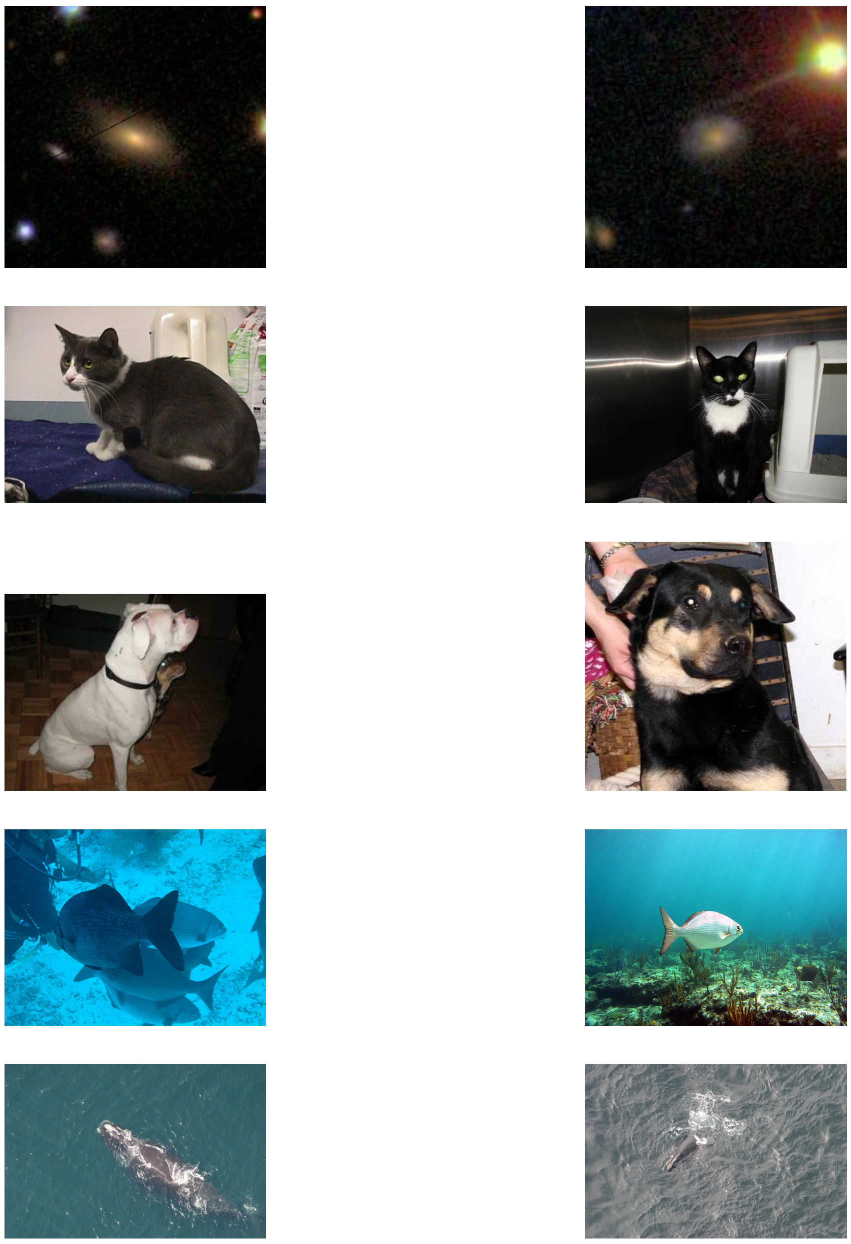


Figure 3.2: Example images of the 5-category dataset. From top to bottom, a pair of images with the galaxy, cat, dog, fish and whale labels are presented, respectively

3.3 Deep Convolution Networks

The pre-trained ConvNet was trained on the full Imagenet dataset, and follows a slightly modified AlexNet architecture, similar to the one proposed for the ZFNet ConvNet [56]. Seeing as the pre-trained model was designed for the Imagenet dataset, the last dense layer reflects this by having 1000 terminal neurons which generate the class labels for the 1000 categories on the original Imagenet challenge. In order to adapt the ConvNet to either of the experimental datasets, this last layer had to be rectified. This was done by "cutting off" the last 1000 units comprising the final

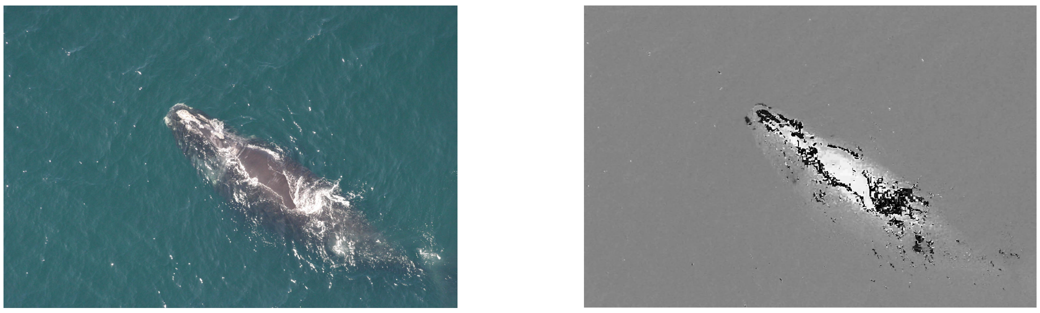


Figure 3.3: Image belonging to the whale category. The hue channel can be visualized on the right

layer and replacing them with an appropriate number of neurons for each dataset (in this case, two and five, respectively). This obviously requires an additional training step to prepare these new units with appropriate weights for the classification task. This training step was implemented utilizing the Theano, Scikit and Pylearn2 toolboxes, over Nvidia CUDA and anaconda packages to allow GPU computation [71, 72]. The hardware utilized for this specific portion of testing was comprised of two GTX970 Nvidia video cards, an i7 quadcore Intel processor (up to 2.3 *GHz* per core) and 16 GB of RAM memory. It should be underlined how indispensable GPU computation is to carry any training in a practical length of time. These results will be further highlighted in chapter 4. Due to their need for vast labelled datasets, techniques to artificially expand the available data were used in training the ConvNet. These manipulations and an overview of multiple considerations related to the training schedule, as well as some architecture details, are presented and discussed below.

3.3.0.1 Training data augmentation

In the original ImageNet ILSVRC challenge, Alex Krizhevsky [48, 73] applied a number of transformations to manipulate the original training data and create modified copies with similar properties and features, thus artificially augmenting the original set and increasing its size. A similar scheme was used to train the deep ConvNet, seeing as the amount of training images available was much too limited to allow proper training without overfitting. The images were further downsampled to a more manageable 250×250 pixel size. Afterwards, much like in the original AlexNet training schedule, portions of the initial images were cropped out. Again, mirroring the original methods employed in the first AlexNet, vertically mirrored copies of the image were added. Colour perturbations which involve altering the intensities of the RBG channel values were also employed. These consist of running PCA over the three colour channels and adding multiples of the principal components on each pixel (thus, on average, reinforcing the pixel values proportionately to their relevance in a feature interpretation). The operation can be seen as adding to each pixel $I_{x,y} = [I^R_{x,y}, I^R_{x,y}, I^R_{x,y}]$ the following quantity:

$$[p_1, p_2, p_3] = [\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T \tag{3.1}$$

Where p_i, λ_i are, respectively, the ith eigenvector and eigenvalue and α_i a parameter that was set to follow a normal distribution, $N(0,0.1)$, mirroring what was done in the original research. Lastly, shearing was applied to the training images, on the input layer of the ConvNet. This operation, which deforms the overall shape of the images, was used on only very slight degrees (coefficient $m \leq 0.2$), as to not overly distort discriminant properties of the image. The overall process is partially exemplified in the scheme in figure 3.4 for an image of the "galaxy" category, with the exception of the colour perturbation and shearing steps, for ease of interpretation (although a colour-perturbed, sheared image would undergo this process exactly). Note the creation of various images through affine operations.

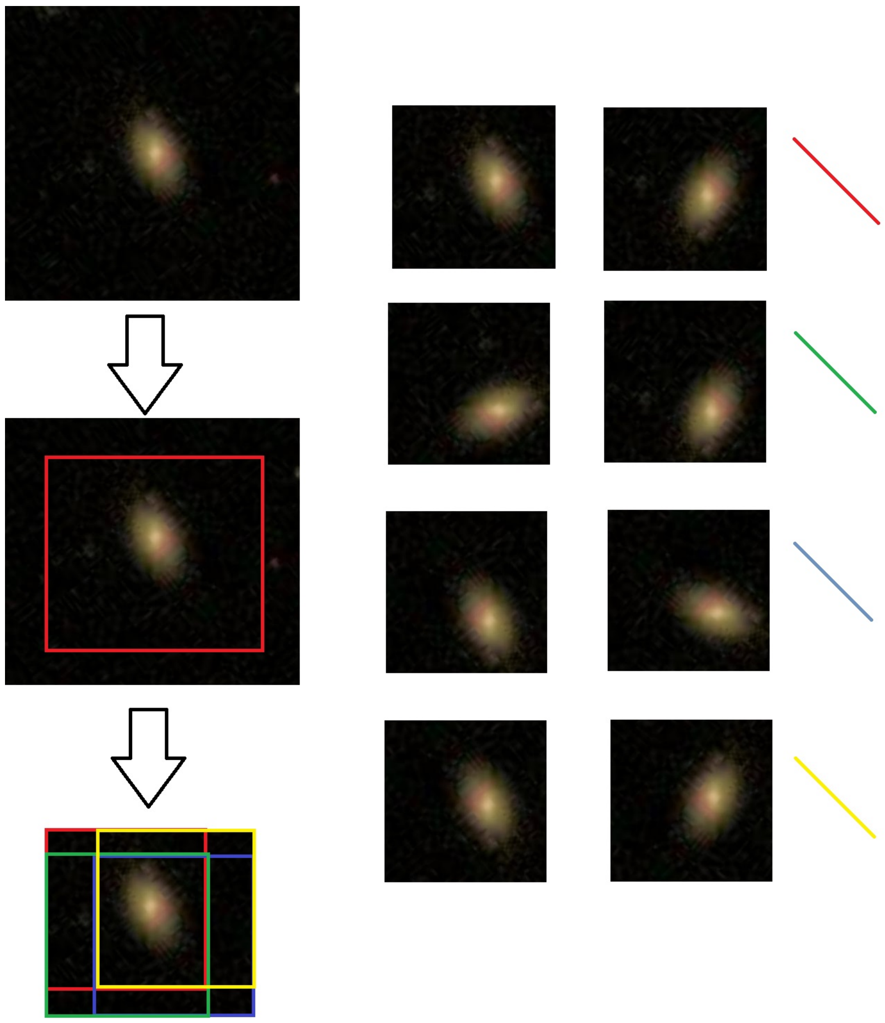


Figure 3.4: Illustration of the preprocessing steps used to create multiple artificial training images. Note that this rendition was done by hand and is less accurate than the actual process

Training images were loaded onto the RAM memory in batches- the original image accompanied by seven altered versions, obtained through the aforementioned process, in order to speed up the training.

3.3.1 Backpropagation and stochastic gradient descent: Nesterov's Accelerated Gradient

In order to train the new dense layer, backpropagation based on gradient descent methods was necessary. The toolboxes used for this process (Theano and Pylearn2) have multiple gradient descent algorithms available. Choosing an appropriate algorithm becomes necessary. It should be noted that iterative stochastic gradient descent typically performs as well as batch algorithms, whilst being far lighter computationally [74, 63]. As such, no such batch algorithms were considered for any layer. Of the available options, as seen, AdaDelta and SGD algorithms with momentum present the fastest and most reliable solutions. The final choice ultimately weighted whether to value the robustness of AdaDelta over the faster convergence of Nesterov's Accelerated Gradient or not [74, 75, 76, 64, 44, 77, 64]. Fortunately, earlier works utilizing deep ConvNets with similar architectures ([78, 49]) provided a good initial seedings for the choice of the momentum parameter. Weight-normalization in the dense layers had to be relaxed in the initial iterations to allow convergence (otherwise, the model could potentially diverge). As stated in section 2, Nesterov's stochastic gradient descent would typically require evaluation at points other than those held in current model value but can, however, be reformulated as:

$$\begin{aligned} v_{t+1} &= \mu v_t + \eta \nabla(\Theta_t + \mu v_t) \\ \Theta_{t+1} &= \Theta_t + v_{t+1} \end{aligned} \tag{3.2}$$

Where μ, η are the momentum and learning rate parameters and v, f, Θ represent the update, the gradient and the model, respectively. This formulation is completely analogous to SGD with classical momentum with the sole addition of a perturbation parameter on the gradient calculation. This approach was tested against the Theano implementation of AdaDelta, marginally outperforming it in the initial 150 training iterations (by about 2%). A fixed learning schedule which decays over the 500 iterations was implemented. The initial value was empirically picked as to allow the SGD method to converge (that is, to effectively improve the model). Due to time limitations, testing on SGD variants and learning rate was extremely limited. Further improvements can likely be drawn by further experimenting with the SGD form and its related hyperparameters, as well as further optimizing the learning schedule.

3.3.2 Overfitting and parameter sharing

Overfitting is a primary concern when training large networks, which is the case for the final model, with over 60 million parameters [6, 48, 52, 51, 79, 44]. In the data augmentation section, methods to artificially increase the training dataset were explored with some detail. However, even with this extension of the dataset accounted for, the overall amount of data is far too small to conduct a full reconditioning of the ConvNet for the new datasets, even with the pre-trained weights as a starting seeding (which, effectively, allows for much faster convergence). To overcome this issue, aggressive dropout rates of 0.9 and 0.99 were applied to the first and second dense layers, thus limiting the overall effect of this smaller dataset during the training of the pre-trained dense layers and hopefully mitigating overfitting (dropout entails "turning off" learning for a given neuron on any iteration with a given rate, in this case, of 90% and 99%). This is absolutely crucial, as the dense layers hold over half the overall parameters of the network, and seem to be the most susceptible to overfitting. The convolutional layers were not trained upon. This is essentially equivalent to the assumption that features extracted in the context of the original problem for which the ConvNet was trained are also useful for the datasets in this study. It should be underlined that if this assumption is shown false, the limited available data for the training schedule in the comparative study and the design of the final ensembles would not be enough to fully train the entire network. This means that, in the event that the two applications are not similar enough for the ConvNet to have acceptable performance, full training of the convolutional and dense layers is not be possible with the proposed artificial limitations imposed on the training schedule. Even if it were, this training procedure is onerous and requires high-end GPUs to be carried in any practical form.

3.4 Feature descriptors and feature extraction

As mentioned, SPM is an extension of the BoW model for image classification. As such, initially, image descriptors for salient and invariant keypoints for each image are obtained. While not a focal part of this study, brief experiments were carried out to find a well-performing and easy to manipulate set of features obtained from these descriptors. It's been shown [66, 34, 80] that the SIFT-type descriptors perform particularly well in most classification tasks. Therefore, most of the tests utilized SIFT descriptors (either the normal SIFT formulation or the PCA-SIFT variant), although, for some initial trials, a similar type of descriptor, SURF, was also tested. The SURF features, however, resulted in comparatively worse results and were ultimately dropped in favour of SIFT implementations. Both of these were combined with colour information encoded through histogram quantification of image patches over which the SIFT descriptor was applied, effectively creating a BoC representation of the image. Generically speaking, for a trial, two distinct feature vectors were created, a D-dimensional vectors from the visual vocabulary and a K-dimensional colour vectors, which were combined into a final $(D+K)$-dimensional feature vectors, as seen in figure 3.5. The PCA applied to the SIFT descriptors kept 64 dimensions (of the original 128) which expressed over 99% of the variance from the initial feature vectors. Implementation of this step utilized the OpenCV 2.4 toolbox, the VLFeat library, Matlab and the PCA-SIFT implementation found in [81]. Descriptor matching made use of the FLANN nearest neighbour search library, embedded in the OpenCV toolbox. The code was mostly written in the C++ programming language which was called externally by the smaller, top-level Matlab program.

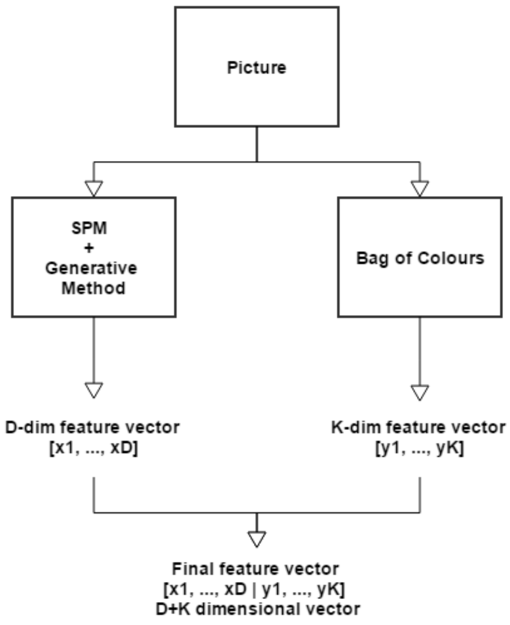

Picture
SPM + Generative Method
Bag of Colours
D-dim feature vector [x1, ..., xD]
K-dim feature vector [y1, ..., yK]
Final feature vector [x1, ..., xD | y1, ..., yK] D+K dimensional vector

Figure 3.5: Schematic illustrating how both types of features are combined into a single $k+D$-dimensional vector

3.5 Creation of the visual vocabulary for classification and feature reduction through semi-supervised generative methods

After obtaining a vector of feature descriptors, generation of the vocabulary codebook is necessary. As stated, a SPM scheme was utilized. In practical terms, relating to the implementation, a visual vocabulary is initially created and utilized to generate a BoW model for different sections of the image (the "pyramid" levels), which are then brought together into a single, final feature vector through some kernel (in its original formulation, the pyramid matching kernel described in 2.4.2). The number of pyramid levels, L, is a hyperparameter that influences the performance regardless of the visual vocabulary is generated, and is therefore present throughout the experimental design for all methods. Sadly, due to time and computational constraints, this parameter could not be thoroughly tested. Instead, after obtaining experimentally optimal values for other parameters for some number L, these were kept and pyramid levels of $L=(2,3)$ were tested. It should be noted that the original authors reported no gain in levels beyond the third [17, 82] and due to the aforementioned limitations, no higher values were tested upon. It should be noted that none of the methods for visual vocabulary generation and reduction influenced the colour features. So if a method generates a K-dimensional feature vectors for a section of the image at pyramid level l, the actual number of features is $K+D$, with D being the features pertaining to this colour information.

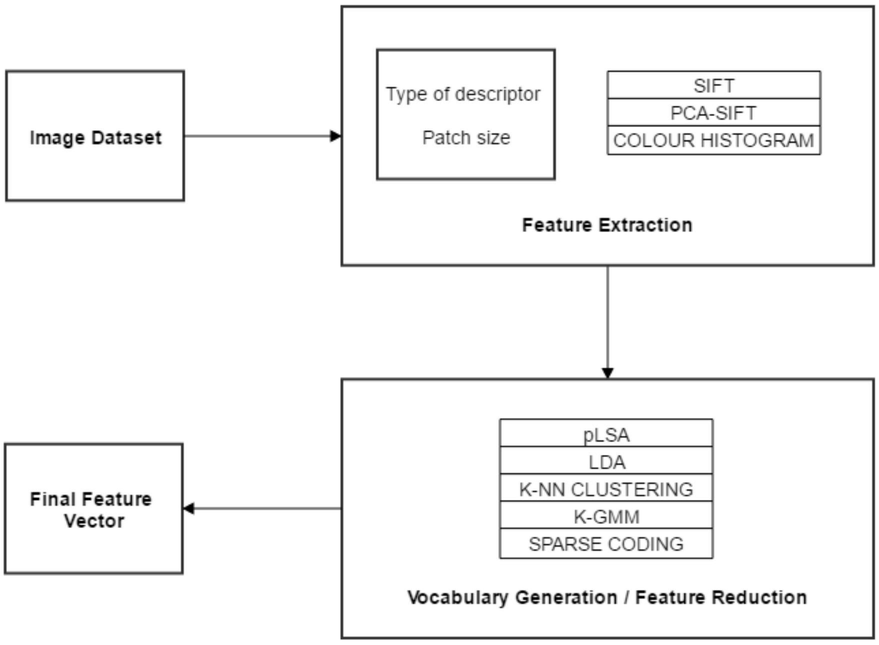


Figure 3.6: Schematic representing the semi-supervised process through which the final feature vectors are generated

3.5.1 Original SPM implementation with vector quantization

The original SPM design was implemented, which utilizes K-NN clustering to group the features obtained by the SIFT descriptors into K clusters, the visual words, thus forming a K-dimensional feature vector for each pyramid patch, obtaining a final histogram representation of the original features obtained through the SIFT descriptor. This is a form of Vector Quantization (VQ). Mathematically, VQ for a histogram representation can be viewed as:

$$z = \frac{1}{K} \sum_{k=1}^{K} u_k \tag{3.3}$$

With u_k a descriptor matched to the kth cluster. The main hyperparameter to consider, beyond the aforementioned L parameter, is the number of clusters K, which represents the vocabulary size. The total dimensionality of the feature data is, therefore, $\sum_{l=0}^{L} 2^{2l} K$, for all pyramid levels. Through the pyramid matching kernel, this matrix is reduced to a single K-element vector. The vocabulary obtained through SPM with VQ tends to achieve better classification results when paired to SVMs which utilize χ or Gaussian kernels [65], so this non-linear kernel was used for this formulation (a disadvantage, as these non-linear kernels are typically slower during training). Implementation of the original SPM model was also done in C++ using the OpenCV and VLfeat toolboxes. SIFT descriptors were obtained through either the SIFT implementation in the VLfeat toolbox or the previously mentioned PCA-SIFT implementation. The results of the descriptors were exported to text files, and then used with conjunction with OpenCV to obtain vocabulary through K-NN clustering. Matching was also done in C++ using OpenCV (using the FLANN library, as mentioned in the previous section).

3.5.2 SPM with sparse coding

Following the previous model, one can now modify the process through which the visual vocabulary is generated. The K-NN clustering algorithm, as seen, tries to solve the problem:

$$\min_{V} \sum_{m=1}^{M} min_{k=1,\ldots,K} \|x_m - v_k\| \tag{3.4}$$

Where V represents the codebook of K clusters (v_k representing the kth cluster centre). This goal function can be rewritten as a matrix factorization problem [65], with the cluster-membership now represented by a matrix U; that is, a matrix which illustrates to which cluster is each point assigned. This problem imposes a cardinality constraint $Card(u_m) = 1$; that is, the descriptor represented by line u_m can only be matched to one cluster (which was previously achieved in the OpenCV implementation by using the FLANN library to match each descriptor on a test image to one of the clusters in the visual vocabulary). Sparse coding relaxes this cardinality constraint by proposing an alternative method for assigning collected keypoints to visual words- instead of matching a point to a single cluster, we can assign scores for the different visual words, forcing a normalization constraint of $\|u_m\| = 1$. With these new constraints, the sparse coding goal function (as seen in 2.29) can be derived from the original VQ goal function:

$$\min_{U,V} \sum_{m=1}^{M} \|x_m - u_m V\|^2 + \lambda \|u_m\| \tag{3.5}$$

With codebook V representing an overcomplete basis set (that is, the number of visual words typically exceeds the number of features to be mapped to those visual words). In terms of implementation, solving this new goal function involves a training and a coding phase, similarly to what's done with VQ. Initially, a descriptor set X from a random collection of image patches is used to solve equation 3.5 with respect to U and V. Afterwards, in the coding phase, for some (or all) images represented in the descriptor set X, the SC codes are obtained optimization of 3.5 with respect to U only. The first step (the training step) supports semi-supervised training by including patches from unlabelled images, which enriches the overall code. The coding scheme by SC features a much lower reconstruction error than VQ [65]. The sparsity of the vectors makes it so only the most salient features are stored (making file I/O operations much faster). This can be seen by analysing the matrix U (the output feature matrix, where each of the initial, input descriptors has been remapped through application of the SC codebook V, their individual responses to this matrix represented in column u_j). Following the original study in [65], the remapped feature matrix U was subjected to max-pooling, that is:

$$z_j = \max |u_{1,j}|, ..., |u_{M,j}| \tag{3.6}$$

With M representing the number of local descriptors. This means that, for each feature descriptor, only the strongest response to the SC matrix V is kept. This is analogous to considering V a neural network, and only the strongest neural response is considered in the output (as explored in 2.5.8). The matrix V stores the SC model to be used in the feature encoding and reduction step. This scheme for vocabulary generation was then applied to the previously explored SPM formulation. The main parameter to study is λ, which controls the sparsity obtained in the SC representation. Sweeps over this parameter were carried out to determine the influence of this parameter. Implementation of this methodology was achieved by adapting code provided publicly by Jianchao Yang ([83]), written in Matlab with calls to external implementations of the conjugate gradient descent method, written in C.

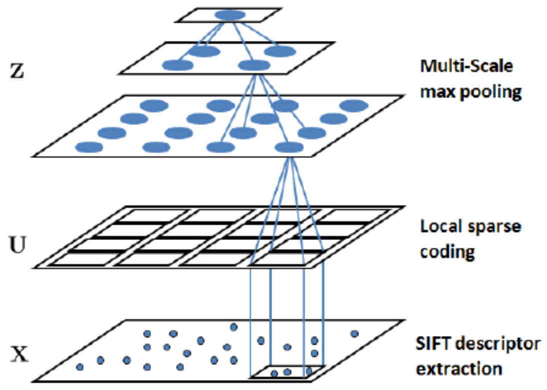


Figure 3.7: Schematic representing the SC-max pooling ensemble

3.5.3 Topical representations through LDA and pLSA

Both the LDA and the pLSA methods were originally designed for text categorization. In applying them with conjunction with BoW models, the extension of an *image* as *document* and an *object category* as a *topic* is made. As underlined in 2.5.7, pLSA is a particular case of LDA. In terms of theoretical considerations and implementation details, the only distinction is that LDA requires maximization over two additional parameters, (α, β). This subsection will address both methods generically, highlighting any difference between either algorithm only when necessary, seeing as the bulk of the considerations are applicable to both methods. The pLSA/LDA model is built on top of the VQ step, as described in 3.5.1. Assuming we have a collection $D = (d_1, ..., d_N)$ of N images and a visual vocabulary $W = (w_1, ..., w_K)$ of K words, the model assigns a latent variable

$z \in Z = (z_i, ..., z_Z)$ to an observation $n(w_i, d_j)$ of the $N \times K$ occurrence table. In practical terms, the model goes through the following steps:

- Select image d_j with probability $P(d_j)$;
- For that document, pick topic z_k with probability $P(z_k|d_j)$;
- Generate word w_i with probability $P(w_i|z_k)$;

Taking the equation shown in 2.28, this can be marginalized over the latent variable z, yielding:

$$P(w,d) = P(d) \sum_{z \in Z} P(w|z)P(z|d) \tag{3.7}$$

Noting that $P(w,d) = P(d)P(w|d)$, we can extract from the previous equation the form of $P(w|d)$:

$$P(w|d) = \sum_{z \in Z} P(w|z)P(z|d) \tag{3.8}$$

This problem can reformulated as being equivalent to a matrix decomposition, as evidenced on the plate diagram in figure 3.8, with the normalization constraint over $P(z|d)$ and $P(w|z)$ to make them probability distributions. Determining $P(z|d)$ and $P(w|z)$ requires maximization of the log-likelihood, as seen in 2.25. This is achieved through usage of the EM-algorithm (2.5.3). Resolution of this problem for the LDA formulation requires maximization over the additional parameter α of the Dirichlet distribution, also through the EM-algorithm. This requires additional computation time and resources compared to the simpler pLSA model (which assumes a uniform Dirichlet prior). Training of this model involves two steps. First, the distributions $P(z|d_{train})$ and $P(w|z)$ are learned from the available data- both labelled and unlabelled. There are two hyperparameters worth consideration on this step: the number of visual words resulting from the VQ step, K, and the overall number of topics Z in the LDA/pLSA model. As stated, once these parameters are set, the VQ representations of the training set are fed to the model, which finds the relevant probability distributions using the EM-algorithm. After these distributions are learned, which represent the statisticial model, classification also requires usage of the EM-algorithm to compute the coefficients $P(z_k|d_{test})$, updating these iteratively in the M-step, but skipping the E-step (utilizing, instead, the parameters from the model each iteration). As stated, sweeps over the parameters K and Z were carried out to study the variation of the performance of the model with the number of visual words and the number of topics. Implementation of these methods were done primarily in C++ ([84]), utilizing the VLFeat toolbox ([60]), exporting the feature vectors to a text file to be used in Matlab.

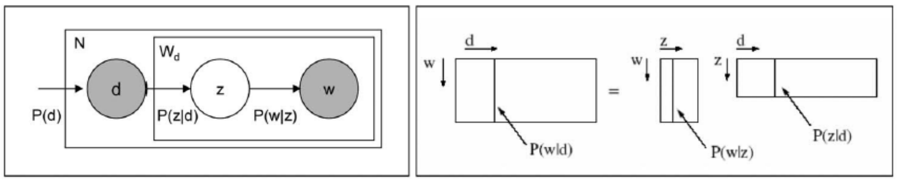


Figure 3.8: Plate diagram for visualization of the pLSA problem described as a matrix decomposition operation

3.5.3.1 Generated topic vectors

The LDA/pLSA models are generated utilizing labelled images from both categories and unlabelled images without distinction. The interpretation of the labelled corpus of either category is tied to the overall dataset- that is, the topical vectors generated for an image of a certain category depends upon the corpus of images utilized when creating the LDA/pLSA model (and thus, its topic representation varies with the corpus of other categories and the unlabelled corpus used to create the model in the first place). Once the model is obtained, one can now generate a topic representation for each image belonging to each category. These topic vectors can be used to generate an overall frequency representation for topics belonging to each category. If this is done for each class, it can subsequently be normalized to obtained a new, category-specific multinomial model that represents each individual probability $p(z_i|C_j)$, with z_i the ith topic and C_j the jth category. In order to keep some form of sparsity-constraint, mirroring what the original LDA/pLSA formulation aims to achieve (it's desirable that topics are associated with a small portion of the most salient words, as to represent the variance between different classes correctly), the median number of topics represented in each category, $\hat{z}_j$,is also computed (since the presence of each topic is binary, the median euclidean norm of the topic feature vectors can be used on this computation). After computing the new multinomial model, uniformly drawn random numbers are used to create new, generated topic vectors. A topic is added to this new, fictitious vector if the random number, r, is such that $r \geq p(z_i|C_j)$. If the number of topics for this vector is greater than $1.5\hat{z}_j$, topics with smaller probabilities (smaller $p(z_i|C_j)$ factors) are set to zero until the sparsity constraint is verified. While this method has the advantage of promoting more salient topics, it also has the adverse of being much more prone to overfitting, so the number of generated topic vectors has to be kept conservatively small. The factor of 1.5 utilized in the sparsity constraint is purely empirical and was set after some brief experimentation with other factors in the 1~2 range.

3.5.4 GMMs and Fisher Vectors

Fisher Vectors add a third method for generating a visual vocabulary, which fit a GMM to the data points obtained by the feature descriptors. The GMM captures statistical information about the distribution of the keypoints, which presents a richer formulation over which a more distinct visual vocabulary is built. Figure 3.9 illustrates how the Gaussian parameters (μ, σ) influence the way keypoints are grouped into visual words through the GMM; the Gaussian mean effectively having an analogous role to the cluster mean in the K-NN VQ method, and the variance being a metric for the "range" of each visual word. The procedure, in terms of implementations, is actually very straightforward- keypoints are extracted through the use of the PCA-SIFT descriptors, upon which a K-GMM model is fit through the usage of the EM-algorithm. This GMM is then essentially fed onto a Fisher Kernel, which is a normalized form of the gradient of the log-likelihood of the fitted GMM (essentially analogous to what was explored in 2.25, but with a normalization parameter). Because the output feature vector from the Fisher Kernel is typically dense (sparsity under 50% in most applications), PCA has to be applied to minimize the number of SIFT components. This has added importance due to assumptions made on the GMM model, namely that the covariance matrix is diagonal. This assumption is important because it both limits overfitting (matrices which aren't diagonal are more prone to overfitting) and reduces computational costs. PCA helps achieving this uncorrelated nature in the keypoint features (and it's downright required for the model to work, as seen in [18]). Furthermore, as seen, this is highly desirable not only for speeding file I/O operations, but also due to optimizations in most SGD algorithms when dealing with sparse vectors (as these are quite common in most practical machine learning applications). The overall dimensionality of the final feature vectors is tied to the number of Gaussians (which should be evident, seeing as the gradient of a GMM will result in a sum of terms with the mean and variance parameters). The parameters are:

1. The K weights, one for each of the K Gaussians in the GMM;
2. The KD mean terms, D for each of the K Gaussians, D being the number of keypoints;
3. Similarly, KD variance terms;

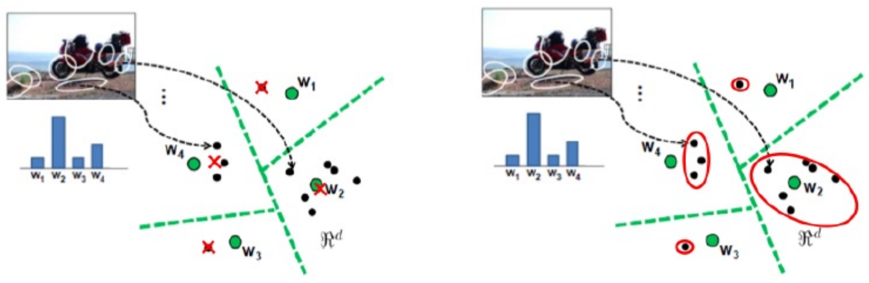


Figure 3.9: Visualization of the relation between the parameters of the GMM and the grouping of different keypoints into visual words

The hyperparameter of interest is, evidently, K, the number of Gaussian functions present on the GMM, to which the number of visual words are proportional and the number of FV components (up to $2K(D+1)$ components, as can be trivially calculated from the aforementioned parameter count). Sweeps over this parameter were performed, to explore the variation of the performance with the number of Gaussians present on the model. Implementation of this method is made trivial through the usage of the VLFeat toolbox, which includes a built-in Fisher Vector and GMM method (simply load the keypoint vectors from an external file and, after loading them onto an appropriate array, only two function calls are necessary to generate the model). VLFeat also enables the use of l^2 normalization, which was utilized due to the increase in performance reported in [18].

3.6 Classifier modelling and training

After appropriate feature vectors were extracted through one of the various schemes described in the previous section for all images of a given dataset, these were fed to a SVM for training and future classification. Recalling the SVM cost function in the soft-margin formulation, as seen in 2.43:

$$\begin{aligned}
&\gamma := \min_{\tilde{\gamma},w} \|w^2\| - C\sum_{i=1}^{n} \varepsilon_i \\
&Subject\ \ to: \\
&\varepsilon_i > 1 \\
&y_i(w^T x_i) \geq 1 - \varepsilon_i
\end{aligned} \tag{3.9}$$

We can modify the scalar term C into a diagonal matrix, where each non-zero entry $c_{(i,i)}$ represents the weight of the ith sample. In practical terms, this can be interpreted as assigning a weight for each data point. Both Scikit and Matlab can use the LibSVM implementation, both of which support class weights and individual sample weights. The SVM training schedule was implemented both in Matlab (for the SPM+SC formulation) and Python (for the remaining methods). With this in mind, we can now implement the novel approach of utilizing pre-trained ConvNets to guide learning in a semi-supervised approach. What was done can be effectively summarized as using the ConvNet class scores as estimators for which category an unlabelled image belongs to. Given their high accuracy, if the classification task is sufficiently similar to the task for which the ConvNet was originally trained, this assumption can be reasonable. The validity of this assumption can be experimentally tested by running the pre-trained ConvNet on the labelled data. If the accuracy score is satisfactory, this approach can be used. "Satisfacory", in this context, is subjective to the application and, in essence, is an empirical quality. In the training scheme, the sample scores were fixed to $0.5 \times \alpha \times c_{CONV,i} \times \lambda$, where α represents the accuracy score of the pre-trained ConvNet on the labelled data and $c_{CONV,i}$ the class score attributed to unlabelled image i by the ConvNet. This weighting reflects both an estimation of the confidence in the ConvNet's tentative performance (through the parameter α) and an estimation of the tentative classification of the image through $c_{CONV,i}$. After class scores were generated for all unlabelled data, each image was assigned a tentative label through $\max(c_{cat,i}, c_{dog,i})$, where $c_{cat,i}, c_{dog,i}$ are the normalized scores for the cat and dog categories, respectively. Values for the average score and maximum score for each class, $c_{max,j}, c_{avg,j}; j = cat, dog$, were also were computed. Three methods for selecting unlabelled images were developed:

1. **Strong scores** scheme, in which the highest scores for each class are selected, if they greater than $c_{avg,i}$. In case this is not possible, copies of the previously selected image are drawn randomly (through a uniform pdf) until the number of required images is satisfied;

2. **Weak scores** scheme, in which unlabelled images are drawn if their score is greater than $c_{avg,i}$ (and therefore not necessarily the highest scoring images for either class in the list). If there aren't enough images scoring over $c_{avg,i}$, images with lower scores are drawn until enough unlabelled samples are obtained;

3. **Absolute scores** scheme, a variation of the strong scores scheme where each image is reassigned a new score, equal to the maximum score (effectively weighting them all equally and maximally);

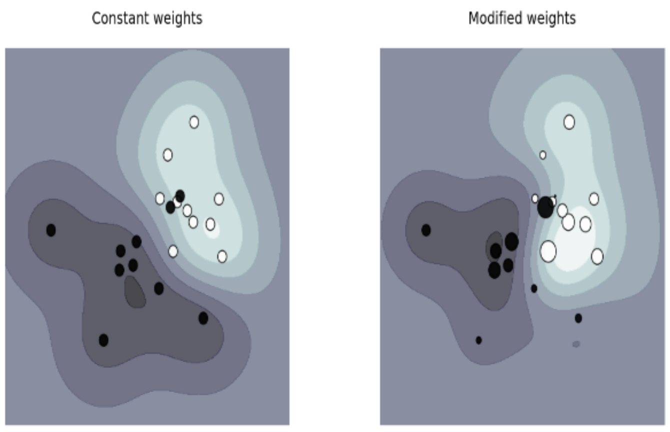


Figure 3.10: Visualization of the effect of adding sample weights in SVM training. A larger circle represents a greater weight

This formulation was extended to the 5-category dataset scoring scheme. The parameter λ is a SVM empirical parameter, optimized through model cross-validation. This parameter controls the penalty of misclassifying a data point to maximize the margin for the remaining points. Following what was reported in [17, 65], a χ^2 kernel was used in the SVM applied to the SPM with VQ vocabulary generation approach. All others made use of a linear SVM kernel, which has reported comparable accuracy and much more accessible training times [18]. For the 5-category dataset, the SVMs were trained on a one-versus-one scheme for all methods except for the SPM+SC formulation, which used the multi-class SVM as formulated in [41, 35]. The TSVM implementation utilized was an adaptation of the LibSVM library (found in [57]). For the 5-category dataset, it was trained in a one-versus-all scheme. TSVM has two primary regularization parameters, ω and u, and a parameter describing the ratios between the classes. The former were empirically optimized, and the latter was estimated from the known training set. The discriminative portion of the models were trained on 1000 image for each category. The number of labelled images was decreased in each trial, being replaced by unlabelled data (in such a way that the training samples always totalled 1000 images per category).

3.6.1 Cross-validation scheme and testing

A stratified 4-fold cross validation scheme was used for all the experiments. The final model is obtained by aggregating the results of all the folds through model averaging. Testing each model for performance was done on 12500 images on both the binary and the 5-category datasets. In both cases, the number of images belonging to each category was the same, allowing for isotropic priors which simplify calculations without any meaningful loss of generality. Some generalizations were made to simplify the testing scheme and due to time limitations regarding overall experimentation. In particular, optimizations regarding the number of pyramid levels used in SPM were extrapolated to other methods (particularly, it's assumed that if increasing or reducing the number of levels on the pyramid representation in the SPM+VQ and/or SPM+SC approaches yields an accuracy gain, the same should hold true for the other SPM formulations). Whilst the overall ensembles are very similar, this assumption is not necessarily true. However, it's sufficiently fair, and it's an absolute necessity seeing as testing the number of pyramid levels for all formulations would require thrice the number of tests. Furthermore, it's also assumed that the set of parameters that yield the best results for tests over only labelled data should also provide the best (or nearly the best) results for tests over labelled and unlabelled data. The reasoning is again similar, noting that since the labelled data holds more weight due to the training scheme described in 3.6, allowing to save time by not re-generating the visual vocabulary with every trial. A similar assumption was made for the regularization parameters when training TSVMs (the sweep was only carried out for one of the ensembles), again merely to reduce the number of trials needed. It should be noted (as it will be made evident in the next chapter) that often small variations over these parameters result in negligible changes in accuracy.

3.7 Hardware resources

All tests were carried out on two computers, each utilizing an Intel i7 processor with four cores (which can be overclocked up to $2.3GHz$). Furthermore, each has a solid state driver to speed up file I/O operations, 16 GB of RAM memory and a Nvidia GTX970 GPU with the relevant CUDA drivers. As stated in 3.3, training and testing related to ConvNets utilized these GPUs for faster computations. Furthermore, many of the required files were loaded onto the RAM memory to accelerate any file I/O operations (ConvNet training utilized upwards of 12 GB). By contrast, the remaining tests were carried more modestly, utilizing the i7 processors (with multicore support whenever it was available through OpenMP) and the solid state drive.

Chapter 4

Results and discussion

In this chapter, the main result of the multiple experiments are presented. These are primarily illustrated in two forms: confusion matrices, for the accuracy variations with the number of labelled and unlabelled images used, and graphics which visualize the variation of the accuracy of each method with the related parameter sweeps. Unless otherwise stated, parameters sweeps were carried out for the labelled training dataset and the TVSM classifier was trained on the 500 labelled + 500 unlabelled images set.

4.1 Feature descriptor results

The initial experiment to determine whether SIFT or PCA-SIFT translated into better classification results, and whether or not adding colour information improved performance are capture in tables 4.1 and 4.2. These were performed for SPM with VQ on the binary dataset. It's shown that colour information indeed enriches the features captured by the SIFT descriptors, leading to a slight improvement in accuracy. Furthermore. PCA-SIFT marginally outperformed the original SIFT implementation, likely due to discarding noisier, less relevant keypoints. A last, small test was conducted on the influence of the number of colour bins in the histogram representation generated by the BoC descriptor, as shown in 4.2. The 32-bin feature representation yielded the best results, capturing richer colour information without excessive fine details to promote overfitting (and this lowering the accuracy of the final ensemble). Following these results, all remaining experiments utilize PCA-SIFT with information extracted from colour patches.

Table 4.1: Comparison between different combinations of feature descriptors. Tested on a SPM+VQ formulation

Feature Descriptor	**Accuracy**
SIFT	0.76
PCA-SIFT	0.76808
Sift+BoC	0.78016
PCA-SIFT+BoC	0.78824

Table 4.2: Comparison between different number of colour bins for the BC descriptor. Tested on a SPM+VQ formulation in conjunction with SIFT descriptors

Number of colour bins	**Accuracy**
8	0.7672
16	0.7704
32	0.78016

4.2 Binary dataset results

The following sections contains all results pertaining to tests done on the binary dataset. This dataset, as stated in 3.2, is comprised of images belonging to two categories, "cat" and "dog". This was expected to be the most challenging set for all methods other than the ConvNet (which was indeed verified). Of the 25000 labelled images, half were used for testing purposes (6250 for each category) and the remaining for training and model validation. Testing with labelled examples only is meant to provide a "control case" which is utilized to compare the different generative formulations in terms of their effect regarding vocabulary generation and feature reduction.

4.2.1 TSVM regularization parameters

The two plots 4.2 and 4.1 represent the variation of the TSVM classifier for 500 labelled and unlabelled training examples, with a visual vocabulary obtained through SPM+VQ. An assumption was made that the regularization parameters which held the best accuracy result (of 0.71) for these feature vectors would perform comparably well for those resulting from other methods. This was required to reduce the number of tests (which would increase tenfold if these were optimized for each method and dataset). This assumption seems fair given the comparable nature of the feature vectors generated by most methods. The TSVM semi-supervised algorithm will provide results which shall be compared to the ConvNet-assisted semi-supervised learning.

4.2.2 Pre-trained ConvNet

The results for the pre-trained ConvNet highlight its strength for this dataset. Even though only 500 training epochs were carried out to optimize the new, last dense layer for the binary output with 1000 labelled examples for each category, it achieves remarkable performance, with an accuracy of 0.94472. This can be readily justified by the fact that, as stated in 3.2, the Imagenet has a wealth of labelled examples to categories related to the "cat" and "dog" classes. Furthermore, the ConvNet model was trained to distinguish between multiple races, having filters trained to capture features for multiple types of cats and dogs. This further validates its use to guide training on other approaches, following the scheme explored in 3.6.

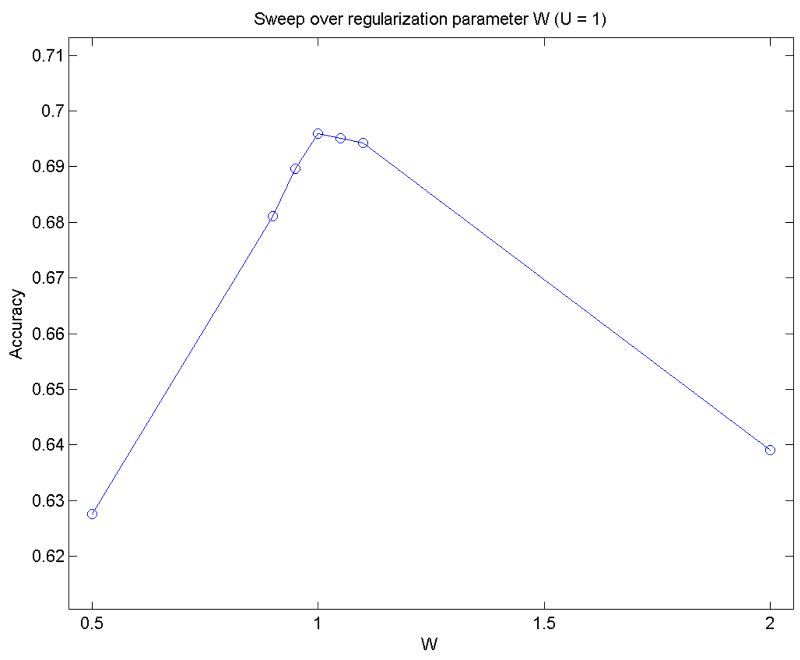


Figure 4.1: Results of the parameter sweep over W for the TSVM implementation, with $U = 1$

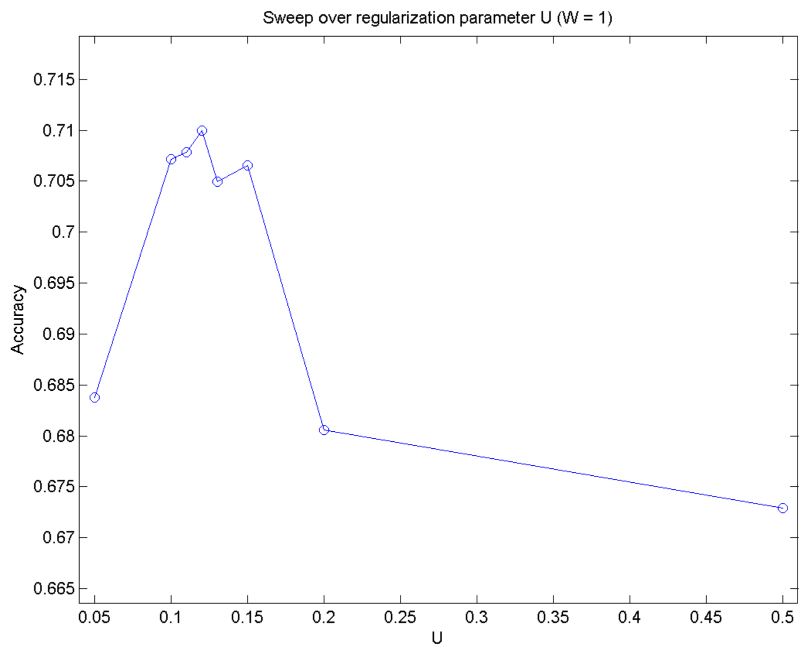


Figure 4.2: Results of the parameter sweep over U for the TSVM implementation, with $W = 1$

4.2.3 SPM with VQ

The results obtained for a SPM implementation which utilizes VQ for visual vocabulary generation achieved a top accuracy of 0.7872, for a visual vocabulary size of 1000 and 3 pyramid levels. An experiment with two pyramid levels was also carried out, resulting in a maximum accuracy

Table 4.3: ConvNet confusion matrix for the binary dataset, accuracy of 0.94472

Labelled \ Predicted	Cat	Dog
Cat	5897	353
Dog	338	5912

of 0.77608, slightly lower than the accuracy obtained for three pyramid levels. Analysing the confusion matrix highlights how the dataset is challenging- the relatively balanced rate of mis-classfication between the two classes seems to reveal that some features might not be sufficiently distinctive between the two categories or, perhaps, that the VQ method for generation of the visual vocabulary induces too much quantization error or groups the keypoints obtained through lower level feature descriptors in suboptimal fashion (namely, creating clusters which bundle features from both classes and result in detection by the SVM classifier in examples of either category). The results for SPM with VQ serve as a baseline, to which the usage of generative methods as a form of feature reduction is compared.

Table 4.4: SPM+VQ confusion matrix for training with 1000 labelled samples for the binary dataset, accuracy of 0.7872

Labelled \ Predicted	Cat	Dog
Cat	4907	1343
Dog	1317	4933

4.2.4 SPM with SC

For the Sparse Coding method of encoding keypoints and generate the visual vocabulary resulted in considerably increased performance compared to the VQ baseline, achieving an accuracy score of 0.81704. The confusion matrix still exhibits a high level of symmetry, reinforcing the notion that the binary dataset proves considerable challenge due to both the irregularity of classes and due to the possibility that some salient features captured by the feature descriptors may overlap in both classes. This point is furthered validate if one considers the multiple different backgrounds for the images of dogs and cats (that is, it's unlikely that some sort of salient keypoint is present in the majority of backgrounds). The increased performance seems to also validate the SC method with max pooling to generate salient, distinctive features, seeing as the model trained only with labelled data outperformed the VQ formulation in accuracy by 4%. Following the scheme elaborated in 3.6, results were obtained for experiments with unlabelled data. The three variations performed comparatively for the case with 750 labelled examples and 250 unlabelled ones, achieving a top score of 0.81816 for the absolute scores scheme. Due to its slightly lower performance, and to save time (each test requires training a new SVM), the worst performing method, weak scores, was abandoned. It should be underlined that, given the large testing dataset, there are enough images with high enough scores so that oversampling the highest scoring data isn't required. This training scheme with the pre-trained ConvNet used in conjunction with unlabelled data kept relatively constant performance (accuracy of 0.81816) until the amount of labelled data went down to 250 labelled examples for the strong scores scheme, where performance dipped to 0.70904. This can be understood by the fact that, due to the soft-margin training of the SVM, some unlabelled examples were misclassified in order to maximize the margin for the labelled examples (that is, the scarcer, labelled data is over-represented during training). This is equivalent to an overfitting problem. The fact that the absolute scores scheme does not share this sharp decrease in performance, remaining at a comparable 0.79296 accuracy rate, seems to support this interpretation (justified by the fact absolute scores, in virtue of the maximization of sample weights, penalizes misclassification of unlabelled data more harshly). That said, although the absolute score scheme outperforms the strong score scheme in this particular instance, this stems from the fact that the pre-trained ConvNet has an exceptionally high accuracy for this dataset. This isn't generically true, and the strong score scheme is preferred as it reflects some level of uncertainty regarding the ConvNet's tentative labelling of the unlabelled data. Parameter sweeps for the dictionary size, K, and the sparsity parameter, λ, resulted in a maximum for $K = 1000$ and $\lambda = 0.34$. Regarding K, these are unsurprising results. It's expected that an insufficient dictionary size does not capture enough salient features to fully characterize a class (further aggravated by the feature reduction resulting from application of a generative method), resulting in underfitting. Likewise, an excessively large dictionary brings the opposite problem, capturing noisy features which do not promote salience on the feature vectors, a problem of overfitting. Initial testing had granted some empirical intuition that optimal dictionary sizes were around 1000, so this was the first value tested upon (and it indeed yelded the best results). As for λ, an accuracy maximum was found for $\lambda = 0.34$. While it could be possible that other maximum points exist, no other values were tested upon both due to time limitations, and due to the findings in [65], where optimal values for the parameter λ were found between 0.3 and 0.4. The highest TSVM accuracy result obtained for this vocabulary was of 0.77.

Table 4.5: SPM+SC confusion matrix for training with 1000 labelled examples for the binary dataset, accuracy of 0.81704

Labelled \ Predicted	**Cat**	**Dog**
Cat	5097	1153
Dog	1134	5116

Table 4.6: SPM+SC confusion matrix for training with 750 labelled and 250 unlabelled examples for the binary dataset (strong scores), accuracy of 0.80928

Labelled \ Predicted	**Cat**	**Dog**
Cat	5058	1192
Dog	1220	5030

Table 4.7: SPM+SC confusion matrix for training with 750 labelled and 250 unlabelled examples for the binary dataset (weak scores), accuracy of 0.806

Labelled \ Predicted	**Cat**	**Dog**
Cat	5077	1173
Dog	1252	4998

Table 4.8: SPM+SC confusion matrix for training with 750 labelled and 250 unlabelled examples for the binary dataset (absolute scores), accuracy of 0.81816

Labelled \ Predicted	**Cat**	**Dog**
Cat	5102	1148
Dog	1125	5125

Table 4.9: SPM+SC confusion matrix for training with 500 labelled and 500 unlabelled examples for the binary dataset (strong scores), accuracy of 0.80928

Labelled \ Predicted	**Cat**	**Dog**
Cat	5058	1192
Dog	1220	5030

Table 4.10: SPM+SC confusion matrix for training with 500 labelled and 500 unlabelled examples for the binary dataset (absolute scores), accuracy of 0.81816

Labelled \ Predicted	**Cat**	**Dog**
Cat	5102	1148
Dog	1125	5125

Table 4.11: SPM+SC confusion matrix for training with 250 labelled and 750 unlabelled examples for the binary dataset (strong scores), accuracy of 0.70904

Labelled \ Predicted	**Cat**	**Dog**
Cat	4462	1788
Dog	1849	4401

Table 4.12: SPM+SC confusion matrix for training with 250 labelled and 750 unlabelled examples for the binary dataset (absolute scores), accuracy of 0.79296

Labelled \ Predicted	**Cat**	**Dog**
Cat	4960	1290
Dog	1298	4952

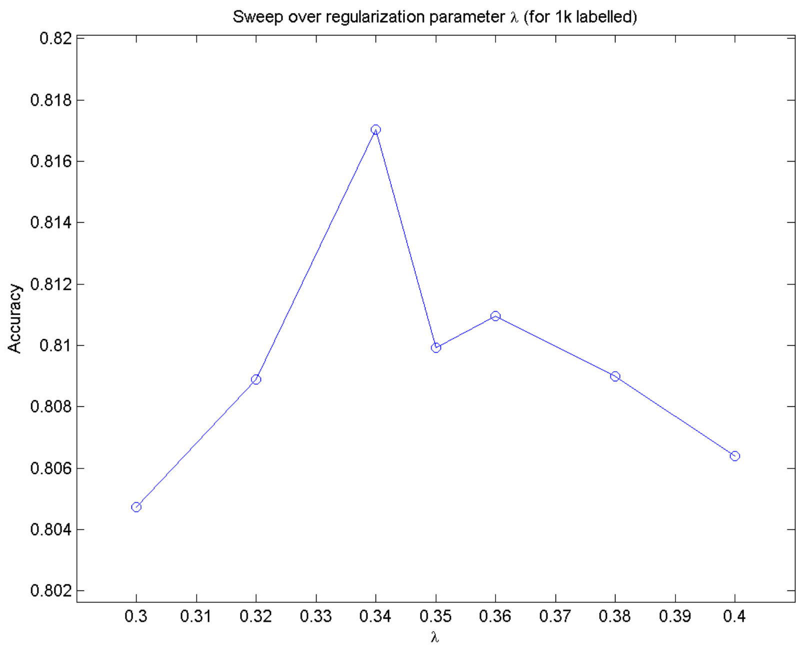


Figure 4.3: Results of the parameter sweep over λ for the SPM+SC method for the binary dataset, with $K = 1000$

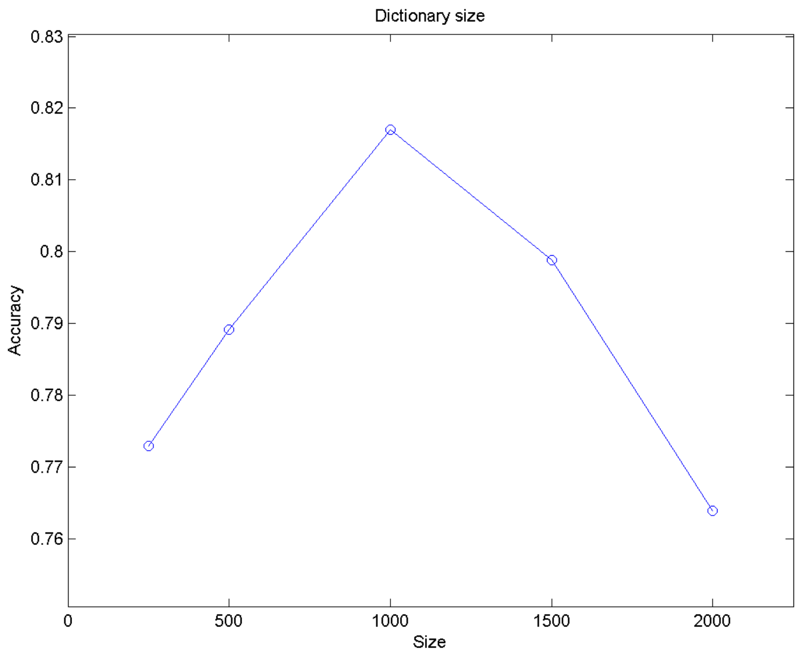


Figure 4.4: Results of the parameter sweep over K for the SPM+SC method for the binary dataset, with $\lambda = 0.81704$

4.2.5 SPM with pLSA

The pLSA ensemble results, again, show how the richer probabilistic description of the properties of the keypoints allied to the feature salience and distinctiveness provided by the generative method upon creating the visual vocabulary can improve performance compared to the regular BoW approach, with an increase of over 5%, with an accuracy score of 0.82696. Similarly, as was the case for the SPM with SC scheme, the ConvNet-assisted semi-supervised learning approach yields favourable results, without any significant performance drop from replacing portion of the labelled data with unlabelled data. Parameter sweeps over the K parameter again yield predictable results. This analysis can be extended to the number of topics, z, which, in practice, share a similar role- too few topics and the topical description isn't rich enough to characterize the classes. Too many, and a form of overftting is verified due to "noisy" topics that do no express sufficient inter-class variance. The highest TSVM accuracy result obtained for this vocabulary was of 0.76.

Table 4.13: SPM+pLSA confusion matrix for training with 1000 labelled examples for the binary dataset, accuracy of 0.82696

Predicted / **Label**	**Cat**	**Dog**
Cat	5160	1090
Dog	1073	5177

Table 4.14: SPM+pLSA confusion matrix for training with 750 labelled and 250 unlabelled examples for the binary dataset (strong scores), accuracy of 0.82496

Predicted / **Label**	**Cat**	**Dog**
Cat	5162	1094
Dog	1100	5150

Table 4.15: SPM+pLSA confusion matrix for training with 500 labelled and 500 unlabelled examples for the binary dataset (strong scores), accuracy of 0.81

Predicted / **Label**	**Cat**	**Dog**
Cat	5064	1186
Dog	1189	5061

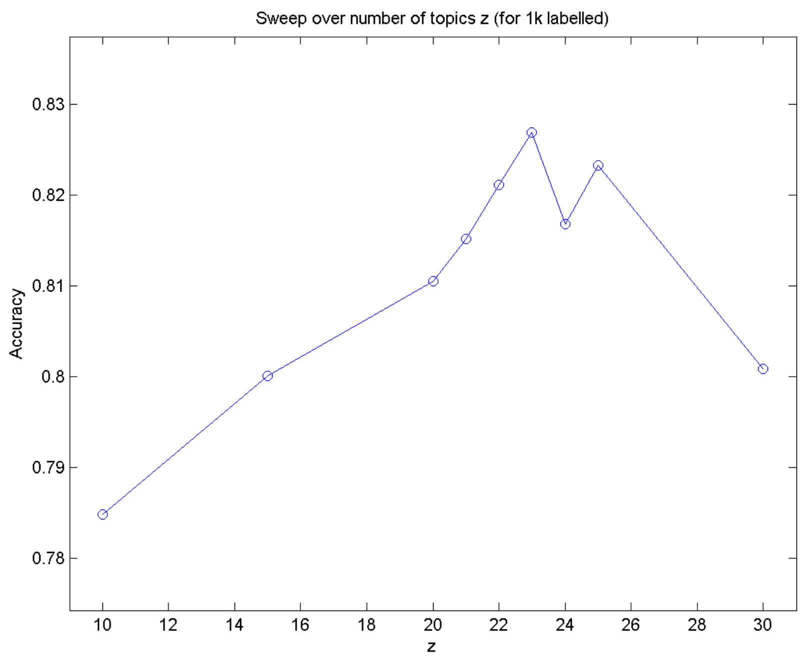


Figure 4.5: Results of the parameter sweep over z for the SPM+pLSA method for the binary dataset, with $K = 1000$

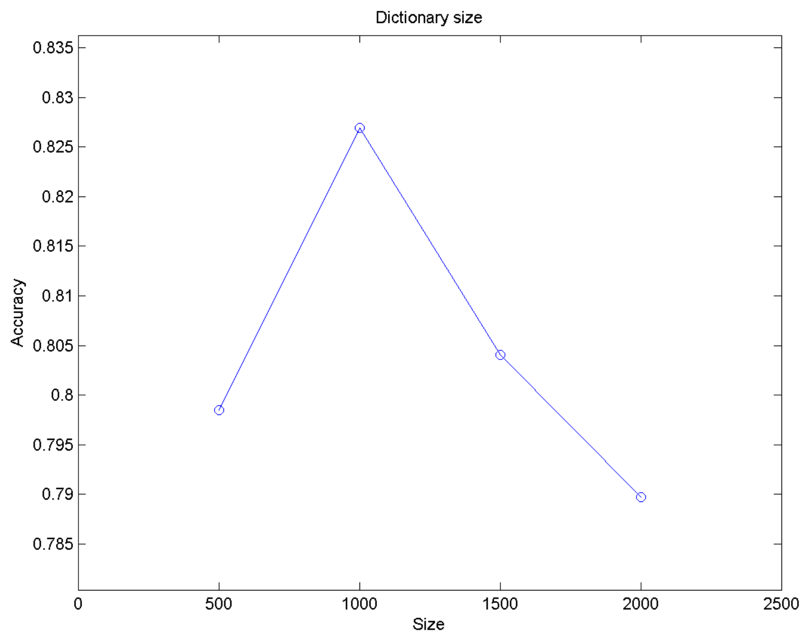


Figure 4.6: Results of the parameter sweep over K for the SPM+SC method for the binary dataset, with $z = 23$

4.2.6 SPM with LDA

The LDA method is a generalization of the pLSA method. Therefore, it comes as a surprise that, whilst it still improvd performance compared to the SPM+VQ ensemble, the obtained accuracy value of 0.80328 is actually lower than that of the pLSA method for topical representation (one would expect, at worse, that both methods would perform evenly, seeing as one includes the other). It should be noted that, firstly, due to the extra α parameter, LDA might be more likely to find local maxima during computation of the EM-algorithm, which might explain its comparatively worse performance for both labelled and labelled+unlabelled datasets. Secondly, due to th aforementioned extra parameter, LDA is more likely to overfit the model [35]. Both these reasons and the fact that LDA presents slower convergence lead to dropping this method in favour of more tests with the better-performing and faster pLSA model. Parameter sweeps over z and K exhibit, as expected, analogous behaviour to the pLSA case, with the highest accuracy value of 0.80328 obtained for $z = 16$ and $k = 1000$. It's interesting to note how a change in the number of topics from 24 to 25 yields a noticeable gain in accuracy. This can be due to multiple reasons, such as the 24th topic erroneously capturing some noisy or irrelevant trend in the data that the 25-topic model does not (what information is captured per topic as the total number of topic changes can be quite unpredictable and, as such, fluctuations like these are, whilst unlikely, not impossible). The highest TSVM accuracy result obtained for this vocabulary was of 0.76.

Table 4.16: SPM+LDA confusion matrix for training with 1000 labelled examples for the binary dataset, accuracy of 0.80328

Label \ **Predicted**	**Cat**	**Dog**
Cat	5015	1235
Dog	1224	5026

Table 4.17: SPM+LDA confusion matrix for training with 500 labelled and 500 unlabelled examples for the binary dataset (strong scores), accuracy of 0.80784

Label \ **Predicted**	**Cat**	**Dog**
Cat	5055	1195
Dog	1207	5043

4.2.7 GMM and FV

Achieving an impressive accuracy rating of 0.83216, the FV method scores the highest amongst all generative methods in terms of performance. This, however, comes at a cost, as the necessary 125 Gaussian components for the optimal model are sadly computationally costly when compared to the other methods. Parameter studies on the FV method focus on two primary variables- the number of Gaussian functions on the GMM and the number of FV components used. Pertaining to

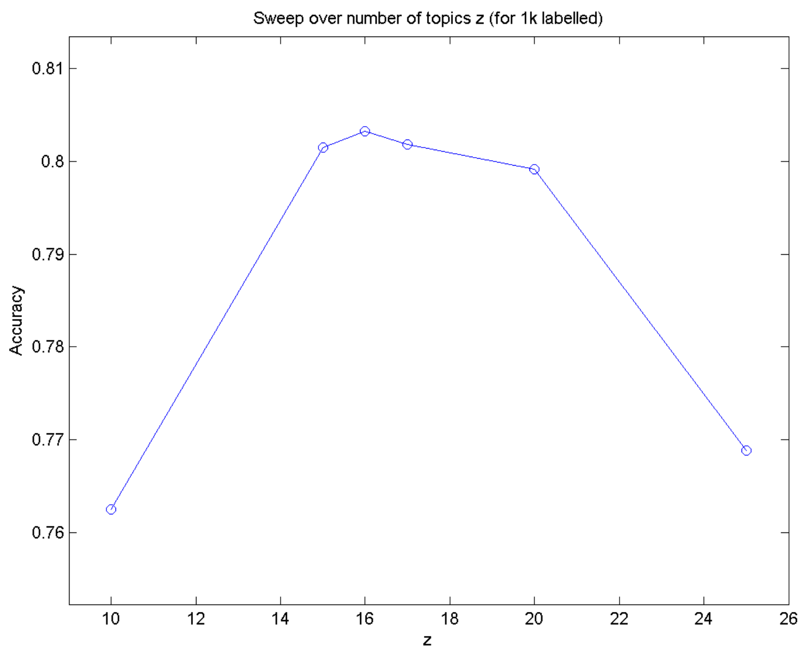


Figure 4.7: Results of the parameter sweep over z for the SPM+SC method for the binary dataset, with $K = 1000$

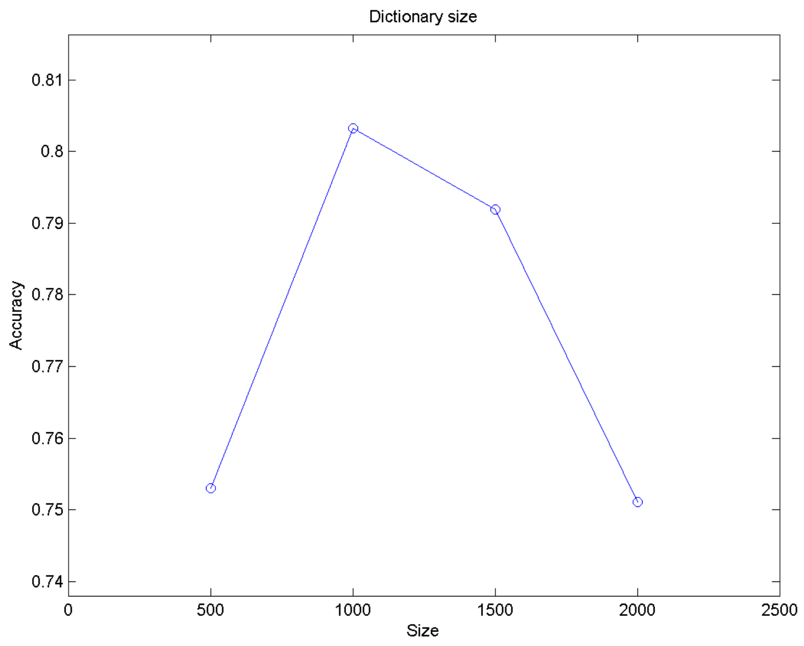


Figure 4.8: Results of the parameter sweep over K for the SPM+SC method for the binary dataset, with $z = 16$

the former, which, as stated, are analogous to the number of clusters in a VQ formulation, graphic 4.9 shows an expected evolution between underfitting and overfitting, with an increase in performance as a greater number of Gaussian functions capture all the meaningful idiosyncrasies of the dataset until a maximum is achieved, past which the new clusters capture nonsense relations and noise instead, resulting in loss of performance. The number of Gaussian functions was quite high even for the binary dataset, which was quite unexpected, but perhaps mirroring the complexity in modelling the two categories due to disparity not only in background, but also high variance within the classes themselves (various races of cats and dogs, for instance). Regarding the components of the FV, as explored in (REF) and as noted in (REF), the μ and σ components present the highest performance. These first and second order statistics seem to express the highest variance pertaining to the categories. Interestingly, the accuracy between using all three FV components and only the μ and σ components seems to be the same. It's likely that the 0th, 1st and 2nd order statistics captured by the GMM weights, means and variances have some overlapping information. In particular, the latter two seem to encapsulate the gross majority held in the former. This result appears to be frequent, as seen in (REF) and (REF). For these reason, in all other ensembles utilizing the FV methodology, only the μ and σ components were considered. Comparing the ConvNet assisted results to the TSVM performance again gives the advantage to the ConvNet, with the highest TSVM being of 0.77904.

Table 4.18: FV confusion matrix for training with 1000 labelled examples for the binary dataset, accuracy of 0.83216

Label \ Predicted	Cat	Dog
Cat	5220	1030
Dog	1068	5182

Table 4.19: FV confusion matrix for training with 500 labelled and 500 unlabelled examples (strong scores) for the binary dataset, accuracy of 0.82784

Label \ Predicted	Cat	Dog
Cat	5180	1070
Dog	1082	5168

Table 4.20: Accuracy variation with the number of FV components considered

Parameters	W	μ	S	$W\mu$	$W\sigma$	$\mu\sigma$	$W\sigma\mu$
Accuracy	0.71008	0.7792	0.79192	0.83216	0.8	0.81504	0.83216

4.2.8 Comparative discussion of the overall results for the binary dataset

All four methods improved the performance of the original SPM formulation with VQ, lending credence to the hypothesis that a statistical formulation of the low-level features extracted from the SIFT and BoC descriptors enriches their information content. Furthermore, the feature reduction

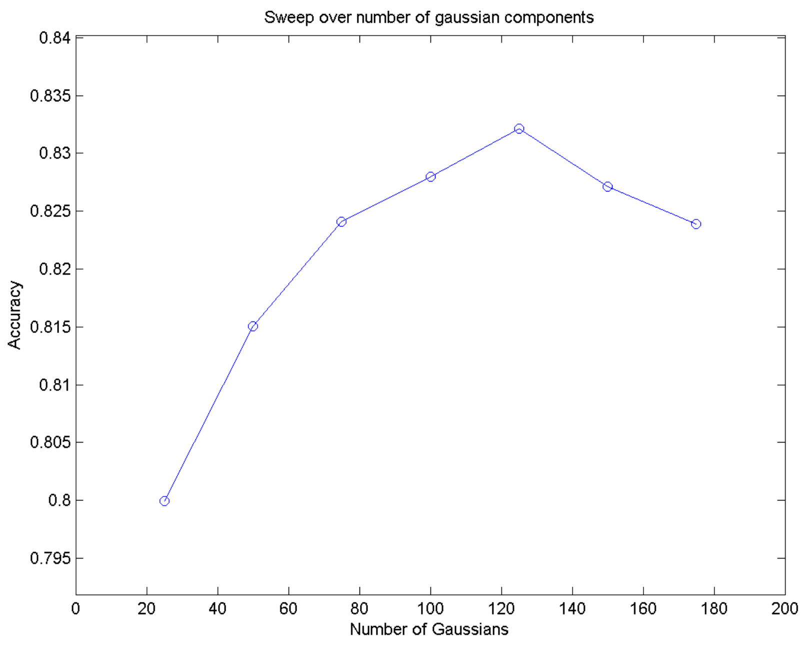


Figure 4.9: Results of the parameter sweep over the number of Gaussian functions for the GMM for the binary dataset, considering the μ and σ FV components

which these generative methods bring results in more distinctive features, allowing for more accurate classification using feature vectors of lower dimensionality and linear classifiers. The pLSA, SC and FV formulations all achieved comparable performance, with a slight edge to the pLSA and FV methods. Surprisingly, LDA underperformed comparatively to pLSA, which led to dropping this method in favour of pLSA for future tests in the 5-category dataset. The ConvNet-assisted training was also shown to outperform the classical TSVM semi-supervised options, validating the viability of pre-trained models as estimators of the class for unlabelled images for sufficiently similar tasks. It should be noted, however, that most TSVM applications are related to text, which typically feature even sparser vectors with a smaller number of features and comparatively less complex patterns. This may reflect on their practical implementations, which may be ill-suited for computer vision tasks. This said, the ConvNet model still outperformed all other approaches. The very same similarity between the classification task for which the ConvNet was originally trained and the one pertaining to this dataset, which allows its use in a semi-supervised approach, also allows for the out-the-box pre-trained ConvNet to outperform all other models. The 5-category dataset will further explore these properties, exposing the ConvNet's weaknesses which were not featured during tests over this dataset.

4.3 5-category dataset results

The results for the extended, 5-category dataset are presented in this secton. This was designed to expose the limitations of the pre-trained ConvNet model, in conrast to the previous binary dataset.

One thousand images were used in each training step. The models were tested on 2500 images of each category, for a total of 12500 images in the testing set. All conditions and assumptions for the binary dataset remained unchanged for these tests.

4.3.1 Pre-trained ConvNet and SPM+VQ baseline

The 5-category dataset exposes the one limitation of ConvNets- with only 1k labelled examples and 500 training epochs, the initial layers of the ConvNet cannot efficiently learn how to pick up distinct features for the new, galaxy class, which has no close analogue in the original ImageNet dataset. This is evident in the confusion matrix- the large volume of misclassified galaxy images across all classes illustrates how distinct features for this classes aren't effectively being picked up by the receptive fields in the convolutional layers. As a result, any noise can cause the neurons to fire erroneously, resulting in the gross misclassification rate for this specific category. This will have consequences in terms of utilizing the ConvNet in assisting the semi-supervised learning. Due to this poor class score, the galaxy classes uses a weak scores approach, instead of the strong scores employed for the remaining categories. However, due to its high accuracy in the other fur categories, the pre-trained ConvNet, with an accuracy of 0.86144, still pulls ahead of the SPM+VQ baseline, with an accuracy of 0.84944.

Table 4.21: ConvNet confusion matrix for the 5-category dataset, accuracy of 0.86144

Predicted / **Labelled**	**Cat**	**Dog**	**Fish**	**Whale**	**Galaxy**
Cat	2245	-	-	-	-
Dog	-	2421	-	-	-
Fish	-	-	2433	-	-
Whale	-	-	-	2368	-
Galaxy	345	366	488	200	1101

4.3.2 SPM with SC

By contrast, the SPM+SC method demonstrates an impressive accuracy of 0.904 for labelled training. The performance for the cat and dog categories remains, as expected, mostly the same, with the comparable misclassification symmetry and rate to the binary case. A similar performance is verified for the whale and fish classes- the bulk of the misclassified examples in either category being attributed to the other. Curiously, the most likely explanation for this is actually the opposite of the justification for the symmetry in the cat and dog classes; due to the similarity in backgrounds between these two classes, and the very comparable colour information (mostly oceanic scenery), it's likely that features captured in the backgrounds (which are quite regular for these classes) are causing this misclassification phenomena. The galaxy class achieves a very high class accuracy of 0.9212. The distinct colour information of elements of this class and its high regularity, both in the objects of interest and the (completely black) backgrounds promotes highly distinct features for this class (it's a class that has very little in common with the others). Training of this class benefited very little from the ConvNet, as most class scores attributed to elements of the galaxy category were comparatively low. As seen before, as the numbered of labelled examples starts to become progressively smaller, overfitting starts to degrade the results. Interestingly, because of the high class regularity for galaxy-type objects, these don't suffer as sharp a decay in class accuracy. For this reason, and due to the fact that the few misclassified galaxy images appear irregularly across classes, the confusion matrix does not include information about galaxy-as-another and another-as-galaxy misclassification. Rather than representing a systemic error or some property of the dataset, these errors generally reflect the presence of dark backgrounds, images with poorer quality or simple an erroneous point with a low class score from the ConvNet's tentative labelling process. Parameter sweeps were again carried out for λ and K, yielding, similar to the binary case, a parameter pair $(\lambda, K) = (0.36, 1500)$ that yields the maximum accuracy, as seen in 4.11 and 4.10. The same curve, which illustrates under and overfitting on the two extremes was obtained, as expected. The highest TSVM accuracy result obtained for this vocabulary was of 0.85.

Table 4.22: SPM+SC confusion matrix for training with 1000 labelled examples for the 5-category dataset, accuracy of 0.904

Labelled \ Predicted	**Cat**	**Dog**	**Fish**	**Whale**	**Galaxy**
Cat	2018	474	-	-	-
Dog	460	2032	-	-	-
Fish	-	-	2366	126	-
Whale	-	-	83	2412	-
Galaxy	-	-	-	-	2472

Table 4.23: SPM+SC confusion matrix for training with 500 labelled and 500 unlabelled examples for the 5-category dataset, accuracy of 0.90096

Labelled \ Predicted	**Cat**	**Dog**	**Fish**	**Whale**	**Galaxy**
Cat	2010	479	-	-	-
Dog	463	2015	-	-	-
Fish	-	-	2356	111	-
Whale	-	-	82	2401	-
Galaxy	-	-	-	-	2480

Table 4.24: SPM+SC confusion matrix for training with 250 labelled and 750 unlabelled examples for the 5-category dataset, accuracy of 0.81776

Labelled \ **Predicted**	**Cat**	**Dog**	**Fish**	**Whale**	**Galaxy**
Cat	2008	453	-	-	-
Dog	466	2001	-	-	-
Fish	-	-	2011	437	-
Whale	-	-	442	1999	-
Galaxy	-	-	-	-	2303

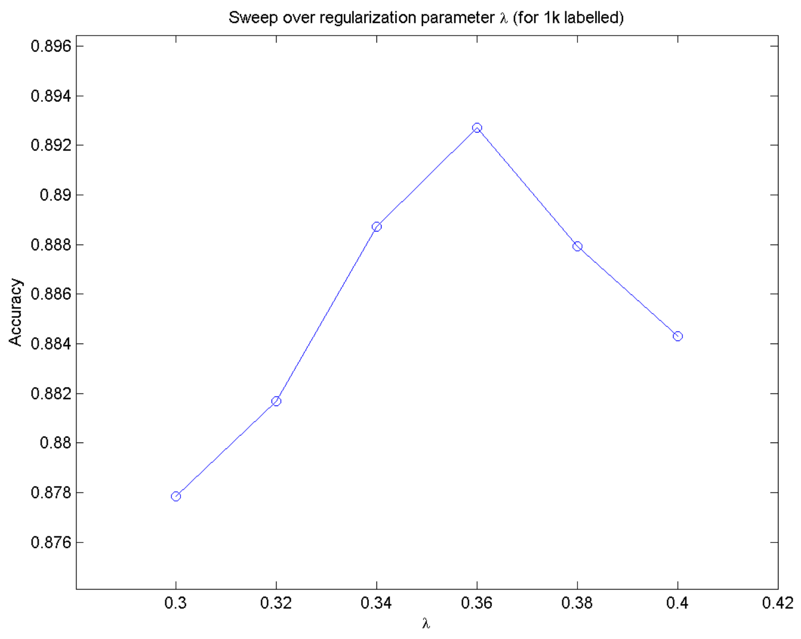


Figure 4.10: Results of the parameter sweep over λ for the SPM+SC method for the 5-category dataset, with $K = 1000$

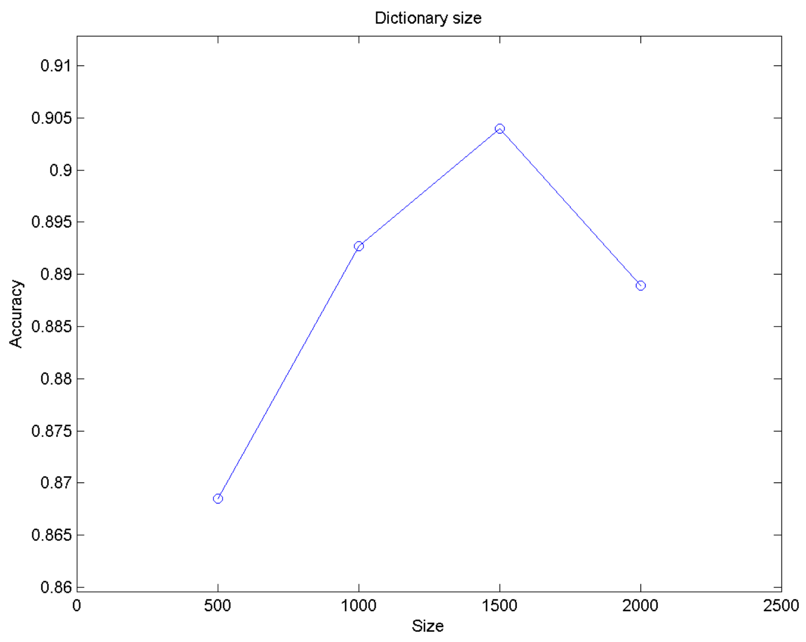


Figure 4.11: Results of the parameter sweep over K for the SPM+SC method for the 5-category dataset, with $\lambda = 0.36$

4.3.3 SPM with pLSA

The SPM+pLSA formulation achieves an acceptable accuracy of 0.8848 on labelled training, which decreases to 0.87648 as unlabelled data is introduced. The simpler multinomial model of the pLSA formulation is more prone to overfit compared to the remaining generative models [84]. Furthermore, the topical representation of the feature space may also be losing an excessive amount of information in the feature reduction step, justifying this slightly lower accuracy score. Again, however, the galaxy class achieves a very high score due to its simplicity and regularity, which makes it comparatively resistant to the adverse effects of overfitting. Again, in a perfectly analogous situation to the experiment on the binary dataset, parameter sweeps for values of z and K were performed, yielding the best results for the parameter pair $(z, K) = (24, 1000)$. The plots 4.13 and 4.12 illustrate the expected behaviour regarding under and overfitting. Lastly, the plot 4.14 shows how the model accuracy can be slightly improved by utilizing a few generated topic feature vectors during training. These need to be used very sparsely since, as seen, they greatly promote overfitting. Usage of these vectors resulted in a gain of almost 1% in terms of accuracy. The highest TSVM accuracy result obtained for the vocabulary obtained through the pLSA formulation was of 0.83.

Table 4.25: SPM+pLSA confusion matrix for training with 1000 labelled examples for the 5-category dataset, accuracy of 0.8848

Labelled \ Predicted	Cat	Dog	Fish	Whale	Galaxy
Cat	2013	429	-	-	-
Dog	440	2004	-	-	-
Fish	-	-	2249	201	-
Whale	-	-	144	2309	-
Galaxy	-	-	-	-	2481

Table 4.26: SPM+pLSA confusion matrix for training with 500 labelled and 500 unlabelled examples for the 5-category dataset, accuracy of 0.87648

Labelled \ Predicted	Cat	Dog	Fish	Whale	Galaxy
Cat	1998	466	-	-	-
Dog	473	2000	-	-	-
Fish	-	-	2231	202	-
Whale	-	-	189	2261	-
Galaxy	-	-	-	-	2466

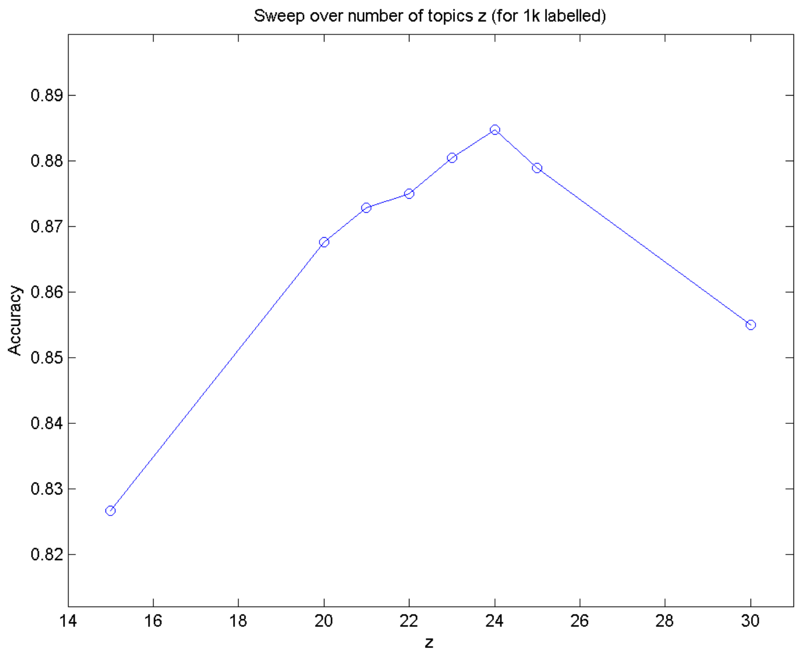


Figure 4.12: Results of the parameter sweep over z for the SPM+pLSA method for the 5-category dataset, with $K = 1000$

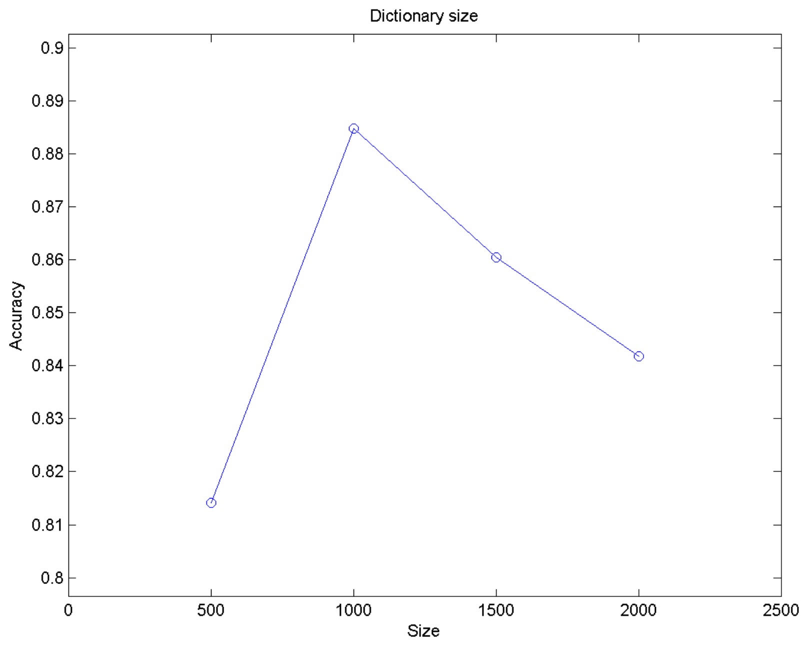


Figure 4.13: Results of the parameter sweep over K for the SPM+pLSA method for the 5-category dataset, with $z = 24$

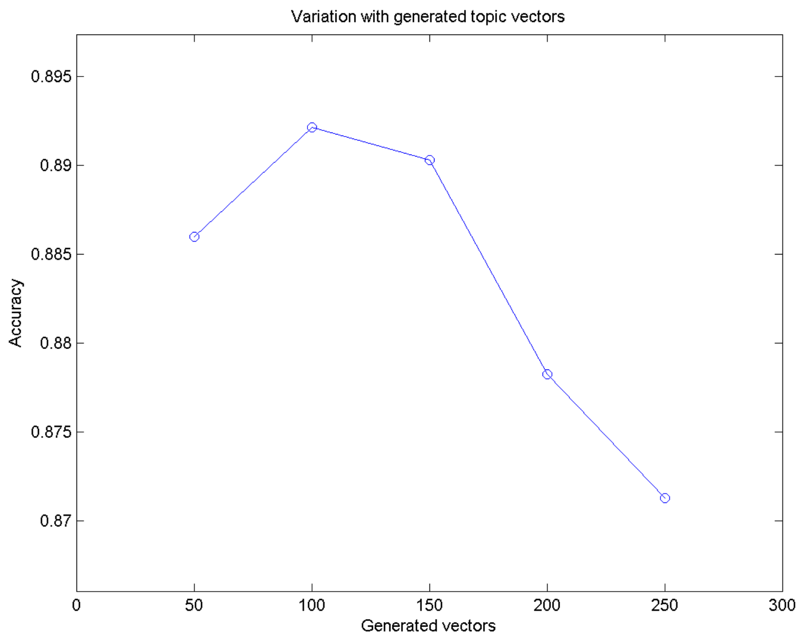


Figure 4.14: Results of utilizing generated feature vectors on the SPM+pLSA method for the 5-category dataset, with $z = 24$ and $K = 1000$

4.3.4 GMM and FV

The Fisher Vector demonstrates, once again, the best results out of all studied generative methods, with a 0.91 accuracy rate. The behaviour of the individual class scores mirrors what was previously observed for other methods, which may hint towards limitations of the feature descriptors and/or linear classifiers- the common elements across all ensembles- more so than a particular limitation or property shared by all the algorithms utilized to create the intermediate visual vocabulary. Additionally, this is also the reflex of the dataset, as previously explained, with the cat and dog classes sharing similarities, the whale and fish classes facing a comparable situation, and the galaxy class being highly regular and quite distinct from the other four. Parameter sweeps with the previously justified FV components of μ and σ increased in accuracy until 200 Gaussian functions were considered. Sadly, due to computational limitations and lack of time, higher number of components could not be tested up (a 200 component GMM running in excess of 9 hours). It's possible that slightly higher number of components could've yielded a slight performance increase (although it's likely this value was very close to the maximum, as eventually overfitting would be observed, as seen in the binary case). In all trials, the FV formulation edged out ahead of all other methods, even if only marginally in some trials, attesting to the effectiveness of the Gaussian distribution at capturing the rich statistical relations in the extracted keypoints. The highest TSVM accuracy result obtained for the FV feature vectors in the 5-category dataset was of 0.86.

Table 4.27: FV confusion matrix for training with 1000 labelled examples for the 5-category dataset, accuracy of 0.91

Predicted / Labelled	**Cat**	**Dog**	**Fish**	**Whale**	**Galaxy**
Cat	2203	260	-	-	-
Dog	341	2101	-	-	-
Fish	-	-	2312	172	-
Whale	-	-	164	2298	-
Galaxy	-	-	-	-	2461

Table 4.28: FV confusion matrix for training with 500 labelled and 500 unlabelled examples for the 5-category dataset, accuracy of 0.90344

Predicted / Labelled	**Cat**	**Dog**	**Fish**	**Whale**	**Galaxy**
Cat	2155	316	-	-	-
Dog	320	2112	-	-	-
Fish	-	-	2270	195	-
Whale	-	-	199	2281	-
Galaxy	-	-	-	-	2475

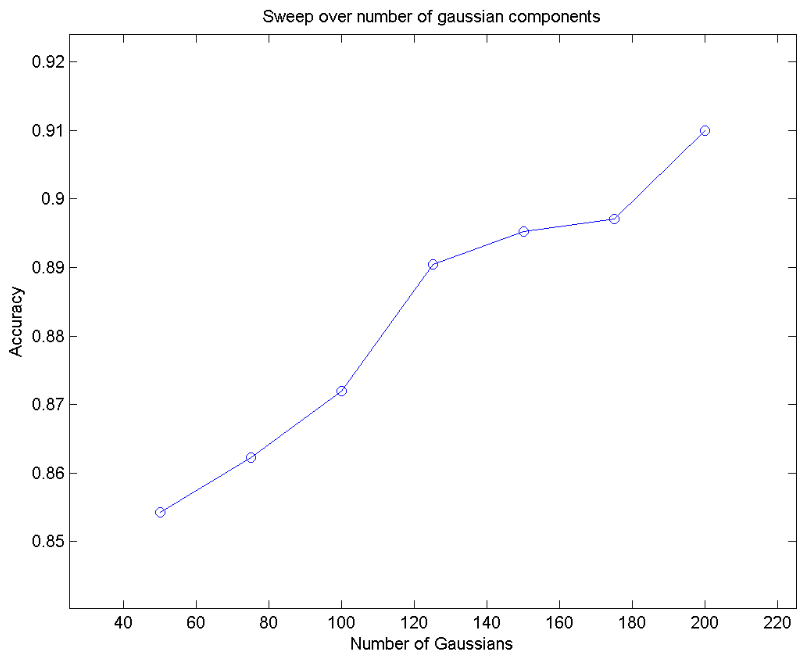


Figure 4.15: Results of the parameter sweep over the number of Gaussian functions for the GMM for the 5-category dataset dataset, considering the μ and σ FV components

4.3.5 Comparative discussion of the overall results for 5-category dataset

Initial analysis of the 5-category dataset yielded surprising results, with all methods achieving remarkable accuracy and the FV method again pulling ahead with an outstanding 0.91 accuracy score for the labelled training. As mentioned, the ConvNet struggles to correctly classify images from the galaxy class. The small number of labelled images and training iterations do not allow the thousands of filters in the convolutional layers to correctly learn distinctive features which can be used to identify and correctly classify objects of this class. Conversely, due to its regularity and distinctiveness from the other classes, the SPM formulations achieved high class accuracy for this particular category. PLSA performed comparatively worse to the other two tested methods, hinting at limitations from this model at dealing with more complex problems, especially from a statistical standpoint (multinomial models being perhaps limited in their statistical description compared to the GMM used by the FV method). Furthermore, the similarity between all results also hints towards limitations or certain properties of the low-level feature descriptors. The same conclusion can be drawn about the linear SVM classifier, although this is less likely. Despite the poor performance of the pre-trained ConvNet on the galaxy category, the TSVM solution was still unable to surpass the performance of the ensembles which utilized it to guide the semi-supervised learning step, the ConvNet likely assisting with classification for the two pairs of classes (cat+dog and fish+whale) in which the SVM tends to struggle the most.

4.3.5.1 Fully-trained ConvNet

In order to explore the true prowess of the deep ConvNet architecture when the limitation on the number of labelled examples is lifted and to verify whether the poor class accuracy obtained for the galaxy class is result of the limited amount of labelled data or the model itself, the ConvNet model was trained with 80000 images from the labelled data of all classes. These were loaded onto the RAM memory and further augmented through the transformations described in 3.2, resulting in over 800000 training examples. This version of the ConvNet model achieved a staggering accuracy score of 0.9752, with only 310 misclassified images over the 12500 present in the testing set, and with no bias towards the galaxy category. It's thus shown that the poor performance of pre-trained ConvNet models stems from insufficient labelled data and excessively dissimilar classification tasks.

4.4 Computational costs and training time

One last important consideration pertains to the computational costs and overall time required to train and test the different models. Creating of the PCA-SIFT keypoints is a relatively lengthy affair (took slightly over 3 hours for the 12500 images of the 5-category dataset), but is only required once per dataset, even if the visual vocabulary has to be recreated or if the SVM classifier has to be retrained. As stated, the ConvNet runs on a Nvidia GPU. Numerical computations on the GPU are considerably faster than those on processors due to the reduced number of system calls. This fact should be omnipresent when comparing runtime between processor and GPU operations. Furthermore, the ConvNet used large amounts of RAM memory for file I/O operations (over 12 GB). With this in mind, tables 4.29 and 4.30 can be briefly analysed. It can be verified that despite its superior performance in terms of accuracy, FV is the slowest of the methods, by a wide margin. Due to the density of its feature vectors, it's also amongst the methods that require the most memory. By contrast, the SPM+SC solution requires the least memory and overall one of the fastest; whilst the reported runtimes often exceeded 6 hours, this is partially due to the lack of multi-core support in Matlab. Were this option available, the runtime would be under 2 hours. To put in perspective the large advantage of GPU processing, training for the last layer of the pre-trained ConvNet took slightly over 2 hours on the Nvidia GTX970, but would take in excess of 70 on the processor. Full training of the ConvNet model took over 90 hours (and an additional 5 hours to classify all data points) with GPU computations, but would take over one month of uninterrupted calculations if it was to be carried out on the i7 processor. The disadvantages of the χ^2 SVM kernel are also made evident in table 4.30, where it's observed that it takes over three times as long to complete its training when compared to the linear kernel. A general rule of thumb for the methods tested is that better accuracy typically comes at a cost of time and computational resources- especially when considering ConvNets.

Table 4.29: Table with typical runtime for visual vocabulary generation and ConvNet training for the 5-category dataset

Method	Time	Multi-Core	GPU	SSD	RAM
SPM+VQ	3hr	yes	no	yes	no
SPM+SC	6hr	no	no	yes	no
SPM+LDA	4.5hr	yes	no	yes	no
SPM+pLSA	3hr	yes	no	yes	no
GMM+FV	>9hr	yes	no	yes	yes
Pre-trained ConvNet	2.5hr	N/A	yes	no	yes

Table 4.30: Table with typical training time for the two used SVM kernels for the 5-category dataset

Method	Time
χ^2	>20hr
linear	6hr

4.5 Closing Considerations

As expected, ConvNets remain unparalleled in terms of performance. If sufficient computational resources and labelled examples are available, they'll always outperform any other of the presented methods. However, when labelled data is limited, simpler SPM formulations paired up with generative methods offer a viable and computationally lighter solution. These can be used ether in alternative to pre-trained ConvNet models or in conjunction with these, in an ensemble that utilizes the strong points of either approach. One thing to note is that the regularity of classes (that is, how similar elements of each class are) is correlated with accuracy. This is an obvious result, as variance within the same class results in more features being captured in each individual image, and also in the possibility of some of those features being similar to those present in other classes (as is the case, for example, for the cat and dog categories). If a class is very regular, it's also very easily predicted as some very salient, distinct features can be found across all elements of such a class. With this in mind, one could split a large class with high variance in its images (again, the cat and dog class present good examples) into multiple subclasses (distinguishing between races, for example, as is done in the Imagenet dataset). While this would result in less labelled example per each of these new subclasses, as the presented methodology showed, a sufficient amount of unlabelled data and pre-trained ConvNet models or generative methods can overcome this issue. Sadly, such a process would initially require manually re-labelling the labelled images for those categories, which is still very undesirable and proved simple too onerous for the time frame available for this project. Considerations about the nature of dataset, the idiosyncrasies of the classification task and limiting factors related to computational resources and time available can weight in favor of some methods and detriment of others. A trade-of was ultimately shown to be present when choosing which method better fits a specific problem.

Table 4.31: Summary table of the best accuracy of each model in each dataset

Dataset / **Method**	**Bin**	**5-class**
SPM+VQ	0.776	0.849
SPM+SC	0.817	0.904
SPM+pLSA	0.827	0.884
SPM+LDA	0.803	-
FV	0.832	0.91
Pre-trained ConvNet	0.945	0.861/0.975

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This study was based on three fundamental questions: whether pre-trained, out-the-box ConvNet could be used to assist semi-supervised learning, whether multiple generative models, namely pLSA, LDA, FV and SC were also viable in increasing performance in a semi-supervised approach and how would these both stack against one another and be potentially used in a single ensemble. Given the limitations on available computational resources and the modest time frame in which the project was developed, these questions were answered satisfactorily. The results validate the hypothesis that pre-trained ConvNet models can be quite useful in providing an earlier estimation of the class to which an unlabelled image belongs, for posterior use in other models. Furthermore, it was shown that, through some empirical tuning of various weight parameters, the class scores generated by these pre-trained models can offer a satisfactory estimation of the confidence for the tentative labelling provided by the ConvNet. Through the various experiments in both datasets, the performance of multiple SPM models with generative methods creating a visual vocabulary on a dataset with a mixture of labelled and unlabelled data was either kept at a competitive accuracy level or, in some cases, surpassed that obtained in a completely labelled dataset. Particularly, pLSA and FV formulations showed slight performance increases in the labelled and unlabelled mixed sets (of slight over 1%), whilst the remaining models showed only very minimal accuracy losses (less than 2% in all cases). This means that the methodology proposed was able to replaced the costly labelled data with unlabelled data, keeping acceptable performance through the usage of a pre-trained model, as originally intended. Furthermore, the study on multiple generative models showed how the statistical formulation of the features extracted by common descriptors like SIFT captures richer information in smaller, sparser feature vectors, increasing both the performance and decreasing the training time for the SVM for all methods (through the replacement of non-linear kernels with linear kernels). The SC, pLSA and FV formulations all showed promising results, the former two with a considerable gain in performance and reduced computational costs compared to more common VQ discriminative approaches, and the latter achieving the best results for the more realistic, 5-category dataset. Although with sufficient labelled data ConvNets remain unparalleled in terms of accuracy, it was shown that for limited data, solution utilizing an appropriately selected pre-trained ConvNet model and one of various generative models for visual vocabulary generation and feature reduction can result in a model which achieves acceptable accuracy, with the added advantage of much shorter computational requirements and training time (in particular, without the need for massive amounts of RAM memory and GPUs dedicated to the training process, as is the case with ConvNets). It was also shown that a small amount of generated, fictitious feature vectors, utilizing the generated probability distributions as estimators, can result in a small increase in performance. While none of the ensembles were able to achieve state of the art results (with the ConvNets remaining uncontested in performance), an alternative method which successfully utilizes unlabelled data for applications where labelled data is scarce and computational resources are limited was achieved. Overall, this study shows that generative methods have much promise, both in semi-supervised approaches or as part of larger, discriminative ensembles.

5.2 Future Work

There's a number of interesting directions in which this study could be expanded. Some notable ones include:

- Better study of the low level descriptors and how they influence the various methods built on top of the keypoints they extract-this includes other types of desriptors, influence of patch sizes and relation to other, more advanced preprocessing operations;
- More thorough sweeps over the multiple parameters of the various generative models;
- Exploring the use of Canonical Cross Correlation instead of Principal Component Analysis;
- Exploring the usage of more sophisticated statistical formulations for generation of fictitious feature vectors;
- Creating more complex ensembles to bolster the performance of ConvNets by using these generative models- for instance, by utilizing them, potentially with deconv layers, to create similar ensembles between each convolutional layer of the deep network;
- Following the previous point, explore the possibility of a truly semi-supervised deep ConvNet;

Furthermore, repeating the various tests on other, larger datasets- specifically the ImageNet dataset- to further test and validate the various models could be carried out. At the time of writing, tests on a 20-category dataset are being carried out but, unfortunately, these results were not agglomerated in time to be presented in this document.

Bibliography

[1] R. C. Gonzalez, R. E. Woods, and B. R. Masters, "Digital Image Processing, Third Edition." *Journal of biomedical optics*, vol. 14, no. 2, pp. 1–976, 2009.

[2] J. Canny, "A Computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[3] R. O. Duda and P. E. Hart, "Use of the Hough Transformation To Detect Lines and Curves in Pictures," vol. 15, no. 1, 1972.

[4] C. M. Bishop, *Neural networks for pattern recognition*, 2005, vol. 92.

[5] P. Paclík, J. Novovičová, and R. P. W. Duin, "A trainable similarity measure for image classification," *Proceedings - International Conference on Pattern Recognition*, vol. 3, no. c, pp. 391–394, 2006.

[6] S. Ertekin, L. Bottou, and C. L. Giles, "Fast Classification with Online Support Vector Machines," *Unpublished Manuscript*, 2010. [Online]. Available: papers2://publication/uuid/A70F175F-7DDF-4EE2-8C77-C357087C5753

[7] M. Guillaumin, J. Verbeek, and C. Schmid, "Multimodal Semi-Supervised Learning for Image Classification," *Cvpr*, pp. 902–909, 2010. [Online]. Available: http://www.computer.org/portal/web/csdl/doi/10.1109/CVPR.2010.5540120

[8] L. I. Smith, "A tutorial on Principal Components Analysis Introduction," *Statistics*, vol. 51, p. 52, 2002. [Online]. Available: http://www.mendeley.com/research/computational-genome-analysis-an-introduction-statistics-for-biology-and-health/

[9] S. Theodoridis, *Machine Learning A Bayesian and Optimization Perspective*, 1st ed., 2015, vol. 53.

[10] D. G. Lowe, "Distinctive image features from scale invariant keypoints," *Int'l Journal of Computer Vision*, vol. 60, pp. 91–11 020 042, 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=996342

[11] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3951 LNCS, pp. 404–417, 2006.

[12] B. T. N. Dalal, "Histograms of Oriented Gradients for Human Detection." *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[13] C. Wengert and M. Douze, "Bag-of-colors for improved image search," 2011.

[14] G. Pass, R. Zabih, and J. Miller, "Comparing images using color coherence vectors," *Proceedings of the fourth ACM international conference on Multimedia (MULTIMEDIA '96)*, pp. 65–73, 1996. [Online]. Available: http://dl.acm.org/citation.cfm?id=244130.244148

[15] S. O'Hara and B. B. a. Draper, "Introduction to the bag of features paradigm for image classification and retrieval," *arXiv preprint arXiv:1101.3354*, no. July, pp. 1–25, 2011.

[16] C. B. Gabriela Csurka, Chris Dance, Lixin Fan, Jutta Willamowski, "Visual Categorization with Bags of Keypoints," *International Workshop on Statistical Learning in Computer Vision*, pp. 1–22, 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.72.604http://www.ncbi.nlm.nih.gov/pubmed/14199369http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC1368854

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

[18] J. Sanchez, F. Perronnin, T. Mensink, J. Verbeek, I. Classification, S. Jorge, P. Thomas, and M. Jakob, "Image Classification with the Fisher Vector : Theory and Practice To cite this version : Image Classification with the Fisher Vector : Theory and Practice," 2013.

[19] G. Tolias, "Fisher Vectors," no. September, 2011.

[20] X. Zhu and A. B. Goldberg, "Introduction to Semi-Supervised Learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 3, no. 1, pp. 1–130, 2009. [Online]. Available: http://www.morganclaypool.com/doi/abs/10.2200/S00196ED1V01Y200906AIM006

[21] S. Antipolis, "Bayesian image classification," *Image and Vision Computing*, vol. 14, pp. 285–295, 1996.

[22] Z. Ghahramani, "Probabilistic Modelling, Machine Learning, and the Information Revolution," Department of Engineering, University of Cambridge,, Tech. Rep., 2012. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.359.7612{&}rep=rep1{&}type=pdf

[23] R. Sundberg, "An Iterative Method for Solution of the Likelihood Equations for Incomplete Data From Exponential Families," p. 10, 1976.

[24] M. Dixit, N. Rasiwasia, and N. Vasconcelos, “Adapted Gaussian Models for Image Classification,” *Computer Vision and Pattern Recognition (CVPR)*, 2011. [Online]. Available: http://www.svcl.ucsd.edu/publications/conference/2011/AdaptedGMM.pdf

[25] A. Najmi and R. M. Gray, “Image classification by a Two Dimensional Hidden Markov Model Hidden Markov Chain,” *Measurement*.

[26] M. Mouret, C. Solnon, and C. Wolf, “Classification of Images Based on Hidden Markov Models,” *2009 Seventh International Workshop on Content-Based Multimedia Indexing*, 2009.

[27] X. Chen, Y. Qi, B. Bai, Q. Lin, and J. G. Carbonell, “Sparse latent semantic analysis,” *Proceedings of the 11th SIAM International Conference on Data Mining, SDM 2011*, pp. 474–485, 2011. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84859173440{&}partnerID=tZOtx3y1

[28] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent Dirichlet Allocation,” *Journal of Machine Learning Research*, vol. 3, no. 4-5, pp. 993–1022, 2012. [Online]. Available: http://www.cs.princeton.edu/{~}blei/lda-c/$\delimiter"026E30F$npapers2://publication/doi/10.1162/jmlr.2003.3.4-5.993$\delimiter"026E30F$npapers2://publication/uuid/4001D0D9-4F9C-4D8F-AE49-46ED6A224F4A$\delimiter"026E30F$npapers2://publication/uuid/7D10D5DA-B421-4D94-A3ED-028107B7F9B6$\delimiter"026E30F$nhttp://www.crossref.org/jmlr

[29] T. Hofmann, “Probabilistic Latent Semantic Indexing,” *Proceedings of the Twent y-Second Annual International SIGIR Conference on Research and Development in Information Retrieval*, 2010.

[30] C.-h. Lee and K.-c. Chiang, “Latent Semantic Analysis for Classifying Scene Images,” *Proceedings of the International MultiConference of Engineers and Computer Scientists 2010*, vol. II, pp. 17–20, 2010.

[31] L. Hong, “A Tutorial on Probabilistic Latent Semantic Analysis,” *Imagine*, no. 2, pp. 1–11, 2010.

[32] A. Y. N. Honglak Lee, Alexis Battle, Rajat Raina, H. Lee, A. Battle, R. Raina, and A. Y. Ng, “Efficient Sparse coding algorithms,” *Advances in nerual infromation processing systems*, vol. 19, no. 2, pp. 801–808, 2006. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.69.2112{&}rep=rep1{&}type=pdf$\delimiter"026E30F$nhttp://books.nips.cc/papers/txt/nips19/NIPS2006{_}0878.txt

[33] M. F. Afonso, “Exploring visual content in social networks,” 2015.

[34] T.-F. Wu, C.-J. Lin, and R. C. Weng, “Probability Estimates for Multi-class Classification by Pairwise Coupling,” *Proceedings of the IEEE Computer Society Conference on Computer*

Vision and Pattern Recognition, vol. 2, no. 2, pp. 2301–2311, 2010. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-45054-8{_}27$\delimiter"026E30F$nhttp://computer.org/tpami/tp2002/i0971abs.htm$\delimiter"026E30F$nhttp://eprints.pascal-network.org/archive/00008315/$\delimiter"026E30F$nhttp://www.csie.ntu.edu.tw/{~}cjlin/papers/svmprob/svmprob.pdf$\delimiter"026E30F$nciteulike-article-id:3047126$\delimiter"026E30F$nhttp

[35] Dan Oneata, "Probabilistic latent semantic analysis," *Proceedings of the Fifteenth conference on Uncertainty . . .* , pp. 1–7, 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=2073829

[36] X. Wu, V. Kumar, Q. J. Ross, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, *Top 10 algorithms in data mining*, 2008, vol. 14, no. 1. [Online]. Available: http://www.scopus.com/scopus/inward/record.url?eid=2-s2.0-37549018049{&}partnerID=7tDmEqzL{&}rel=3.0.0{&}md5=13fa5a34a4668388a6e467930d0b397a

[37] "Cluster kernels for semi supervised learning ."

[38] F. Harrell, *Regression modeling strategies: with applications to linear models, logistic regression, and survival analysis*, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1885585711002726

[39] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[40] a. J. Smola and B. Scholkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, 2004. [Online]. Available: {\T1\textless}GotoISI{\T1\textgreater}://WOS:000222770200003

[41] R. Batuwita and V. Palade, "Class Imbalance Learning Methods for Support Vector," *Imbalanced Learning: Foundations, Algorithms, Applications*, pp. 83–100, 2013.

[42] T. Joachims, "Transductive Inference for Text Classification using Support Vector Machines," *16th International Conference on Machine Learning (ICML-99)*, pp. 200–209, 1999.

[43] J. Wang, X. Shen, and W. Pan, "On Transductive Support Vector Machines," *Prediction and Discovery*, no. 1998, 2005. [Online]. Available: http://www.google.pt/books?hl=pt-PT{&}lr={&}id=0xK9AwAAQBAJ{&}oi=fnd{&}pg=PA7{&}ots=sNdpzxtJOy{&}sig=UmHQIRqnQTiuh7YEoqPq5PgUtQ4{&}redir{_}esc=y{#}v=onepage{&}q{&}f=false

[44] C. Szegedy, W. Zaremba, and I. Sutskever, "Intriguing properties of neural networks," *arXiv preprint arXiv: . . .* , pp. 1–10, 2013. [Online]. Available: http://arxiv.org/abs/1312.6199

[45] Karpathy@cs.stanford.edu, "CS231n Convolutional Neural Networks for Visual Recognition." [Online]. Available: http://cs231n.github.io/convolutional-networks/

[46] J. Dai, Y. Lu, and Y. N. Wu, "Generative Modeling of Convolutional Neural Networks," *Iclr 2015*, pp. 1–13, 2015.

[47] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks arXiv:1311.2901v3 [cs.CV] 28 Nov 2013," *Computer Vision–ECCV 2014*, vol. 8689, pp. 818–833, 2014. [Online]. Available: http://link.springer.com/10.1007/978-3-319-10590-1{_}53$\delimiter"026E30F$nhttp://arxiv.org/abs/1311.2901$\delimiter"026E30F$npapers3://publication/uuid/44feb4b1-873a-4443-8baa-1730ecd16291

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances In Neural Information Processing Systems*, pp. 1–9, 2012.

[49] C. Szegedy, W. Liu, Y. Jia, and P. Sermanet, "Going deeper with convolutions," *arXiv preprint arXiv: 1409.4842*, 2014. [Online]. Available: /citations?view{_}op=view{_}citation{&}continue=/scholar?hl=ja{&}as{_}sdt=0,5{&}scilib=1{&}citilm=1{&}citation{_}for{_}view=KtmM-dAAAAAJ:JV2RwH3{_}ST0C{&}hl=ja{&}oi=p

[50] Y. Lecun, "What's Wrong With Deep Learning?" 2015. [Online]. Available: http://yann.lecun.com

[51] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout : A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research (JMLR)*, vol. 15, pp. 1929–1958, 2014.

[52] N. Srivastava, "Improving neural networks with dropout," *Thesis*, 2013. [Online]. Available: http://www.cs.toronto.edu/{~}nitish/msc{_}thesis.pdf

[53] "CVPR 2015 open access." [Online]. Available: http://www.cv-foundation.org/openaccess/CVPR2015.py

[54] Pierre Chapuis, "Quora - What are the major open problems in computer vision?" [Online]. Available: https://www.quora.com/What-are-the-major-open-problems-in-computer-vision

[55] "ABOUT | OpenCV." [Online]. Available: http://opencv.org/about.html

[56] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, and U. C. B. Eecs, "Caffe: Convolutional Architecture for Fast Feature Embedding," 2014.

[57] The Theano Development Team, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. B. Snyder, N. Bouchard, N. Boulanger-Lewandowski,

X. Bouthillier, A. de Brébisson, O. Breuleux, P.-L. Carrier, K. Cho, J. Chorowski, P. Christiano, T. Cooijmans, M.-A. Côté, M. Côté, A. Courville, Y. N. Dauphin, O. Delalleau, J. Demouth, G. Desjardins, S. Dieleman, L. Dinh, M. Ducoffe, V. Dumoulin, S. E. Kahou, D. Erhan, Z. Fan, O. Firat, M. Germain, X. Glorot, I. Goodfellow, M. Graham, C. Gulcehre, P. Hamel, I. Harlouchet, J.-P. Heng, B. Hidasi, S. Honari, A. Jain, S. Jean, K. Jia, M. Korobov, V. Kulkarni, A. Lamb, P. Lamblin, E. Larsen, C. Laurent, S. Lee, S. Lefrancois, S. Lemieux, N. Léonard, Z. Lin, J. A. Livezey, C. Lorenz, J. Lowin, Q. Ma, P.-A. Manzagol, O. Mastropietro, R. T. McGibbon, R. Memisevic, B. van Merriënboer, V. Michalski, M. Mirza, A. Orlandi, C. Pal, R. Pascanu, M. Pezeshki, C. Raffel, D. Renshaw, M. Rocklin, A. Romero, M. Roth, P. Sadowski, J. Salvatier, F. Savard, J. Schlüter, J. Schulman, G. Schwartz, I. V. Serban, D. Serdyuk, S. Shabanian, É. Simon, S. Spieckermann, S. R. Subramanyam, J. Sygnowski, J. Tanguay, G. van Tulder, J. Turian, S. Urban, P. Vincent, F. Visin, H. de Vries, D. Warde-Farley, D. J. Webb, M. Willson, K. Xu, L. Xue, L. Yao, S. Zhang, and Y. Zhang, "Theano: A Python framework for fast computation of mathematical expressions," p. 19, 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[58] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python," *. . . of Machine Learning . . .*, vol. 12, pp. 2825–2830, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2078195$\delimiter"026E30F$nhttp://arxiv.org/abs/1201.0490

[59] I. Goodfellow and D. Warde-Farley, "Pylearn2: a machine learning research library," *arXiv preprint arXiv:1308.4214*, pp. 1–9, 2013. [Online]. Available: http://arxiv.org/abs/1308.4214

[60] A. Vedaldi, B. Fulkerson, K. Lenc, D. Perrone, M. Perdoch, M. Sulc, and H. Sarbortova, "VLFeat.org: Fisher Vector and VLAD Tutorials." [Online]. Available: http://www.vlfeat.org/overview/encodings.html

[61] "MATLAB - The Language of Technical Computing." [Online]. Available: http://www.mathworks.com/products/matlab/?s{_}tid=hp{_}fp{_}ml

[62] K. B. Petersen and M. S. Pedersen, "The Matrix CookBook," pp. 1–25, 2006.

[63] "Pylearn2 - SGD implementations." [Online]. Available: https://github.com/lisa-lab/pylearn2/pull/136

[64] J. Martens and G. Hinton, "On the importance of initialization and momentum in deep learning," no. 2010, 2012.

[65] J. Yang, K. Yu, Y. Gong and T. Huang., "Linear spatial pyra- mid matching using sparse coding for image classification," *Cvpr'09*, 2009.

[66] a. Bosch, A. Zisserman, and X. Munoz, “Scene classification using a hybrid generative/ discriminative approach,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 4, pp. 712–727, 2008.

[67] A. Goldbloom and B. Hamner, “Kaggle main page.” [Online]. Available: https://www.kaggle.com/

[68] “Dogs vs. Cats.” [Online]. Available: https://www.kaggle.com/c/dogs-vs-cats

[69] “Galaxy Zoo - The Galaxy Challenge.” [Online]. Available: https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge

[70] L. Fei-Fei, K. Li, O. Russakovsky, J. Krause, J. Deng, and A. Berg, “About ImageNet.” [Online]. Available: http://image-net.org/about-overview

[71] “Sander Dieleman - CUDA GPU acceleration for Theano.” [Online]. Available: http://benanne.github.io/2014/04/03/faster-convolutions-in-theano.html

[72] “CUDA Convnet acceleration.” [Online]. Available: https://code.google.com/archive/p/cuda-convnet2/

[73] A. Krizhevsky, “One weird trick for parallelizing convolutional neural networks,” *arXiv preprint*, pp. 1–7, 2014. [Online]. Available: http://arxiv.org/abs/1404.5997

[74] “ConvNetJS Trainer Comparison on MNIST.” [Online]. Available: https://cs.stanford.edu/people/karpathy/convnetjs/demo/trainers.html

[75] M. D. Zeiler, “ADADELTA: AN ADAPTIVE LEARNING RATE METHOD.”

[76] “The Zen of Gradient Descent.” [Online]. Available: s.cmu.edu/~aarti/Class/10701_Spring14/slides/DeepLearning.pdf

[77] U. Montreal, “ADVANCES IN OPTIMIZING RECURRENT NETWORKS Yoshua Bengio, Nicolas Boulanger-Lewandowski and Razvan Pascanu U. Montreal.”

[78] F. O. R. L. Arge and C. I. Mage, “VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION,” pp. 1–14, 2015.

[79] P. Y. Simard, D. Steinkraus, and J. C. Platt, “Best practices for convolutional neural networks applied to visual document analysis,” *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pp. 958–963, 2003.

[80] S. Maji, “A Comparison of Feature Descriptors,” *University of California, Berkeley*, 2006. [Online]. Available: http://www.cs.berkeley.edu/{~}smaji/reports/cs294-6-report.pdf

[81] Y. Ke and R. Sukthankar, “PCA-SIFT: A More Distinctive Representation for Local Image Descriptors.” [Online]. Available: http://www.cs.cmu.edu/{~}yke/pcasift/

[82] S. Lazebnik and C. Schmid, “Beyond Bags of Features : Spatial Pyramid Matching for Recognizing Natural Scene Categories,” pp. 1–12, 2006.

[83] J. Yang, “Jianchao Yang webpage.” [Online]. Available: http://www.ifp.illinois.edu/{~}jyang29/

[84] M. Palm, “latent semantic analysis experiments.” [Online]. Available: https://github.com/matpalm/lsa