CIRCULANT STRUCTURES IN COMPUTER VISION

JOÃO F. HENRIQUES

Department of Electrical and Computer Engineering
Faculty of Science and Technology
University of Coimbra

July 2015

A dissertation submitted in partial fulfillment of
the requirements for the degree of Doctor of Philosophy

ADVISOR
Professor Jorge Batista

Dedicated to my loving parents and brother

ABSTRACT

Visual recognition systems are extremely data-hungry. To accurately recognize a new kind of object, a learning algorithm requires a massive dataset of example images, often augmented artificially by cropping different image regions. More examples seem to invariably raise the computational burden of learning. Is this an inescapable fact? In this thesis, we show that it is not true – that the structure of these datasets hides important shortcuts. The key observation is that samples are not independent, since samples cropped from the same image share most pixels. Using an analytical model of image translation, the cyclic shift, we show that the resulting dataset contains circulant matrices. As a result, we can diagonalize it with the Discrete Fourier Transform (DFT), which reduces both storage and computation by orders of magnitude. The use of the DFT further reveals an interesting link to correlation filters from classical signal processing. We accelerate learning algorithms such as Ridge Regression and Support Vector Regression, addressing linear and non-linear kernel methods. We propose two trackers, the Dual and Kernelized Correlation Filters, which run at hundreds of frames-per-second, and yet perform better than more complex trackers on a 50 videos benchmark. For detection, we propose a decomposition that is several times faster than hard-negative mining, a staple of detector learning. We also generalize these results for other kinds of datasets, such as rotated images or non-rigidly deformed images, which accelerates the learning of pose estimators. The proposed solutions require only a few lines of code to implement, relying on the Fast Fourier Transform and optional off-the-shelf solvers for the bulk of the computations, which easily run in parallel. The software produced during this thesis is open-source.

Keywords: *Computer vision, machine learning, circulant matrices, Discrete Fourier Transform, correlation filters, image transformations, visual tracking, object detection, pose estimation.*

RESUMO

Os sistemas de reconhecimento visual necessitam de vastas quantidades de dados. Para reconhecer um novo tipo de objecto, um algoritmo de aprendizagem requer uma grande base de dados de imagens-exemplo, muitas vezes aumentada artificialmente através da extracção de diferentes regiões dessas imagens. Intuitivamente, processar mais exemplos implica aumentar invariavelmente o custo computacional do processo de aprendizagem. Será que esta intuição corresponde à realidade? Esta tese demonstra que tal não é verdade – que a estrutura destas bases de dados contém atalhos ainda inexplorados. A principal observação é que as amostras não são independentes, já que amostras extraídas da mesma imagem vão ter vários píxeis em comum. Com base num modelo analítico da translação de imagem, chamado "deslocação cíclica", é demonstrado que a base de dados resultante contém matrizes circulantes. Consequentemente, podemos diagonalizá-la com a Transformada de Fourier Discreta (TFD), o que reduz significativamente os requisitos de armazenamento e de computação. O uso da TFD revela uma ligação importante aos filtros de correlação estudados em processamento de sinal. Demonstra-se que é possível acelerar algoritmos de aprendizagem tais como o método dos mínimos quadrados com regularização, e regressão de vectores de suporte, abordando tanto métodos lineares como de kernel (núcleo). São propostos dois métodos de seguimento visual, o Filtro de Correlação Dual e o de Kernel, capazes de processar vídeo a centenas de imagens por segundo, e que demonstram maior precisão que outros métodos mais complexos numa base de dados de 50 vídeos. Para detecção de objectos, é proposta uma decomposição várias vezes mais rápida que a procura sistemática de exemplos negativos, o método mais comum de aprendizagem de detectores. Estes resultados são também generalizados para outros tipos de bases de dados, tais como imagens que sofreram rotação ou deformações não-rígidas, o que permite ainda acelerar detectores de pose. As soluções propostas podem ser implementadas com poucas linhas de código, usando apenas a Transformada de Fourier Rápida, e opcionalmente algoritmos de aprendizagem externos, que podem ser executados em paralelo. O código-fonte relativo a esta tese é de acesso livre.

PUBLICATIONS

The following publications contain preliminary reports of the findings in this thesis:

J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015.

J. F. Henriques, P. Martins, R. Caseiro, and J. Batista. Fast training of pose detectors in the fourier domain. In *Advances in Neural Information Processing Systems*, 2014.

J. F. Henriques, J. Carreira, R. Caseiro, and J. Batista. Beyond hard negative mining: Efficient detector learning via block-circulant decomposition. In *IEEE International Conference on Computer Vision*, 2013.

J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *European Conference on Computer Vision*, 2012.

Publications in related areas were also made during the course of this work:

P. Martins, J. F. Henriques, R. Caseiro, and J. Batista. Bayesian constrained local models revisited. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015 (to appear).

R. Caseiro, P. Martins, J. F. Henriques, and J. Batista. Beyond the shortest path: Unsupervised domain adaptation by sampling subspaces along the spline flow. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

P. Martins, R. Caseiro, J. F. Henriques, and J. Batista. Likelihood-enhanced bayesian constrained local models. In *IEEE International Conference on Image Processing*, 2014.

R. Caseiro, P. Martins, J. F. Henriques, J. Carreira, and J. Batista. Rolling riemannian manifolds to solve the multi-class classification problem. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

R. Caseiro, J. F. Henriques, P. Martins, and J. Batista. Semi-intrinsic mean shift on riemannian manifolds. In *European Conference on Computer Vision*, 2012.

P. Martins, R. Caseiro, J. F. Henriques, and J. Batista. Discriminative bayesian active shape models. In *European Conference on Computer Vision*, 2012.

P. Martins, R. Caseiro, J. F. Henriques, and J. Batista. Let the shape speak - face alignment using conjugate priors. In *British Machine Vision Conference*, 2012.

R. Caseiro, P. Martins, J. F. Henriques, and J. Batista. A nonparametric riemannian framework on tensor field with application to foreground segmentation. *Pattern Recognition*, 45(11), 2012.

R. Caseiro, J. F. Henriques, P. Martins, and J. Batista. A nonparametric riemannian framework on tensor field with application to foreground segmentation. In *IEEE International Conference on Computer Vision*, 2011.

J. Henriques, R. Caseiro, and J. Batista. Globally optimal solution to multi-object tracking with merged measurements. In *IEEE International Conference on Computer Vision*, 2011.

"Your assumptions are your windows on the world.
Scrub them off every once in a while,
or the light won't come in."

— Alan Alda [1]

ACKNOWLEDGMENTS

I would like to start by thanking my advisor, Prof. Jorge Batista. He is responsible for fostering my enthusiasm with Computer Vision as a student, by turning every class assignment into an intriguing research project. He taught me how to be a researcher and gave me immense freedom in pursuing outlandish ideas, showing unwavering support throughout my PhD.

I thank my colleagues and co-authors, who also taught me so much. Pedro Martins and Rui Caseiro, for constantly sharing and debating the latest papers, for helping polish new ideas, for being there through the hard deadlines and for the long reflections on the lowly condition of the graduate student – an endangered species. João Carreira, for never shying away from a research challenge, for his enthusiasm and his critical eye that can turn a good paper into a great one. It was a pleasure working with all of them. I also thank João Faro, David Delgado, Gonçalo Cabrita, Ricardo Faria, and all the colleagues and professors who have helped me one way or another me throughout the years.

I would like to thank my family, my parents and my brother, for their unconditional love and support. My father, for being my role model, for showing me electronics, computers and actual working robots at a very impressionable young age, and for spending an inordinate amount of time explaining to me how they work. He made me realize that scientists must have a lot more fun than astronauts. My mother, for her patience and care, for constantly teaching me how to be a better person, and for helping me keep my feet firmly on the ground. My brother, for the fun times, and for sharing his knowledge about learning from the perspective of the humanities, which colors my views of machine learning.

Last but not least, I thank the people who are closest to me: Sara Ribeiro, André Lages, Artur Serrano and João Pedro Ribeiro. I also

thank all of my dear friends and people who have been a part of my journey, of whom this is but a short list: Vanessa Costa, André Costa, Daniela Rosado, Joana Silva, José Gonçalves, Ana Lopes, Gabriel Salgado, Diogo Pinto, Beatriz Cruz, João Oliveira, André Abrantes. For their compassion and encouragement; for the numerous adventures, for getting me in trouble and getting me out of it every time. For showing me arts, music, for all the laughs over drinks and loud music, and for the long debates on the Big Questions. Thank you!

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The aim of computer vision is to answer meaningful questions about images and video in an automatic way. Although computer vision is a vibrant field and there are a myriad questions that fall under this umbrella, a central problem is that of object recognition – knowing what objects are present, their locations and perhaps other quantitative and qualitative properties. An autonomous system with such a capability can be argued to possess a small degree of intelligence, and its development presents a modest but sure step towards an understanding of cognition [113].

Going beyond pure scientific curiosity, the recent successes of computer vision have catalyzed a wealth of commercial applications that are now commonplace. Automatic focus on faces in consumer cameras [71], unsupervised tagging and categorization of photos on the internet [96, 31], biometric locking of smartphones [53], and gesture-based interfaces for games and entertainment [50] are only a few examples.

The visual world has enough confounding factors, noise, ambiguity and missing information that an accurate physical model of vision would be too complicated and brittle to use for recognition. In fact, the most developed biological vision systems are not born with the ability to see, but the ability to *learn* to see [113]. Some aspects of recognition are innate while others must be learned, a middle-ground within the old *nature vs. nurture* debate in psychology [107].

In the context of artificial vision systems, these concepts can be given very precise meanings. The *nature* aspect corresponds to the features of the system which are fixed – the architecture or mathematical model, that predicts the sought answer from an image. The *nurture* aspect corresponds to the features of the system which are adaptable, and can have different values depending on the environment – the free parameters of the model. The choice of model must reflect prior knowledge that is independent of the environment (e.g. assuming the model belongs to a known family of equations). On the other hand, the free parameters will reflect the particular vagaries of the environment and can only be chosen based on empirical data (e.g. images, measurements and annotations from the real world).

Finding the optimal parameters that best fit the empirical data, referred to as the learning process, usually takes the center stage in most modern systems. However, prior knowledge is still present, implicit in the skillfull engineering of a system. A canonical example is the use of image features instead of pixels, such as Histograms of Oriented Gradients (HOG) [30] or Scale Invariant Feature Transform (SIFT) [82], that are invariant to local non-rigid deformation and illumination. Thus these sources of image variability, which are often not important for recognition tasks, are discarded. Another common example is the application of the model to multiple regions of the image, in a sliding-window manner, which expresses a form of invariance to translation.

The implicit nature of these assumptions can make their engineering very dependent on intuition and experimentation. It would be more desirable to enforce explicit assumptions, such as invariance to translation or rotation, in a systematic manner. Unfortunately, this can lead to hard to analyze and sometimes even intractable models [94, 110, 131].

A simple solution is to only enforce the assumption approximately, by use of virtual samples. The empirical data generally consists of a set of samples (e.g. images). Using the prior knowledge that a transformation (e.g. translation or rotation) does not change the meaning of an image, we can generate new virtual samples by applying the transformation to the original samples.

Virtual samples are commonly used in practice [37, 76, 44, 30, 33], though they are usually treated as an engineering trick and given relatively little discussion. There are two main issues surrounding their use. The first is that, since they are perceived as somewhat obvious and ad-hoc, they have been subjected to relatively little analysis. Nevertheless, an important result by Niyogi et al. [98] states that the use of virtual samples can be equivalent to explicit invariance assumptions, making them theoretically well justified. The second issue is their high computational demand. Generating new virtual samples invariably increases the size of the learning problem, many times by a large factor. This seriously inhibits the amount of virtual samples that are used in practice, and thus the quality of the invariance approximation that can be achieved.

In this thesis, we aim to solve the efficiency issue of virtual samples. We begin by making the observation that solving a standard learning problem with explicitly generated virtual samples is computationally wasteful, because the samples are highly correlated. This can be easily understood from the intuitive fact that, after transforming

an image by translation, most pixels remain the same – they are simply assigned to different positions. The same observation holds for more complicated transformations such as scale, rotation and even non-rigid deformations.

We propose a theoretical framework whereby translations are modeled as cyclic shifts (Chapter 3). The key insight is that datasets with many virtual samples generated by cyclic shifts will acquire circulant structure. Leveraging the relationship between circulant matrices and the Discrete Fourier Transform, we formulate several learning algorithms in a new basis that eliminates all the correlations that arise from virtual samples, reducing computation by orders of magnitude (Chapters 4 and 5). The proposed formulations have the attractive properties of being closed-form, using Fast Fourier Transform operations, and that they decompose the learning problem into parallelizeable chunks. At their core, the decomposed chunks are similar in form to the original learning problem, so they can be solved using standard machinery, which enables the use of highly optimized off-the-shelf libraries.

The same results are then extended to other image transformations, using a variation of the cyclic shift model and circulant matrices (Chapter 6). Because those transformations often do not admit a simple description as a function of the input image, an important innovation is the use of an implicit transformation model. The optimal solution is computed taking into account a transformation model that is latent in example data, but never instantiated by the algorithm. In this way, we can accelerate learning problems with virtual samples obtained by planar rotation, but also highly non-trivial transformations such as out-of-plane rotation and non-rigid deformation.

1.1 BACKGROUND

To provide some context to the contributions in this thesis, it is important to have a notion of how discriminative learning algorithms are typically used in computer vision. We provide a broad overview in Section 1.1.1, which can be safely skipped by someone familiar with this setting.

Sections 1.1.2 and 1.1.3 will then describe what we call the sampling problem, and focus on previous work that is more directly related to the theoretical framework that we develop. Reviews of the literature pertaining to specific applications are given within each chapter of this thesis, where they are the most relevant (Chapters 4-6).

1.1.1 *Image recognition*

The image recognition task that is probably most directly formulated as a learning problem is image classification [105, 69, 78, 55, 116]. Given an image containing mostly a single object, the task is to identify it from a discrete set of classes, such as cats, dogs or humans. Scene classification and fine-grained categorization are related variants [78, 116].

Instead of learning a model over raw pixels, the input to the learned model is typically a representation extracted by a multi-stage pipeline, with the goal of exhibiting invariance to a number of confounding factors. The first stage is usually the extraction of local features over a grid of locations, such as Histograms of Oriented Gradients (HOG) [30], Scale Invariant Feature Transform (SIFT) [82], or Local Binary Patterns (LBP) [99], to mention a few popular descriptors. They are locally invariant to brightness and illumination changes, since most are based on edge detection, and show some invariance to local deformations, by computing statistics over small regions. For that reason, such features are commonly used as a first processing step in virtually all recognition tasks, not just classification. To illustrate this point, we should mention that this is the case for most experiments in this thesis, which are based on HOG features.

Somewhat more specific to classification is a coding or pooling stage, which computes global statistics to form the final representation of the image. Examples are vector quantization or bag-of-words models [126], spatial pyramids [78, 55], Fisher vectors [105] and Vector of Locally Aggregated Descriptors (VLAD) [69]. The global aggregation implicitly yields some invariance to geometric transformations and distortions. A discriminative learning algorithm then learns to predict the image class from this representation. Because the output is discretized into classes, the model in this case is called a classifier. A relatively large dataset of input-output pairs is needed for the algorithm to learn the model's parameters, in this case images and the correct (ground-truth) class.

In the same way, it is also possible to learn a classifier to predict the presence of an object, i.e. the classes are object and non-object (positive and negative classes, respectively). We can then test for the presence of the object at several locations of an image, and in this way perform object detection [30, 44, 13, 135]. Due to the computationally intensive nature of evaluating the model at many locations, the global aggregation techniques used for classification are not common in detection. For that reason, many successful detectors employ

one or more simple linear models over HOG features, evaluated in a sliding-window manner and at multiple scales [30, 44, 13, 135]. Sliding-window detectors were made popular by the well-known Viola-Jones detector [141]. Competitive detectors require large amounts of negative samples, which are also collected using a sliding window. This means that negative samples are related by translation, and can be instantiated as virtual samples. We exploit this fact to accelerate learning, and discuss detection in more detail in Chapter 5.

A very related problem is that of single-object tracking, which consists of following an object given only its initial position and size [127, 148]. It can also be addressed as a detection problem, sharing the same basic approach discussed earlier. A major difference is that as the object is re-detected in a new frame of video, the learned model should be updated to account for the new empirical data that it provides. In this view, tracking is an online learning problem, as oposed to the batch learning problems of detection and classification. We discuss the problem of tracking in more detail in Chapter 4. The samples collected in a new frame are also obtained by translation, and since they all belong to the same image we can make some simplifying assumptions in our analysis of virtual samples.

Predicting other extrinsic aspects of an object's appearance is usually called pose estimation [106, 4]. They may include rigid pose parameters, such as an object's rotation or position relative to the camera, either in 2D or 3D [140]. They may also include non-rigid deformation parameters, such as the relative angles of a person's joints [106]. It is possible to learn a model that predicts the pose directly, as real-valued, continuous variables [106, 4, 140]. Another approach is to discretize it into a set of poses, and to learn a classifier to identify each pose [13, 88]. This method can more directly benefit from the advances in classifier and detector learning. It also makes it easier to trade-off computation (by increasing the number of discrete poses) for increased accuracy. Pose estimation is discussed in more detail in Chapter 6, where the proposed formulation for virtual samples allows us to accelerate the learning process significantly.

1.1.2 *The sampling problem*

A serious challenge for learning algorithms when applied to computer vision is what we will call the "sampling problem". It is mostly an issue of exploiting prior knowledge well. Consider an image that will be used as a sample for learning. Most of the time, any subregion of that image is an equally valid sample. This is especially true

for negative samples (i.e. samples that do not contain an object of interest). Thus, a single image can be a virtually limitless source of samples. Traditional methods deal with this fact by selecting a limited number of samples per image, due to hardware limitations on available memory [30, 44, 151, 73, 117].

The most straightforward method is to simply select the samples randomly, a technique that is most prevalent in tracking applications due to their time-sensitive nature [151, 73, 6, 117, 57]. On the other hand, detector learning mostly relies on hard-negative mining, which is performed offline [44]. It consists of first training an initial detector using random samples (similarly to tracking). This detector is then evaluated on a pool of images, and any wrong detections (named "hard-negatives") are selected as samples for re-training. Hard-negative mining is a very expensive process, but crucial for good detector performance. We discuss it in more detail in Chapter 5. A similar technique is used also in tracking, where detection mistakes are found using a set of structural constraints [73].

A related issue can also occur when evaluating a detector. In order to localize an object, the learned model is evaluated over many subregions of an image. The amount of computation is proportional to the amount of subregions that are considered, mirroring the sampling problem in learning. Several ideas have been proposed in the literature to address this problem. One of them is to use branch-and-bound to find the maximum of a classifier's response while avoiding unpromising candidate regions [77]. Unfortunately, in the worst-case the algorithm may still have to iterate over all regions. Though it does not preclude an exhaustive search, another notable optimization is to use a fast but inaccurate classifier to select promising regions, and only apply the full, slower classifier on those [59, 139]. A related method can quickly discard regions (and thus their subregions) for which the evaluated score will be considered too low [3], however it is formulated only for distances between image pairs.

Although it may not be apparent at first, virtual samples provide an elegant solution to the sampling problem, making it more amenable to analysis. Subregions of an image extracted at slightly different locations are related by translation. One may approximate them from one subregion by generating virtual samples by translation. The approximation is accurate for most pixels, differing only at the borders. We propose to use virtual samples to approximate learning with all possible subregions of several images, which if done naively would be impossible using current hardware. The algorithms proposed in Chapters 4 and 5 allow training with all virtual sample translations

at a fraction of the computational cost of standard methods, such as hard-negative mining.

1.1.3 *Fourier-domain methods*

The initial motivation for this line of research was the recent success of correlation filters in tracking [11, 10]. Correlation filters have proved to be competitive with far more complicated approaches, but using only a fraction of the computational power, at hundreds of frames-per-second. They take advantage of the fact that the convolution of two images (loosely, their dot-product at different relative translations) is equivalent to an element-wise product in the Fourier domain. Thus, by formulating their objective in the Fourier domain, they can specify the desired output of a linear model for several translations, or image shifts, at once.

Fourier transforms have long been used to perform fast convolution, and were employed recently to accelerate detectors at test time [38]. They were also used to accelerate detector training, by modifying an SVM solver with a more efficient subgradient computation [39]. A Fourier domain approach can be very efficient, and has several decades of research in signal processing to draw from [83, 86, 10]. Unfortunately, it can also be extremely limiting. We would like to simultaneously leverage more recent advances in computer vision, such as more powerful features, large-margin classifiers or kernel methods [36, 44, 121].

This hinted that a deeper connection between learning algorithms and the Fourier domain was necessary to overcome the limitations of direct Fourier formulations, motivating the present work.

1.2 OUTLINE

This thesis is organized into 3 introductory chapters, followed by 3 chapters with core contributions, and a concluding chapter.

- Chapter 1 motivates the proposed approach by presenting the "sampling problem", and provides some background on how it fits within the broader context of object recognition in computer vision. More detailed expositions of the state-of-the-art in each application are deferred to the appropriate chapters (Chapters 4-6).
- Chapter 2 briefly reviews the concepts of regularized risk minimization, the dual space, the kernel trick, and shows some examples of machine learning algorithms.
- Chapter 3 introduces circulant matrices, detailing their relationship to cyclic shifts and the Discrete Fourier Transform (DFT) in a self-contained manner. Several aspects of circulant matrices will be derived, which will be useful in later chapters.
- Chapter 4 shows how to exploit the properties of circulant matrices within kernel ridge regression problems, providing an effective sampling of thousands of image patches at a very low computational cost. Two trackers, the Dual and Kernelized Correlation Filters (DCF and KCF) are proposed and evaluated extensively.
- Chapter 5 focuses on linear algorithms, expanding the methodology of circulant matrices to sample from multiple images simultaneously, and to deal with more general learning algorithms than ridge regression. The proposed Circulant Decomposition is evaluated in several object detection tasks.
- Chapter 6 generalizes the results from Chapter 5 to sample image patches at different rotations and other transformations. The proposed algorithms are validated in various detection and pose estimation applications.
- Chapter 7 gives some concluding remarks and directions for future work.

1.3 CONTRIBUTIONS

The broad idea put forward in this thesis is that cyclic shifts provide an accurate model for the sampling process inherent in recognition algorithms, and that these algorithms can be greatly accelerated by leveraging properties of cyclic shifts. In their simplest form, cyclic shifts model samples obtained from images by translation. This thesis presents a systematic study of several algorithms and settings, showing that they exhibit circulant structure under this model, and proposing novel solutions that are orders of magnitude faster than the state-of-the-art. The focus is on learning algorithms of practical relevance in modern computer vision applications.

The contributions in this thesis can be categorized as follows.

1. A self-contained characterization of the most important properties of circulant matrices, in an introductory format, and their relation to cyclic shifts (Chapter 3). A new, short proof of their eigendecomposition is also presented, using only the convolution theorem and elementary matrix algebra (Theorem 3).

2. An efficient solution for linear Ridge Regression under the cyclic shift model. This solution is shown to be formally equivalent to a correlation filter, revealing a link between learning algorithms and classical signal processing (Section 3.8).

3. An analysis of linear and non-linear kernel Ridge Regression under the cyclic shift model (Chapter 4). The focus is on fast solutions that are suitable for online learning, such as tracking.

    a) Proof that, for unitarily invariant kernels, the kernel matrix is circulant (Theorem 4).

    b) Novel solutions in closed-form (with log-linear complexity, using the Discrete Fourier Transform) for:

        i. Kernel Ridge Regression under cyclic shifts. We named this algorithm the Kernelized Correlation Filter (KCF), in analogy with the linear case (Section 4.3).

        ii. Fast detection with kernel classifiers (Section 4.4).

        iii. Computation of a variety of kernels at several image regions, including the popular Gaussian and polynomial kernels (Section 4.4).

    c) A fast extension of classical correlation filters to support multiple channels (and thus modern local features instead

of raw pixels). We named this algorithm the Dual Correlation Filter (DCF).

d) Minimalistic trackers based on KCF and DCF are proposed and evaluated. They are shown to achieve and surpass the state-of-the-art performance, but with an implementation that is simpler and orders of magnitude faster than competing trackers.

4. A novel analysis of general linear regression algorithms under the cyclic shift model (Chapter 5). The previous analysis is extended to deal with multiple sample images simultaneously, in a batch setting.

   a) Proof that the Gram matrix is circulant at the block level (Section 5.2.1).

   b) A closed-form transformation, using the Discrete Fourier Transform, that eliminates redundant degrees of freedom, and simultaneously decomposes the problem into small independent sub-problems (Sec. 5.2-5.3). We call it the Circulant Decomposition. It is valid for several algorithms, including Ridge Regression and Support Vector Regression (Sec. 5.3).

   c) An explicit closed-form expression of the data matrix which allows the use of fast linear solvers [42], that scale linearly with the number of learning samples (Sec. 5.4-5.5).

   d) Detectors based on the Circulant Decomposition are proposed and evaluated. Experiments show that the decomposition accurately approximates learning with all subregions of large training sets (INRIA and Caltech Pedestrians). It is shown to be orders of magnitude faster than hard-negative mining, a staple of detector learning that is very expensive.

5. A generalization of cyclic shifts to model other geometric transformations, not just image translation (Chapter 6). This extension enables applications in more general pose estimation.

   a) Proof that, for cyclic orthogonal transformation models, the Gram matrix is circulant (Theorem 7).

   b) Closed-form solutions that fully exploit the known structure, for Ridge Regression and Support Vector Regression, based on the Discrete Fourier Transform and off-the-shelf solvers.

c) Another closed-form solution for learning multiple classifiers simultaneously, for different poses. It has the same computational cost as the fast solution for a single classifier, yielding another boost in efficiency when applied to pose estimation.

d) A method for simultanous detection and pose estimation is proposed and evaluated. Since the proposed formulas do not require explicitly estimating or knowing the transformation, we demonstrate applicability to both datasets of virtual samples and structured datasets with pose annotations. The performance is shown to be comparable to naive algorithms on 3 widely different tasks, while being several orders of magnitude faster.

2

LEARNING ALGORITHMS

Learning algorithms form the backbone of most modern recognition systems. In their simplest supervised form, they are generic algorithms that learn a function $f : \mathbb{R}^m \to \mathbb{R}$ from a set of examples input vectors $\mathbf{x}_i \in \mathbb{R}^m$ and outputs $y_i \in \mathbb{R}$, for $i \in \{1, \ldots, n\}$. For instance, $\mathbf{x}_i$ may be a collection of images of objects, with $y_i = 1$ if image i contains mugs and $y_i = -1$ otherwise. The learned function f would then distinguish between mugs and other objects. This entails a tremendous flexibility, in that other functions for different objects can be learned in the same way, and variations on this theme can be tailored for different applications.

Informally, the goal is for f to approximate well the set of examples, $f(\mathbf{x}_i) \simeq y_i$, but also to generalize well given unseen data. Assume that the data $(\mathbf{x}_i, y_i)$ is drawn independently and identically distributed (i.i.d.) from an underlying joint distribution $\mathcal{P}$. A straightforward, but very limiting approach would be to assume that $\mathcal{P}$ belongs to a known family of parametric distributions, e.g. a mixture of multivariate Gaussians, and to estimate its parameters from the data. The optimal function f could then be obtained analytically. For example, assuming the data is drawn from two Gaussians with different labels y_i and identical covariances, one obtains the parameters of the optimal linear function f through Linear Discriminant Analysis [40], a classic result. If we assume that the unseen data is drawn from the same distribution, this function will generalize well.

However, real data often does not follow well-behaved distributions. Statistical Learning Theory [137], on the other hand, provides many important results that are distribution-free, and for finite sample sizes. This means that, under mild assumptions on $\mathcal{P}$, such as smoothness and compactness, one can obtain bounds for the mistakes that a given function f will make on unseen data from an arbitrary distribution, when learned from a limited number of samples. One of the most useful formulations is regularized risk minimization [137, 121, 28], which we will use throughout the rest of the thesis. It encompasses several standard algorithms that are routinely used in Computer Vision, some of which will be analized in more detail. It consists of the following optimization problem:

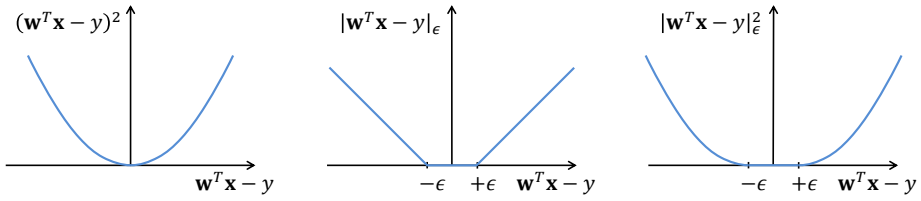
$$\min_f \sum_i^n L\left(f\left(\mathbf{x}_i\right), y_i\right) + \lambda\Omega\left(f\right). \tag{2.1}$$

The first term is the empirical risk, and measures how well f's predictions agree with the data, according to a convex loss function L. As a concrete example, one may use the well-known squared error $L\left(f\left(\mathbf{x}\right), y\right) = \left(f\left(\mathbf{x}\right) - y\right)^2$. The second term contains the regularization $\Omega\left(f\right)$, penalizing complicated functions, which are prone to over-fitting, and thus biasing f towards simpler functions. It can be intuitively understood as a realization of Occam's Razor [121], and is an important ingredient in obtaining good generalization bounds. Finally, the positive factor λ is called the regularization parameter. It controls the relative weight of both objectives in the optimization, effectively trading off between generalizing well but fitting the data poorly, and generalizing poorly but fitting the data well.

A very tractable model, which is often useful in practice, is a linear function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, parameterized by a vector of weights $\mathbf{w} \in \mathbb{R}^m$. In this case, the regularization can take the form $\Omega\left(f\right) = \left\|\mathbf{w}\right\|^2$, i.e., a simple L^2 norm (defined as $\left\|\mathbf{w}\right\|^2 = \mathbf{w}^T\mathbf{w}$). This method is also called Tikhonov regularization [133, 137, 121]. The intuition is that a $\mathbf{w}$ with a larger norm can make the output of f vary abruptly with smaller variations of the input $\mathbf{x}$, and in that sense it is a more complicated, and thus less likely, function. More formally, it can also be shown to correspond to an isotropic Gaussian prior on the weights, by interpreting Eq. 2.1 in a Bayesian setting [121]. A similar norm is defined for non-linear kernel functions, described in Section 2.3. It is possible to choose other regularizers, such as the L^1 norm $\left\|\mathbf{w}\right\|_1$, which promotes sparsity of the parameters [51], or the Total Variation norm $|\nabla\mathbf{w}|$, which promotes smoothness of the parameters (i.e., neighboring elements will tend to have the same values) [100, 51]. We will mostly use the squared norm regularizer throughout the rest of the thesis due to its good analytical properties and relative simplicity.

To further simplify the presentation, the linear model that we consider does not explicitly include a bias term. This can be achieved trivially by appending a constant feature to the samples, so that the corresponding weight represents the bias:

$$f(\mathbf{x}) = \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}^T \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \mathbf{w}^T\mathbf{x} + b. \tag{2.2}$$



(a) Squared loss (b) ϵ-insensitive (c) Squared ϵ-insensitive

Figure 2.1: Examples of loss functions used in regression. (a) Squared loss, used in Ridge Regression. (b) ϵ-insensitive loss, used in Support Vector Regression (SVR). (c) Squared ϵ-insensitive loss, used in L^2-SVR.

Such a bias term b is regularized, as all other weights are, so that a solution with b closer to 0 will be preferred.[1]

2.1 OPTIMIZATION

Eq. 2.1 is very general, and the choice of model f, loss function L and regularizer Ω will result in very different learning algorithms. The chosen loss function and regularizer are typically convex functions, so that the overall objective is convex and thus has a unique global minimum that can be found relatively easily [14]. Several popular algorithms fall under this framework, including the Support Vector Machine, Logistic Regression, Ridge Regression, Ada-boost, LASSO, and many others [132, 46, 28, 121]. In this section we will give a brief overview of algorithms for linear models f and squared-norm regularizers, that are both representative and useful for later chapters. Non-linear models that are nonetheless easy to learn will be treated in Section 2.3, using the "kernel trick". Other non-linear models, such as random forests and other decision trees [40], as well as deep neural networks [76], can lead to non-convex objectives that are harder to analyze, and will not be explored in this thesis.

2.1.1 *Ridge Regression and Least-Squares*

Ridge Regression (RR) has the advantage of having a closed-form solution for the optimization problem, which makes it much easier to

1 Several expositions include an unregularized bias term, which must be handled explicitly [28, 121]. However, unregularized parameters such as the bias can be argued to encourage mild overfitting, and can make a learning problem harder to optimize by removing strong convexity [123].

study. It consists of using the squared loss and squared norm regularizer:

$$\min_{\mathbf{w}} \sum_{i}^{n} \left(\mathbf{w}^T \mathbf{x}_i - y_i\right)^2 + \lambda \left\|\mathbf{w}\right\|^2 . \tag{2.3}$$

Fig. 2.1-a shows a plot of the squared loss versus the model error, $\mathbf{w}^T\mathbf{x} - y$. Note that the loss is non-zero for all $\mathbf{w}^T\mathbf{x} - y \neq 0$, so samples that are very close to the model always incur some penalty, affecting the optimal solution. On the other hand, samples that lie far away from the model (outliers) suffer a quadratic penalty, so they can have a very large influence on the solution even though they do not fit the model at all. This fact has motivated the development of robust, non-quadratic loss functions, some of which will be explored over the following sections.

The objective in Eq. 2.3 is convex, and thus has a unique global minimum. The solution can be obtained by differentiating it with respect to $\mathbf{w}$, obtaining:

$$\mathbf{w} = \left(X^T X + \lambda I\right)^{-1} X^T \mathbf{y}, \tag{2.4}$$

where we define $X \in \mathbb{R}^{n \times m}$ as the data matrix, containing sample $\mathbf{x}_i$ in row i, and $\mathbf{y}$ as the vector of regression targets, with elements y_i. I is the identity matrix of appropriate size. The closed-form solution of Eq. 2.4 allows one to implement Ridge Regression with nothing but a basic numerical package (BLAS).

A special case that is worth noting is when choosing $\lambda = 0$. In this case, one obtains the familiar Least-Squares regression algorithm, which is probably the earliest sistematic approach to data fitting [130]. The lack of regularization means that the Least-Squares algorithm is prone to over-fitting, and this accounts for the fact that it is generally considered a less robust algorithm. However, the Ridge Regression formulation alleviates this issue significantly, and is often competitive in practice with more complicated algorithms [112]. The addition of the diagonal matrix λI, which is positive-definite for $\lambda > 0$, ensures that the matrix inverse in Eq. 2.4 is always well-defined, even in the event of colinear data vectors. The diagonal matrix λI looks like a "ridge", giving the name to this algorithm.

2.1.2 *Support Vector Machine*

The special case of binary regression targets $y_i \in \{-1, 1\}$ arises often, and is refered to as (binary) classification. In this case the output of f

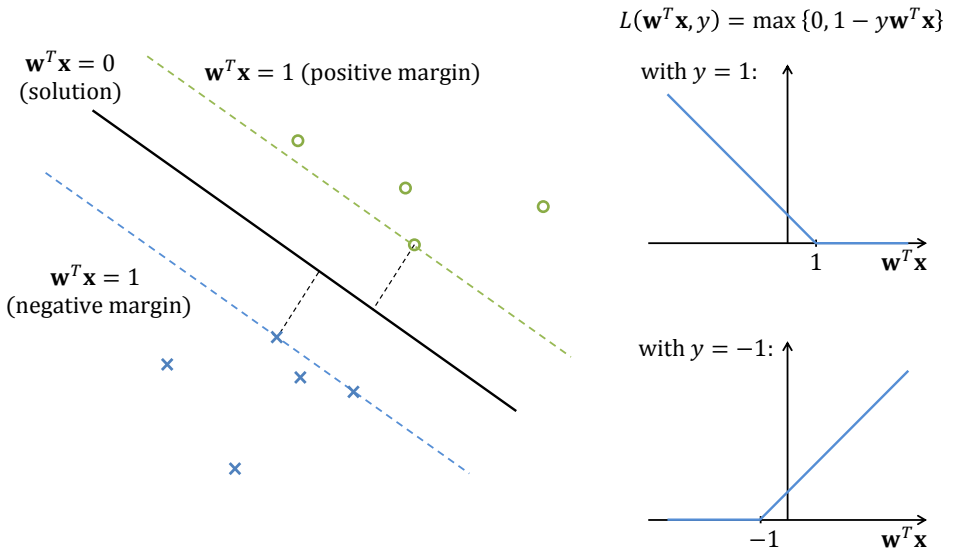


Figure 2.2: Illustration of the margin and hinge loss of a Support Vector Machine (SVM). (left) A linearly separable dataset, represented as green circles (positive samples) and blue crosses (negative samples). In this case, SVM finds the hyperplane $\mathbf{w}^T\mathbf{x} = 0$ that has the largest distance (margin) from the closest positive and negative samples. (right) The hinge loss, shown in the case of a positive sample at the top and negative sample at the bottom. Note if the sample is classified correctly, the loss is 0.

is binarized into one of the regression targets by taking its sign (positive or negative). Ideally, we would like the loss to be 0 for correctly classified samples and 1 (or some other constant) for wrongly classified samples. This would be called the 0-1 loss [121]. Unfortunately, it results in a non-convex minimization problem that can be difficult to optimize. The hinge loss, on the other hand, is a convex upper-bound of the 0-1 loss, so it captures the essence of this goal but results in a convex optimization problem (see Fig. 2.2, right). It is given by:

$$L\left(f\left(\mathbf{x}\right), y\right) = \max\left\{0,\, 1 - y\mathbf{w}^T\mathbf{x}\right\}. \tag{2.5}$$

The hinge loss defines the Support Vector Machine (SVM), one of the most successful classification algorithms to have been proposed based on the concepts of Statistical Machine Learning [27, 28]. The corresponding optimization problem is:

$$\min_{\mathbf{w}} \sum_{i}^{n} \max\left\{0,\, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right\} + \lambda\left\|\mathbf{w}\right\|^2. \tag{2.6}$$

For binary classification, $\mathbf{w}$ can be interpreted as the normal of a hyperplane that separates positive from negative samples [28]. The

SVM objective consists of finding the hyperplane $\mathbf{w}^T\mathbf{x} = 0$ with the highest distance from the closest positive sample and the closest negative sample (Fig. 2.2, left). Intuitively, this leaves the largest possible margin between both distributions, and thus has the lowest chance of misclassification. For this reason, algorithms of this sort are also called Large-Margin Classifiers. Eq. 2.6 is the soft-margin SVM [28], which maximizes the margin but allows some samples to lie on the wrong side of the hyperplane, though incurring a penalty in the objective function. This enables the SVM to work for distributions that cannot be perfectly separated by a hyperplane, which is what typically occurs in practice given noisy data.

The behavior of the hinge loss can be characterized easily: it is 0 for correctly classified samples, and grows linearly as a misclassified sample gets further away from the hyperplane.

Eq. 2.6 can also be formulated as a quadratic minimization with linear inequalities, also known as a Quadratic Program [14], by introducing slack variables ξ_i to replace the max operation:

$$\begin{aligned} &\min_{\mathbf{w}} \ \|\mathbf{w}\|^2 + \frac{1}{\lambda}\sum_{i}^{n}\xi_i \\ &\text{subject to: } \begin{cases} y_i\mathbf{w}^T\mathbf{x}_i \geq 1 - \xi_i \\ \xi_i \geq 0 \end{cases} \\ &\qquad\qquad \forall\, i \in \{1, \ldots, n\}\,. \end{aligned} \tag{2.7}$$

The slack variables ξ_i represent the penalty assigned to each sample for lying on the wrong side of the hyperplane, and thus being misclassified. For correctly classified samples ξ_i will be 0. Quadratic Programs (QP) in general are well-studied in the optimization literature [14]. This QP in particular is convex, and the unique solution can be found by standard solvers, such as ones based on interior-point or conjugate-gradient methods [121]. There are also specialized SVM libraries such as `liblinear` and `libsvm` [23, 42].

2.1.3 *Support Vector Regression*

It is possible to create a loss function that is an analogue of the hinge loss, but for real regression targets instead of binary targets. It is called the epsilon-insensitive loss, and defined by

$$\left|\mathbf{w}^T\mathbf{x} - y\right|_\epsilon = \max\left\{0,\ \left|y - \mathbf{w}^T\mathbf{x}\right| - \epsilon\right\}. \tag{2.8}$$

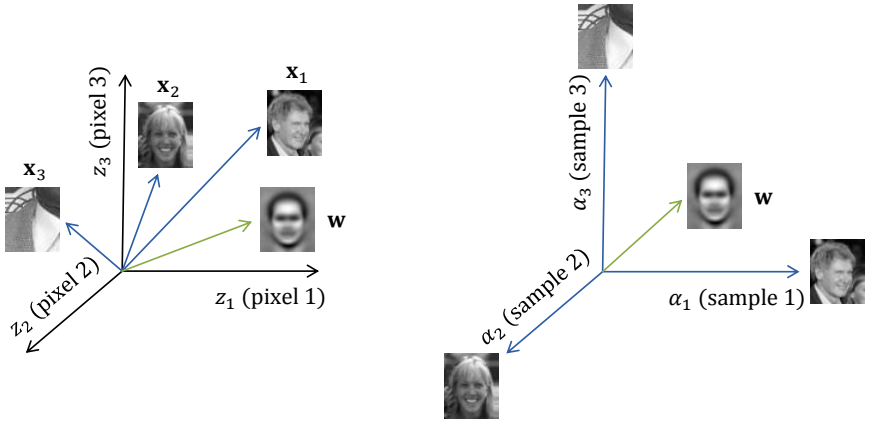
The Support Vector Regression (SVR) algorithm is obtained with this loss function:

$$\min_{\mathbf{w}} \sum_{i}^{n} \left|\mathbf{w}^T\mathbf{x}_i - y_i\right|_\epsilon + \lambda \left\|\mathbf{w}\right\|^2. \tag{2.9}$$

An illustration of the ϵ-insensitive loss is shown in Fig. 2.1-b. The flat region between $-\epsilon$ and ϵ corresponds to samples that are close to the regression model (at a distance smaller than ϵ). As the loss is 0 in this case, samples that are considered correct do not affect the solution. The outer regions correspond to samples that are far from the regression model, and will incur a loss that grows linearly with the distance. Note that the influence of outliers on the solution is diminished, compared to the squared loss (Section 2.1.1). The combination of a 0 loss for correct samples and a linear loss for incorrect samples is similar to the hinge loss that defines the SVM.

A straightforward variant, that sometimes may simplify analysis, is to consider the squared ϵ-insensitive loss instead, $\left|\mathbf{w}^T\mathbf{x} - y\right|_\epsilon^2$ (Fig. 2.1-c). This is usually refered to as the L^2-SVR [121]. Squaring the loss loses the outlier-resistant property of the original ϵ-insensitive loss, however it still seems to work well in practice (Section 5.6). A possible explanation is that outliers in many recognition problems are not placed arbitrarily away from the origin, which would induce a large influence on the solution, because the values of the samples are always bounded to some range. For example, a grayscale image with m pixels may be limited to the range $\mathbf{x} \in [0,\, 255]^m$, and a similar reasoning applies to other image features.

Both SVR and L^2-SVR admit formulations as convex Quadratic Program [121], and SVM libraries usually include specialized SVR solvers [23, 42].



(a) (b)

Figure 2.3: Illustration of primal and dual spaces. (a) The Representer Theorem states that the solution $\mathbf{w}$ is a linear combination of the samples. As a simple example, a face classifier $\mathbf{w}$ can be learned based on pixel values, which constitute the primal space. (b) As $\mathbf{w}$ is a linear combination of samples, the combination coefficients $\boldsymbol{\alpha}$ can be understood as coordinates in a dual space, where each coordinate corresponds to a particular sample instead.

2.2 THE DUAL SPACE

A concept that will play a central role in later chapters is the dual space. It allows us to specify a different formulation for a given regularized risk minimization problem (Eq. 2.1). This alternative view will sometimes expose an underlying structure in the data that is not apparent in the original space (Chapters 4-6).

Lagrangian duality is a very well studied topic in optimization theory [14], and we could certainly use those tools to derive the dual formulations of the learning algorithms from Sections 2.1.1-2.1.3, by treating them as generic constrained optimization problems. However, it is perhaps more instructive to see what the dual space means in the context of learning algorithms. Its essence can be stated quite simply by the Representer Theorem [120]:

Theorem 1 (Representer Theorem [120])**.** *Consider the linear regularized risk minimization problem*

$$\min_{\mathbf{w}} \sum_{i}^{n} L\left(\mathbf{w}^T \mathbf{x}_i,\, y_i\right) + \lambda \left\|\mathbf{w}\right\|^2. \tag{2.10}$$

The solution $\mathbf{w}$ ***belongs to the span of the samples.***
In other words,

$$\mathbf{w} = \sum_{i}^{n} \alpha_i \mathbf{x}_i = X^T \boldsymbol{\alpha}, \tag{2.11}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^n$ *is a vector of coefficients* α_i *that implicitly define the solution.*

Proof. Let us split the solution as $\mathbf{w} = \mathbf{w}_{/\!/} + \mathbf{w}_{\perp}$, into a part $\mathbf{w}_{/\!/}$ that lies in the span of the samples, and another part $\mathbf{w}_{\perp}$ that is orthogonal to them. We have:

- $\mathbf{w}_{/\!/} = \sum_i^n \alpha_i \mathbf{x}_i$, since it can be defined as a linear combination of the samples.
- $\mathbf{x}_i^T \mathbf{w}_{\perp} = 0$ for all $\mathbf{x}_i$, since $\mathbf{w}_{\perp}$ does not intersect the span of the samples.

As such, Eq. 2.10 is equivalent to

$$\min_{\mathbf{w}} \sum_{i}^{n} L\left(\mathbf{w}_{/\!/}^T \mathbf{x}_i + \mathbf{w}_{\perp}^T \mathbf{x}_i,\, y_i\right) + \lambda \left\|\mathbf{w}_{/\!/} + \mathbf{w}_{\perp}\right\|^2 \tag{2.12}$$

$$= \min_{\mathbf{w}} \sum_{i}^{n} L\left(\mathbf{w}_{/\!/}^T \mathbf{x}_i,\, y_i\right) + \lambda \left\|\mathbf{w}_{/\!/}\right\|^2 + \lambda \left\|\mathbf{w}_{\perp}\right\|^2. \tag{2.13}$$

From Eq. 2.13, for a fixed $\mathbf{w}_{/\!/}$ the minimum of the objective is achieved by reducing the norm $\left\|\mathbf{w}_{\perp}\right\|^2$ as much as possible, i.e. $\mathbf{w}_{\perp} = 0$. Because $\mathbf{w}_{\perp} = 0$ for any optimal solution, $\mathbf{w} = \mathbf{w}_{/\!/}$, and thus we have shown that the optimal solution always lies in the span of the samples. □

An illustration of the primal and dual spaces is given in Fig. 2.3.

The Representer Theorem allows one to replace the optimization in m variables, $\mathbf{w} \in \mathbb{R}^m$, with an optimization in n variables, $\boldsymbol{\alpha} \in \mathbb{R}^n$. When there is a large discrepancy between the number of features m and the number of samples n, choosing the most economical representation can provide a significant computational advantage.

For simplicity, Theorem 1 assumes a squared norm regularizer $\Omega\left(f\right) = \left\|\mathbf{w}\right\|^2$. It is possible to generalize it to include monotonically

increasing functions of $\|\mathbf{w}\|$ [120], and non-linear f (Section 2.3). The dual space is fundamental to the development of kernel algorithms, which will be explored in Section 2.3.

We will now make the dual formulation more concrete, by presenting the dual versions of the algorithms from Sections 2.1.1-2.1.3.

2.2.1 *Ridge Regression (dual)*

Since there is a closed-form solution for RR (Eq. 2.4), we can use the explicit relationship in Eq. 2.11 to obtain the dual form of the solution by simple algebraic manipulation [142]. By invoking the Sherman-Morrison-Woodbury formula [67] (also known as the *matrix inversion lemma*),

$$\left(A^{-1} + B^T B\right)^{-1} B^T = AB^T \left(BAB^T + I\right)^{-1}, \tag{2.14}$$

and performing the substitution in Eq. 2.4 with $A = \frac{1}{\lambda} I$ and $B = X$, we obtain

$$\mathbf{w} = X^T \left(XX^T + \lambda I\right)^{-1} \mathbf{y}. \tag{2.15}$$

Comparing this result to Eq. 2.11, the dual space solution is simply

$$\boldsymbol{\alpha} = \left(G + \lambda I\right)^{-1} \mathbf{y}. \tag{2.16}$$

where we have defined for convenience

$$G = XX^T. \tag{2.17}$$

G is called the Gram matrix, and will play a special role in later chapters.

Eq. 2.4 requires taking the inverse of a $m \times m$ matrix, but the inversion in Eq. 2.16 is for a $n \times n$ matrix instead. This can yield substantial computational savings if $n \ll m$.

2.2.2 *Support Vector Machine (dual)*

For optimization problems without a closed-form solution such as the SVM, the direct approach from Section 2.2.1 is not very useful. In general, the dual formulation can be obtained by resorting to the Lagrangian dual of a constrained optimization problem [14]. The coefficients $\boldsymbol{\alpha}$ that we used so far correspond exactly to the Lagrange multipliers obtained in this way, that represent the solution in the dual space [121]. For convex optimization problems such as the ones

studied in this thesis, the primal and dual solutions are exactly equivalent, as evidenced by the Representer Theorem (Theorem 1).

Turning to the dual formulation of the SVM, which can be obtained as the Lagrangian dual of Eq. 2.7 [129, 121], it is

$$\begin{aligned} \min_{\bar{\boldsymbol{\alpha}}} \quad & \frac{1}{2}\bar{\boldsymbol{\alpha}}^T G \bar{\boldsymbol{\alpha}} - \sum_i^n \bar{\alpha}_i \\ \text{subject to:} \quad & 0 \leq \bar{\alpha}_i \leq \frac{1}{\lambda}, \quad \forall i \in \{1, \ldots, n\}\,. \end{aligned} \tag{2.18}$$

To simplify notation, and following standard practice [121], the coefficients $\bar{\alpha}_i$ in Eq. 2.18 differ from the α_i used elsewhere by just a sign flip at the negative samples, i.e. $\alpha_i = y_i \bar{\alpha}_i$ (recall that $y_i \in \{-1, 1\}$).

Just like the primal problem (Eq. 2.7), this optimization problem is a Quadratic Program that can be solved with standard tools [28, 14].

2.2.3 *Support Vector Regression (dual)*

The Lagrangian dual for the SVR can be derived from the primal problem much in the same way as for the SVM [14]. The dual formulation is [121, 66]

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T G \boldsymbol{\alpha} - \boldsymbol{\alpha}^T \mathbf{y} + \epsilon \left|\boldsymbol{\alpha}\right| \\ \text{subject to:} \quad & -\frac{1}{\lambda} \leq \alpha_i \leq \frac{1}{\lambda}, \quad \forall i \in \{1, \ldots, n\}\,, \end{aligned} \tag{2.19}$$

where $|\boldsymbol{\alpha}|$ is the L^1 norm (sum of absolute values) of $\boldsymbol{\alpha}$.

The variant L^2-SVR, which considers the square of the ϵ-insensitive loss, has the dual form [66]

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T G \boldsymbol{\alpha} + \frac{\lambda}{2}\left\|\boldsymbol{\alpha}\right\|^2 - \boldsymbol{\alpha}^T \mathbf{y} + \epsilon \left|\boldsymbol{\alpha}\right|, \tag{2.20}$$

which is very similar to Eq. 2.19, but with no constraints on $\boldsymbol{\alpha}$.

2.3 THE KERNEL TRICK

Consider the learned linear model $f(\mathbf{z}) = \mathbf{w}^T\mathbf{z}$. A different symbol $\mathbf{z}$ is used here to emphasize the fact that it may or may not belong to the set of training samples $\mathbf{x}_i$. The learned function is simply a dot-product in a feature space $\mathbb{R}^m$. However, f can become more powerful if it has access to aspects of the data – *features* – that are more useful

to predict the regression target. With this goal in mind, we can distinguish between the *input space*, which contains our input data $\mathbf{x}'_i \in \mathbb{R}^{m'}$, and the *feature space*, which is the data $\mathbf{x}_i \in \mathbb{R}^m$ that our learning algorithm actually sees. The input data is transformed to the feature space by a function $\varphi : \mathbb{R}^{m'} \to \mathbb{R}^m$,

$$\mathbf{x}_i = \varphi(\mathbf{x}'_i). \tag{2.21}$$

Note that typically $m \gg m'$, since we can extract several characteristics by non-linear combinations of the original input dimensions. Although φ may be non-linear, after mapping the samples with Eq. 2.21 we can still learn a linear model f in this high-dimensional space ($\mathbb{R}^m$), using the algorithms from the previous sections. We can obviously also consider the trivial mapping $\mathbf{x}_i = \mathbf{x}'_i$, so that the feature space coincides with the input space ($m = m'$), in which case the algorithms will simply operate on the raw input data.

The dual algorithms described in Sections 2.2.1-2.2.3 never instantiate a vector in the feature space $\mathbb{R}^m$ – they only have indirect access to it by means of the Gram matrix G (Eq. 2.17). The elements of the $n \times n$ Gram matrix, G_{ij}, store the dot-products between all pairs of samples $(\mathbf{x}_i, \mathbf{x}_j)$,

$$G_{ij} = \mathbf{x}_i^T \mathbf{x}_j. \tag{2.22}$$

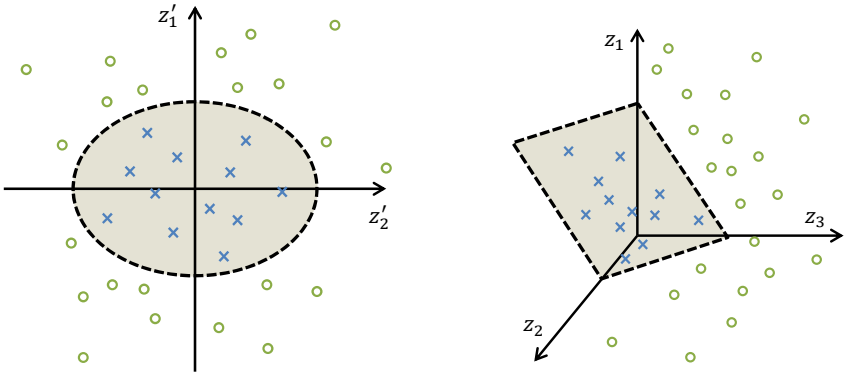
Notice that all the learning algorithms under consideration only require these dot-products to operate, not the original $\mathbf{x}_i$. Assume momentarily that we have found a nice function $\kappa : \mathbb{R}^{m'} \times \mathbb{R}^{m'} \to \mathbb{R}$, that satisfies

$$\kappa(\mathbf{u}, \mathbf{v}) = \varphi^T(\mathbf{u})\,\varphi(\mathbf{v}), \tag{2.23}$$

and can be evaluated in constant time, independent of the size of $\mathbb{R}^m$. The function κ is called a *kernel function* and we will see some specific examples in Sections 2.3.1 and 2.3.2. The kernel function κ allows us to evaluate dot-products in $\mathbb{R}^m$ without creating vectors in that space. If m is very large, or potentially infinite, this can entail a tremendous advantage. The elements of the Gram matrix can then be computed using

$$G_{ij} = \kappa\left(\mathbf{x}'_i, \mathbf{x}'_j\right), \tag{2.24}$$

with a computational load independent of m. In this case the Gram matrix is often called the kernel matrix instead [121], and is usually denoted by K. A dual algorithm can find the optimal model $\boldsymbol{\alpha}$ in the



(a) (b)

Figure 2.4: A non-linear decision boundary, such as the ellipse in (a), can correspond to a linear decision boundary (a hyperplane) in a higher-dimensional space, depicted as a three-dimensional plane in (b). This mapping is at the core of the kernel trick. See text for details.

dual space $\mathbb{R}^n$, without creating any vectors in the potentially large feature space $\mathbb{R}^m$.

We can express the learned function $f(\mathbf{z})$ using the kernel function too. From Eq. 2.11, we map the inputs by φ and apply Eq. 2.23:

$$f(\mathbf{z}') = \left(\sum_i^n \alpha_i \, \varphi(\mathbf{x}'_i) \right)^T \varphi(\mathbf{z}') = \sum_i^n \alpha_i \kappa \left(\mathbf{x}'_i, \mathbf{z}' \right) . \tag{2.25}$$

In this way, the learned function in Eq. 2.25 also does not require any vectors in the high-dimensional $\mathbb{R}^m$. On the other hand, it is now non-linear (even though the learning algorithm operates in a linear space), and the number of kernel function evaluations grows with the number of training samples, n. This methodology, of indirect access to a high-dimensional space by using kernel functions, is called the *kernel trick* [121].

2.3.1 *Example*

We will illustrate the kernel trick using a small toy example, which will hopefully make its principles clear. Consider a dataset with samples $\mathbf{x}'_i \in \mathbb{R}^2$ ($m' = 2$), and binary targets $y_i \in \{-1, 1\}$, illustrated in Fig. 2.4-a. The positive samples (green circles) and negative samples (blue crosses) can be cleanly separated by an ellipse, making an elliptical equation the optimal choice for f (regardless of the loss function). A linear equation, on the other hand, cannot separate the two classes, and would make many classification errors.

However, by means of the non-linear map $\varphi : \mathbb{R}^2 \to \mathbb{R}^3$,

$$\mathbf{z} = \varphi(\mathbf{z}') = \begin{bmatrix} (z'_1)^2 \\ \sqrt{2} z'_1 z'_2 \\ (z'_2)^2 \end{bmatrix}, \tag{2.26}$$

the ellipse becomes an hyperplane in $\mathbb{R}^3$, illustrated in 2.4-b. This is because an *elliptical* equation in the elements of $\mathbf{z}'$ can be written as a *linear* equation in the elements of the higher-dimensional $\mathbf{z}$ (defined in Eq. 2.26).

This shows that a linear function f learned in an appropriate higher-dimensional space $\mathbb{R}^m$ can be as expressive as a non-linear function in the input space $\mathbb{R}^{m'}$.

Now let us analize a dot-product between two vectors $(\mathbf{u}, \mathbf{v})$ mapped by φ:

$$\varphi^T(\mathbf{u})\, \varphi(\mathbf{v}) = \begin{bmatrix} (u_1)^2 \\ \sqrt{2} u_1 u_2 \\ (u_2)^2 \end{bmatrix}^T \begin{bmatrix} (v_1)^2 \\ \sqrt{2} v_1 v_2 \\ (v_2)^2 \end{bmatrix} \tag{2.27}$$

$$= (u_1 v_1)^2 + 2 u_1 u_2 v_1 v_2 + (u_2 v_2)^2 \tag{2.28}$$

$$= \left(\mathbf{u}^T \mathbf{v}\right)^2. \tag{2.29}$$

Eq. 2.29 implies that the kernel function $\kappa(\mathbf{u}, \mathbf{v}) = \left(\mathbf{u}^T \mathbf{v}\right)^2$, which only requires performing a dot-product in $\mathbb{R}^2$ and squaring the output, is equivalent to a dot-product in the mapped space $\mathbb{R}^3$. Although it does not seem like a huge gain, the same property holds for similar kernels of higher-order (e.g. $\kappa(\mathbf{u}, \mathbf{v}) = \left(\mathbf{u}^T \mathbf{v}\right)^b$), which have an exponentially larger dimension. Using the kernel trick, we can compute dot-products in a high-dimensional feature space without incurring the overhead (sometimes prohibitive) of creating vectors in that space.

2.3.2 *Kernel functions*

There are several kernel functions, that result in a myriad of implicit feature spaces.

2.3.2.1 *Polynomial and dot-product kernels*

A generalization of the example kernel from Section 2.3.1 is the *homogenous polynomial kernel*,

$$\kappa(\mathbf{u}, \mathbf{v}) = \left(\mathbf{u}^T \mathbf{v}\right)^b, \tag{2.30}$$

where $b \in \mathbb{N}$ is a tunable parameter that specifies the polynomial's order. This kernel induces a feature map φ into the space of all monomials with order b, i.e., products of exactly b elements of the mapped vector (similar to Eq. 2.26 but for higher orders). The space of all possible monomials grows exponentially with b, and thus the feature space $\mathbb{R}^m$ can be very high-dimensional.

Another variant is the *inhomogenous polynomial kernel*, often referred to simply as "the" polynomial kernel:

$$\kappa(\mathbf{u}, \mathbf{v}) = \left(\mathbf{u}^T \mathbf{v} + a\right)^b. \tag{2.31}$$

With $a \neq 0$, the induced feature space contains all monomials with order up to b. The parameter a is often simply set to 1, but can otherwise be tuned to obtain better numerical conditioning of the computed values.

A more general family is defined by the *dot-product kernel*,

$$\kappa(\mathbf{u}, \mathbf{v}) = g\left(\mathbf{u}^T \mathbf{v}\right), \tag{2.32}$$

for which we have the freedom to choose the function $g : \mathbb{R} \to \mathbb{R}$. It includes polynomial kernels as special cases, but can also be used to construct other kernels.

2.3.2.2 *Gaussian and radial basis function (RBF) kernels*

Another general parametric form is the *radial basis function (RBF) kernel*,

$$\kappa(\mathbf{u}, \mathbf{v}) = h(\|\mathbf{u} - \mathbf{v}\|^2), \tag{2.33}$$

for a given function $h : \mathbb{R} \to \mathbb{R}$. It has the property of translation invariance: $\kappa(\mathbf{u} + \Delta, \mathbf{v} + \Delta) = \kappa(\mathbf{u}, \mathbf{v})$, for any $\Delta \in \mathbb{R}^{m'}$.[2]

A commonly used specialization of the RBF kernel is the *Gaussian kernel*,

$$\kappa(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{1}{\sigma^2} \|\mathbf{u} - \mathbf{v}\|^2\right). \tag{2.34}$$

The Gaussian kernel has a tunable parameter σ, that can be interpreted as a bandwidth or scale. A surprising characteristic of the Gaussian kernel is that it can be shown to correspond to an infinite-

2 Note that translation in this vector space is not the same as the image translation discussed over the following chapters of this thesis, which is defined differently.

dimensional feature space [128]. As such, it would be impossible for a learning algorithm to use such features without the kernel trick.

2.3.2.3 *Additive kernels*

Several kernels can be decomposed into sums of scalar functions, computed over each dimension. They are called *additive kernels*, and can be specified as

$$\kappa(\mathbf{u}, \mathbf{v}) = \sum_{d}^{m'} k\left(u_d, v_d\right), \tag{2.35}$$

where $k : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is the scalar function that defines the additive kernel, and the elements of $\mathbf{u}$ and $\mathbf{v}$ are, respectively, u_d and v_d.

We will now give a few examples of additive kernels. They are all typically used to measure the similarity between two empirical distributions, specified as histograms $\mathbf{u}, \mathbf{v} \in \mathbb{R}_0^+$. They correspond to different distance metrics, an angle that will be explored over the next section. For now we will simply state their definitions. One measure of histogram similarity is the *intersection kernel*,[3]

$$\kappa(\mathbf{u}, \mathbf{v}) = \sum_{d}^{m'} \min\left\{u_d, v_d\right\}. \tag{2.36}$$

Another histogram similarity measure is Hellinger's kernel,

$$\kappa(\mathbf{u}, \mathbf{v}) = \sum_{d}^{m'} \sqrt{u_d v_d}, \tag{2.37}$$

which is also known as Bhattacharyya's coefficient [138]. Finally, we can also consider the χ^2 *kernel*, defined as

$$\kappa(\mathbf{u}, \mathbf{v}) = \sum_{d}^{m'} \frac{2u_d v_d}{u_d + v_d}. \tag{2.38}$$

2.3.2.4 *Kernels and distance metrics*

Eq. 2.36-2.38 are motivated by their correspondence to different distance metrics. Notice that dot-products and the Euclidean distance $\|\cdot\|$ are related by

$$\|\mathbf{u} - \mathbf{v}\|^2 = \mathbf{u}^T\mathbf{u} + \mathbf{v}^T\mathbf{v} - 2\mathbf{u}^T\mathbf{v}. \tag{2.39}$$

3 For general vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}$, the intersection kernel is not positive semidefinite (see Section 2.3.2.5), but only Conditionally Positive Definite [87]. It is possible, however, to restore positive semidefiniteness by shifting the origin so that $\mathbf{u}, \mathbf{v} \in \mathbb{R}_0^+$.

The dot-product in a feature space, computed by a particular kernel, then corresponds to a particular distance metric $\|\cdot\|_\kappa$, using

$$\|\mathbf{u}-\mathbf{v}\|_\kappa^2 = \kappa(\mathbf{u},\mathbf{u}) + \kappa(\mathbf{v},\mathbf{v}) - 2\kappa(\mathbf{u},\mathbf{v}). \tag{2.40}$$

We can use this view of kernels, as generators of distance metrics, to interpret the additive kernels from the previous section. Using the L^1 metric $|\mathbf{u}-\mathbf{v}|$ (sum of absolute differences) to compute distances between histograms can be accomplished by using the intersection kernel of Eq. 2.36 [7, 87]. Conversely, using the Hellinger metric $\left\|\sqrt{\mathbf{u}}-\sqrt{\mathbf{v}}\right\|$ to compute distances is equivalent to using the Hellinger kernel shown in Eq. 2.37 [138]. Finally, the χ^2 kernel in Eq. 2.38 corresponds to using the distance $\chi^2(\mathbf{u},\mathbf{v}) = \frac{1}{2}\sum_d^{m'} \frac{(u_d-v_d)^2}{u_d+v_d}$ between distributions, a metric that is inspired by Pearson's χ^2 test statistic [138].

2.3.2.5 *General conditions for a kernel to be valid*

We will now state a general existence result about kernel functions and induced mappings, to place the above kernels into context. In order for a candidate kernel function κ to induce a feature space (Eq. 2.23) it must be a Mercer kernel [121]. For finite real inputs, a Mercer kernel must satisfy the following conditions:

1. It is symmetric: $\kappa(\mathbf{u},\mathbf{v}) = \kappa(\mathbf{v},\mathbf{u})$.
2. It is a positive semidefinite operator: for an arbitrary choice of vectors $\mathbf{x}'_i$, the corresponding Gram matrix (with elements $G_{ij} = \kappa(\mathbf{x}'_i,\mathbf{x}'_j)$) is positive-semidefinite.

The main advantage of this characterization is that it allows us to prove that κ is a valid kernel, and use it in applications, without having to explicitly find the mapping φ, a task that can be highly non-trivial. For example, we can prove that the Gaussian kernel is valid without dealing with the fact that the dimensionality of an explicit mapping would not be finite (although explicit maps are known [128]). While it is important to know what sorts of functions can be used as kernels, in Chapter 4 we will mainly use the kernel functions discussed over the preceding sections.

3

CIRCULANT MATRICES

Circulant matrices are the main new ingredient of the algorithmic improvements presented in this thesis, as they can model natural regularities commonly found in computer vision datasets (Chapters 4-6). They are square matrices with a simple deterministic pattern (Fig. 3.1), that will be explored over the remainder of the chapter.

Despite their apparent simplicity (or partly because of it), circulant matrices connect vastly different subjects in the landscape of mathematics. Their uses in engineering include the characterization of the limits of linear time-invariant systems in signal processing [56], sparse signal recovery with circulant sensing matrices in compressed sensing [109], and deblurring methods in image processing [8, 100]. They are an important topic in cryptography, being used to construct error-correcting codes [84], and are an integral part of the widely used Advanced Encryption Standard [29]. In physics, circulant matrices are related to the representations of Weyl-Heisenberg groups in discrete quantum mechanics [118, 2], and can be used to obtain solutions of partial differential equations [15]. Circulant matrices have also yielded important insights in the study of roots of general polynomial equations [74], and even in some attempts to prove the infamous Fermat's Last Theorem, which remained unsolved for 358 years [145, 125].

Nevertheless, it may be argued that circulant matrices are interesting mathematical objects in their own right. It is not uncommon for prefaces of monographs on circulant matrices to praise their aesthetically pleasing patterns and remarkably elegant properties [32, 75]:

> *"Some mathematical topics, circulant matrices, in particular, are pure gems that cry out to be admired and studied with different techniques or perspectives in mind."*
>
> I. Kra and S. R. Simanca [75]

The claims in this thesis are much more modest: the intent is to show that circulant matrices fit very nicely the sorts of datasets that appear in computer vision. In exploring the consequences of this proposition, some deeper connections between learning theory and signal processing are also developed.

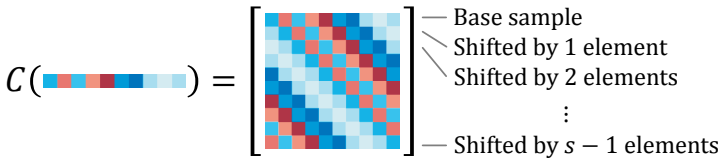


Figure 3.1: Illustration of a circulant matrix. The rows are cyclic shifts of a vector image, or its translations in 1D. The same properties carry over to circulant matrices containing 2D images.

In light of this diversity, it is possible to give a characterization of circulant matrices that is quite general. In order to ground the results in practical computer vision, we will emphasize their relationship to cyclic shifts, and end the chapter with an example that shows a new derivation of classical correlation filters.

3.1 DEFINITION

Consider a given vector $\mathbf{x} \in \mathbb{R}^s$, each element denoted by x_i. We can use it to generate a circulant matrix $C(\mathbf{x}) \in \mathbb{R}^{s \times s}$ as follows:

$$C(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_s \\ x_s & x_1 & x_2 & \cdots & x_{s-1} \\ x_{s-1} & x_s & x_1 & \cdots & x_{s-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_2 & x_3 & x_4 & \cdots & x_1 \end{bmatrix}. \tag{3.1}$$

A more visual representation of the pattern is given in Fig. 3.1. Notice that the pattern is deterministic, and fully specified by the generating vector $\mathbf{x}$, which appears in the first row. Each of the remaining rows is just the previous row shifted to the right by one element, and the right-most element wraps around to the left. We refer to this operation as a cyclic shift, which we will specify more formally in the next section. Shifting without wrapping around would result in a Toeplitz matrix, which is a generalization of circulant matrices [56]. However, Toeplitz matrices do not retain many of the nice properties of circulant matrices, such as the direct relationship to the Discrete Fourier Transform (DFT) which will be the basis for most of our algorithms.

3.2 CYCLIC SHIFTS

We can define the cyclic shift of the rows by resorting to a specific permutation matrix. It is called the *cyclic shift operator*, and is defined by

$$P = \begin{bmatrix} 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}. \tag{3.2}$$

This matrix has the effect of permuting the elements of a vector in the shape necessary to perform a cyclic shift by one element:

$$P\mathbf{x} = [x_s, x_1, x_2, \ldots, x_{s-1}]^T. \tag{3.3}$$

By chaining u applications of the permutation P, which can be achieved with a matrix power P^u, we can shift the vector by an arbitrary amount $u \in \mathbb{Z}$. Negative amounts will simply shift the vector in the reverse direction. However, because P is a permutation matrix with a single cycle of size s [32], its powers are periodic, $P^s = P^0 = I$. This means that with $P^u\mathbf{x}$ we get the same vector $\mathbf{x}$ periodically every s shifts, and thus there are only s unique shifted versions of $\mathbf{x}$:

$$\{P^u\mathbf{x} \,|\, u = 0, \ldots s-1\}. \tag{3.4}$$

We can now use these cyclic shifts to give an alternate definition for a circulant matrix:

$$C(\mathbf{x}) = \begin{bmatrix} (P^0\mathbf{x})^T \\ (P^1\mathbf{x})^T \\ (P^2\mathbf{x})^T \\ \vdots \\ (P^{s-1}\mathbf{x})^T \end{bmatrix}. \tag{3.5}$$

The transpose is only necessary to turn the column-vectors $P^u\mathbf{x}$ into row-vectors. For an arbitrary shift amount u, the row $(u \bmod s)+1$ contains the resulting vector, where mod is the modulus operator (remainder of division).

As an interesting side note, we remark that P itself is a circulant matrix, and can be defined concisely as $P = C([0, \ldots, 0, 1]^T)$.

3.2.1 *Cyclic shifts as a model for translation*

The significance of cyclic shifts for computer vision is that they can be used to effectively model translations in image-space. In the simplest case, $\mathbf{x}$ can be understood to be a one-dimensional image (i.e. a horizontal or vertical slice of a two-dimensional image). This image is translated arbitrarily by $P^u\mathbf{x}$, and $C(\mathbf{x})$ contains all possible cyclic shifts of $\mathbf{x}$. As such, we can use $C(\mathbf{x})$ as the data matrix for a learning algorithm (Chapter 2), which includes many translated images as the samples.

We will now make a short digression into why such a data matrix should be important. In order for a learning algorithm to be useful, it must be trained with samples that represent faithfully the distribution of inputs that will be encountered by the learned function. Image translations are a very common form of nuisance, and learning with translated images is an important factor to increase robustness. Extracting many patches of images at different translations is standard practice in computer vision [30, 6, 57, 4, 76], but this process is very time-consuming and results in large, redundant datasets. Our goal, in modeling translations with cyclic shifts, is to find shortcuts in these redundant computations.

The cyclic shifts described in this Chapter are defined for one-dimensional images. The generalization to two-dimensional images is straightforward, but restricting the derivations to one-dimensional images simplifies notation significantly. In many cases it is also possible to extend the results to multiple channels (e.g. 3 channels in the case of RGB images, or more standard local features, such as Histograms of Oriented Gradients [30]).

3.3 THE DISCRETE FOURIER TRANSFORM

The Discrete Fourier Transform (DFT) will play an important role in the coming sections, so to make the presentation self-contained we will begin with some well-known definitions. The DFT of a vector[1] $\mathbf{x} \in \mathbb{R}^s$ will be denoted $\mathcal{F}(\mathbf{x})$. For convenience, we will also use a hat over a variable to denote its DFT, $\hat{\mathbf{x}} = \mathcal{F}(\mathbf{x})$. The elements of the resulting vector $\hat{\mathbf{x}}$ can be computed by

$$\hat{x}_j = \sum_{k=1}^{s} x_k \omega^{(j-1)(k-1)}, \qquad \text{with } \omega = e^{-2\pi \mathrm{i}/s}, \tag{3.6}$$

1 Although for our purposes $\mathbf{x}$ is always real-valued, the same properties hold for complex $\mathbf{x}$ as well.

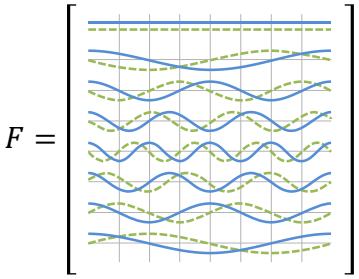


Figure 3.2: Illustration of the basis vectors of a DFT matrix (Eq. 3.8). The elements of each row contain a complex sinusoid of a distinct frequency. The solid blue lines depict the real part of the sinusoid, while the dashed green lines represent the imaginary part. Note that the first frequency, commonly referred to as the DC component, is constant, with its imaginary part equal to zero.

where i is the pure imaginary unit. From its definition in Eq. 3.6, we can see that ω is a root of unity, i.e. $\omega^s = 1$. This means that ω lies on the complex unit circle, and its powers in Eq. 3.6 must also lie on the complex unit circle.

Notice that the DFT operation is linear in the arguments ($\omega^{(j-1)(k-1)}$ are constants). It can be interpreted as a linear projection on a new basis, composed of complex sinusoids with varying frequencies. Because it is a linear operation, we can express it as a matrix-vector product,

$$\hat{\mathbf{x}} = F\mathbf{x}, \tag{3.7}$$

for an appropriately defined complex matrix $F \in \mathbb{C}^{s\times s}$, with elements

$$F_{jk} = \omega^{(j-1)(k-1)}. \tag{3.8}$$

F is called the DFT matrix; its rows are the complex sinusoid basis of the DFT. It can be easily verified to be symmetric, $F = F^T$. However, it is not Hermitian: $F \neq F^H$, where $F^H = (F^*)^T$ is the Hermitian transpose, and * denotes complex-conjugation.

3.4 UNITARITY

A matrix A is said to be unitary if $AA^H = I$, with I the identity matrix. A unitary matrix is the complex-valued extension of an orthogonal matrix, which represents a pure rotation in space when multiplied by a vector. As we will see, the DFT is analogous to a change of basis by rotation.

Unitarity is a very useful property, since it makes the inverse quite easily described in closed-form. Multiplying both sides of $AA^H = I$ by the inverse of A, we get $A^{-1} = A^H$. It also means that A preserves the L^2 norm of vectors, i.e. $\|A\mathbf{x}\|^2 = \|\mathbf{x}\|^2$, hence the analogy to a rotation (which also preserves Euclidean distances and thus L^2 norms).

We can verify by direct computation that F is *almost* unitary, up to a constant factor: $FF^H = sI$. Multiplying both sides by the inverse of F, we get $F^{-1} = \frac{1}{s}F^*$. So we can obtain the inverse DFT from the forward DFT, simply by complex-conjugation and rescaling with $\frac{1}{s}$.

Let us try to define a proper unitary DFT matrix U. We merely have to account for the constant factor that was mentioned before,

$$U = \tfrac{1}{\sqrt{s}}F. \tag{3.9}$$

Because $UU^H = \frac{1}{s}FF^H = I$, it is indeed unitary. As such, it preserves the L^2 norm $\|U\mathbf{x}\|^2 = \|\mathbf{x}\|^2$, and its inverse is given by $U^{-1} = U^H$. Since F and U are symmetric, we also have $U^{-1} = U^*$.

We can use these facts to obtain an expression for the norm of the non-unitary DFT. From Eq. 3.9 we have $\|U\mathbf{x}\|^2 = \left\|\frac{1}{\sqrt{s}}F\mathbf{x}\right\|^2 = \frac{1}{s}\|F\mathbf{x}\|^2$, and by substitution in $\|U\mathbf{x}\|^2 = \|\mathbf{x}\|^2$ it follows that $\|\hat{\mathbf{x}}\|^2 = s\|\mathbf{x}\|^2$. This is a fundamental property of the DFT, and is usually referred to in the literature as Parseval's Theorem [83].

3.5 THE CONVOLUTION THEOREM

One of the most direct applications of the DFT is in the fast computation of convolutions. The convolution theorem [83] states that the element-wise product of two vectors in the Fourier domain is equivalent to their convolution. There are several variations of the convolution theorem [83], including circular and non-circular variants, for discrete and continuous transforms. However, the variant that is more relevant to our problem setting is as follows.

Theorem 2 (Convolution theorem [83])**.** *Define the circular convolution of two vectors,* $\mathbf{x} * \mathbf{z}$*, as the vector with elements*

$$(\mathbf{x} * \mathbf{z})_j = \sum_{k=1}^{s} x_k \, z_{1+(j-k) \bmod s}. \tag{3.10}$$

Then it can also be computed in the Fourier domain, with

$$\mathbf{x} * \mathbf{z} = \mathcal{F}^{-1} \left(\hat{\mathbf{x}} \odot \hat{\mathbf{z}} \right), \tag{3.11}$$

where $\mathcal{F}^{-1}$ *is the inverse DFT, and* $\odot$ *denotes the element-wise product of two vectors.*

The modulus operation in Eq. 3.10 ensures that the index wraps around at s, making the convolution circular.

We can also express a circular convolution using a circulant matrix,

$$\mathbf{x} * \mathbf{z} = C^T (\mathbf{x}) \, \mathbf{z}, \tag{3.12}$$

which can be verified by expanding the right-hand-side and comparing it to Eq. 3.10. To avoid any confusion of notation, note that $C^T (\mathbf{x}) = (C (\mathbf{x}))^T$.

3.6 DIAGONALIZATION OF CIRCULANT MATRICES

Characterizing the eigenvalues and eigenvectors of a class of matrices is usually a good first step towards understanding its properties. In the case of circulant matrices, some of the most puzzling properties will become obvious if we are given the eigendecomposition beforehand.

Theorem 3. *The eigenvalues of a circulant matrix $C(\mathbf{x})$ are given by the DFT, $\hat{\mathbf{x}} = \mathcal{F}(\mathbf{x})$, and the eigenvectors by the unitary DFT matrix, U (Eq. 3.9). Equivalently,*

$$C(\mathbf{x}) = U\, diag(\hat{\mathbf{x}})\, U^*. \tag{3.13}$$

Proof. By the convolution theorem and Eq. 3.12,

$$C^T(\mathbf{x})\mathbf{z} = \mathbf{x} * \mathbf{z} = \mathcal{F}^{-1}(\hat{\mathbf{x}} \odot \hat{\mathbf{z}}). \tag{3.14}$$

Replace DFT and inverse DFT operations by their matrix-vector forms,

$$C^T(\mathbf{x})\mathbf{z} = \tfrac{1}{s} F^* (\hat{\mathbf{x}} \odot (F\mathbf{z})) \tag{3.15}$$

Because the element-wise product verifies $\mathbf{u} \odot \mathbf{v} = \mathrm{diag}(\mathbf{u})\,\mathbf{v}$ [95],

$$C^T(\mathbf{x})\mathbf{z} = \tfrac{1}{s} F^* \mathrm{diag}(\hat{\mathbf{x}})\, F\mathbf{z}. \tag{3.16}$$

Since Eq. 3.16 is valid for any $\mathbf{z}$, then it is also true that

$$C^T(\mathbf{x}) = \tfrac{1}{s} F^T \mathrm{diag}(\hat{\mathbf{x}})\, F. \tag{3.17}$$

By taking the transpose of both sides of the equation, and using the definition of U, we finally obtain

$$C(\mathbf{x}) = U \mathrm{diag}(\hat{\mathbf{x}})\, U^*. \tag{3.18}$$

□

The proof given here for Theorem 3 is considerably shorter than previously known derivations, which have resorted to difference equations [56] or polynomial representations of $C(\mathbf{x})$ [32]. In this proof, the same conclusions result from just the convolution theorem and some elementary matrix manipulations.

Theorem 3 shows that a circulant matrix $C(\mathbf{x})$ is diagonalized by the unitary DFT U, and its eigenvalues are simply the (non-unitary) DFT of the generating vector $\mathbf{x}$. Notice that the eigenvectors do not

depend on $\mathbf{x}$ at all, so every possible circulant matrix shares the same set of eigenvectors. This remarkable fact is the main reason for the somewhat uncommon properties that we will mention in the next section.

3.7 SOME PROPERTIES

We will now mention some useful properties of circulant matrices. They follow almost trivially from the eigendecomposition in Theorem 3, but we will make them explicit.

Consider two vectors $\mathbf{x}$, $\mathbf{z} \in \mathbb{R}^s$, with DFT $\hat{\mathbf{x}}$, $\hat{\mathbf{z}} \in \mathbb{C}^s$, respectively, and the circulant matrices $X = C(\mathbf{x})$ and $Z = C(\mathbf{z})$.

1. $XZ = ZX$. In words, any two circulant matrices X and Z commute. This is a consequence of them sharing the same eigenvectors.

2. $\alpha X + \beta Z = C(\alpha\mathbf{x} + \beta\mathbf{z})$. Circulant matrices are linear in the arguments.

3. $XZ = C\left(\mathcal{F}^{-1}\left(\hat{\mathbf{x}} \odot \hat{\mathbf{z}}\right)\right)$. The product of two circulant matrices is equivalent to an element-wise product in the Fourier domain.

4. $X\mathbf{z} = \mathcal{F}^{-1}\left(\hat{\mathbf{x}} \odot \hat{\mathbf{z}}\right)$. An analogous property holds for a matrix-vector product, but the result is a vector. This product can be understood as a cross-correlation operation.

5. If X is non-singular, $X^{-1} = C\left(\mathcal{F}^{-1}\left(1/\hat{\mathbf{x}}\right)\right)$, where division is taken element-wise. This means that matrix inversion is simply an element-wise inversion in the Fourier domain.

6. $X^T = C\left(\mathcal{F}^{-1}\left(\hat{\mathbf{x}}^*\right)\right)$. Matrix transposition is equivalent to complex-conjugation in the Fourier domain.

3.8 DERIVING A CORRELATION FILTER

Suppose that we want to learn a linear function using Ridge Regression (Section 2.1.1). Recall that the expression for the solution $\mathbf{w}$ is

$$\mathbf{w} = \left(X^T X + \lambda I\right)^{-1} X^T \mathbf{y}, \tag{3.19}$$

given a data matrix X and regression targets $\mathbf{y}$.

The samples may be composed of cyclic shifts of a base sample $\mathbf{x} \in \mathbb{R}^s$, so in this case we have a circulant data matrix $X = C(\mathbf{x})$.

We can apply the knowledge we gained in the previous sections to considerably simplify the computation. Notice that many operations would be expensive if performed with a naive algorithm (e.g. matrix products can scale with the cube of the matrix size, $\mathcal{O}(s^3)$). However, for circulant matrices, most of them can be performed element-wise in the Fourier domain (Section 3.7), bounding the complexity at $\mathcal{O}(s \log s)$ when using a Fast Fourier Transform algorithm [83].

Take the term $X^T X$, which can be seen as a non-centered covariance matrix. Replacing Eq. 3.13 in it,

$$X^T X = U \operatorname{diag}(\hat{\mathbf{x}}^*)\ U^* U \operatorname{diag}(\hat{\mathbf{x}})\ U^*. \tag{3.20}$$

Additionally, we can eliminate the factor $U^* U = I$, by the unitarity of U. We are left with

$$X^T X = U \operatorname{diag}(\hat{\mathbf{x}}^*)\ \operatorname{diag}(\hat{\mathbf{x}})\ U^*. \tag{3.21}$$

Because operations on diagonal matrices are element-wise, we can define the element-wise product as $\odot$ and obtain

$$X^T X = U \operatorname{diag}(\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}})\ U^*. \tag{3.22}$$

An interesting aspect is that the vector in brackets is known as the *auto-correlation* of the signal $\mathbf{x}$ (in the Fourier domain, also known as the power spectrum [83]). In classical signal processing, it contains the variance of a time-varying process for different time lags, or in our case, space.

The above steps summarize the general approach taken in diagonalizing expressions with circulant matrices. Applying them recursively to the full expression for linear regression (Eq. 3.19), we can put most quantities inside the diagonal,

$$\hat{\mathbf{w}} = \operatorname{diag}\left(\frac{\hat{\mathbf{x}}^*}{\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}} + \lambda}\right) \hat{\mathbf{y}}, \tag{3.23}$$

or better yet,

$$\hat{\mathbf{w}} = \frac{\hat{\mathbf{x}}^* \odot \hat{\mathbf{y}}}{\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}} + \lambda}. \tag{3.24}$$

The fraction denotes element-wise division. We can easily recover $\mathbf{w}$ in the spatial domain with the Inverse DFT, which has the same cost as a forward DFT. The detailed steps of the recursive diagonalization that yields Eq. 3.24 are given in Appendix A.1.5.

At this point we just found an unexpected formula from classical signal processing – the solution is a regularized correlation filter [11, 83].

Before exploring this relation further, we must highlight the computational efficiency of Eq. 3.24, compared to the prevalent method of extracting patches explicitly and solving a general regression problem. For example, Ridge Regression has a cost of $\mathcal{O}(s^3)$, bound by the matrix inversion and products[2]. On the other hand, all operations in Eq. 3.24 are element-wise ($\mathcal{O}(s)$), except for the DFT, which bounds the cost at a nearly-linear $\mathcal{O}(s \log s)$. For typical data sizes, this reduces storage and computation by several orders of magnitude.

Consequently, Eq. 3.24 yields a simple learning algorithm that can be used for tracking with very little engineering. In Section 4.7 we will compare it to several other algorithms, where it is referred to as "DCF on raw pixels". It achieves surprisingly competitive performance for such a small implementation effort and computational cost.

3.8.1 *Connection to classical signal processing*

Correlation filters have been a popular topic in signal processing since the 80's, with solutions to a myriad of objective functions in the Fourier domain [86, 83]. Recently, they made a reappearance as MOSSE filters [11], which have shown remarkable performance in tracking, despite their simplicity and high FPS rate.

The solution to these filters looks like Eq. 3.24 (see Appendix A.2), but with two crucial differences. First, MOSSE filters are derived from an objective function specifically formulated in the Fourier domain. Second, the λ regularizer is added in an ad-hoc way, to avoid division-by-zero. The derivation we showed above adds considerable insight, by specifying the starting point as Ridge Regression with cyclic shifts, and arriving at the same solution.

2 We remark that the complexity of training algorithms is usually reported in terms of the number of samples n, disregarding the number of features m. Since in our case $m = n$ (X is square), we conflate the two quantities. For comparison, the fastest SVM solvers have "linear" complexity in the samples $\mathcal{O}(mn)$, but under the same conditions $m = n$ would actually exhibit quadratic complexity, $\mathcal{O}(n^2)$.

Circulant matrices allow us to enrich the toolset put forward by classical signal processing and modern correlation filters, and apply the Fourier trick to new algorithms. Over the next chapter we will see one such instance, in training non-linear filters.

4

KERNELIZED CORRELATION FILTERS

Correlation filters can quickly find the linear model that provides the best fit to a desired correlation output, in the least-squares sense. They are an attractive algorithm for tracking, due to their good performance and high computational efficiency [11, 10], and have also found application in detection [86, 12]. It would be very desirable to apply the powerful kernel trick to correlation filters, so that this linear model would work on a rich high-dimensional space of non-linear features (Section 2.3). The kernel trick is known to generally improve the performance of linear algorithms [121, 105, 57, 138].

This problem has been attacked on several fronts [16, 103, 70, 146], but has resisted every effort – previous proposals resort to ad-hoc modifications of the objective function or suffer from an unsustainable computational penalty (Section 4.2.1).

However, the new derivation of correlation filters that was presented at the end of the last chapter (Section 3.8) presents an obvious path for applying the kernel trick. It begins with a Ridge Regression problem, where the samples are cyclic shifts, and arrives at the final formula for correlation filters by using the properties of circulant matrices. Because the starting point is standard Ridge Regression, the same steps can be followed for kernel Ridge Regression instead. In this way, we can easily obtain the kernelized version of correlation filters (which we named KCF). The main difficulty is dealing with the non-linearity of the kernel, which we do in Theorem 4. We also develop a linear version (Dual Correlation Filter, or DCF), which unlike previously proposed correlation filters, is able to incorporate several feature channels simultaneously. This enables the use of powerful features that result in significantly better performance.

Hopefully the value of kernelizing a successful linear algorithm for the first time is apparent, at least theoretically. On the other hand, this approach provides an answer to the sampling problem in tracking, which is of practical importance, and we will explore this point of view over the next section.

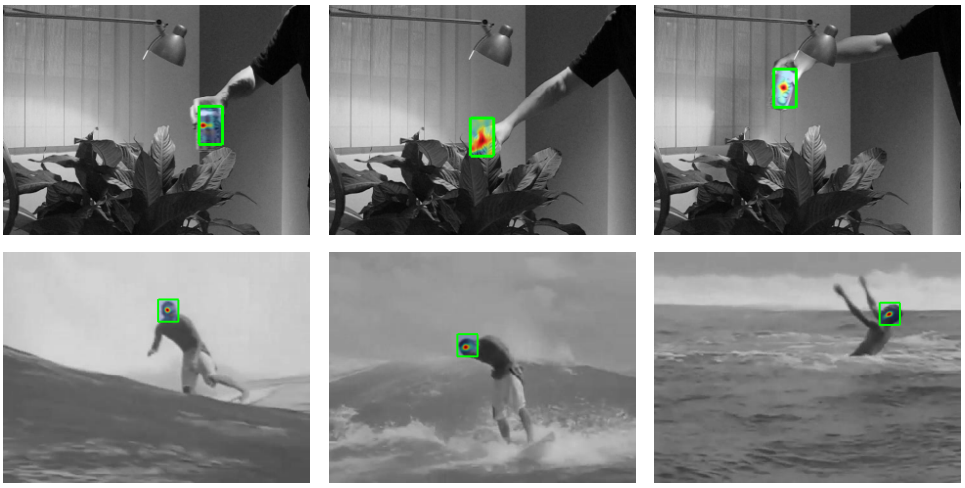


Figure 4.1: Example tracking results in three frames of two videos (`coke` and `surfer`). We show bounding boxes and response maps for Kernelized Correlation Filters on raw pixels. The response maps are color-coded with red/opaque for high responses, and blue/-transparent for low responses. The dataset used in the evaluation has 50 videos, shown in Fig. 4.3.

4.1 THE SAMPLING PROBLEM IN TRACKING

Arguably one of the biggest breakthroughs in recent visual tracking research was the widespread adoption of discriminative learning methods. The task of tracking, a crucial component of many computer vision systems, can be naturally specified as an online learning problem [127, 148]. Given an initial image patch containing the target, the goal is to learn a classifier to discriminate between its appearance and that of the environment. This classifier can be evaluated exhaustively at many locations, in order to detect it in subsequent frames. Of course, each new detection provides a new image patch that can be used to update the model.

It is tempting to focus on characterizing the object of interest – the positive samples for the classifier. However, a core tenet of discriminative methods is to give as much importance, or more, to the relevant environment – the negative samples. The most commonly used negative samples are image patches from different locations and scales, reflecting the prior knowledge that the classifier will be evaluated under those conditions.

An extremely challenging factor is the virtually unlimited amount of negative samples that can be obtained from an image. Due to the time-sensitive nature of tracking, modern trackers walk a fine line between incorporating as many samples as possible and keeping computational demand low. It is common practice to choose only a few

samples each frame, either randomly [151, 73, 6, 117, 57] or selected using structural constraints [73].

Although the reasons for doing so are understandable, our experiments support the hypothesis that undersampling negatives is the main factor inhibiting performance in tracking. Using the tools developed in Chapter 3, we can analytically incorporate thousands of samples at different relative translations, without iterating over them explicitly. This is made possible by the counterintuitive discovery that some learning problems may actually become *easier* as we add *more* samples, by acquiring circulant structure, which is easily exploited.

Using circulant matrices, we propose a tracker based on kernel Ridge Regression [112] that does not suffer from the "curse of kernelization", which is its larger asymptotic complexity, and even exhibits lower complexity than unstructured linear regression. We leverage the powerful kernel trick at the same computational complexity as linear correlation filters, which form the basis for the fastest trackers available [11, 10]. Our framework easily incorporates multiple feature channels, and by using a linear kernel we additionally show a fast extension of linear correlation filters to the multi-channel case.

4.2 RELATED WORK

Single-object tracking is a fundamental task in computer vision, with applications in video surveillance, human-machine interfaces and robot perception. Even though some settings allow for strong assumptions about the target [65, 150], single-object tracking usually consists of following a target given just its initial position and size in a video stream.

A comprehensive review is outside the scope of this chapter, but we refer the interested reader to two excellent and very recent surveys [127, 148]. It is possible to categorize current methods very broadly into two distinct approaches: generative and discriminative.

Generative tracking typically consists of building an appearance model to describe the object of interest, which can be used to search for it in a new frame of video. This search can also be guided by a probabilistic motion model [68]. However, the emphasis is on finding strong appearance models and distance metrics to compare them. A canonical example is Incremental Visual Tracking [114], which uses a low-dimensional subspace representation of the target to capture its variations over time, similar to Principal Components Analysis (PCA) [121] but learned online. Locally Orderless Tracking (LOT) [101] uses image superpixels as an appearance model, and compares them us-

ing a specialization of the Earth Mover's Distance, a natural metric to compare distributions. LOT uses a particle filter as a motion model to pursue different hypothesis simultaneously, which is the basis of Condensation [68], another canonical generative tracker. Additional examples of non-discriminative trackers include the work of Wu et al. [144], who formulate tracking as a sequence of image alignment objectives, and of Sevilla-Lara and Learned-Miller [122], who propose a strong appearance descriptor based on distribution fields.

Discriminative tracking has risen in popularity in recent years, following the success of machine learning methods in several areas of computer vision [147]. It consists of training a classifier online to directly predict the presence or absence of the target in an image patch [151, 73, 6, 117]. This classifier is then tested on many candidate patches to find the most likely location. Alternatively, the position and other properties of the target can also be predicted directly [57]. This discriminative approach to tracking is also called tracking-by-detection.

Canonical examples include those based on Support Vector Machines (SVM) [5], Random Forest classifiers [117], or boosting variants [54, 6]. All the mentioned algorithms had to be adapted for online learning, in order to be useful for tracking. Zhang et al. [151] propose a projection to a fixed random basis, to train a Naive Bayes classifier, inspired by compressive sensing techniques. Aiming to predict the target's location directly, instead of its presence in a given image patch, Hare et al. [57] employed a Structured Output SVM and Gaussian kernels, based on a large number of image features. Another discriminative approach by Kalal et al. [73] uses a set of structural constraints to guide the sampling process of a boosting classifier. Finally, Bolme et al. [11] employ classical signal processing analysis to derive fast correlation filters, which we generalize to multi-channel and non-linear filters.

4.2.1 *Kernel methods and correlation filters*

Classical correlation filters are usually formulated as variations of linear least-squares problems, but in the Fourier domain [11]. Least-squares problems are typically easy to extend with the kernel trick, obtaining more powerful non-linear models [121]. It would then seem that creating kernelized versions of correlation filters would be straightforward, but that is not the case.

Several recent works have analyzed this problem, but reported mostly negative results, showing that the standard Fourier metho-

Figure 4.2: Examples of vertical cyclic shifts of a base sample. Our Fourier domain formulation allows us to train a tracker with *all* possible cyclic shifts of a base sample, both vertical and horizontal, without iterating them explicitly. Artifacts from the wrapped-around edges can be seen (top of the left-most image), but are mitigated by the cosine window and padding.

dology cannot yield efficient algorithms [16, 103, 70]. This was found to be the case for objective functions that consider power spectrum or image translations, such as Minimum Average Correlation Energy [86], Optimal Trade-Off [146] and Minimum Output Sum of Squared Error (MOSSE) filters [11]. The main issue is that the application of the Discrete Fourier Transform (DFT) is done first, without a compelling theoretical justification, and the attempt to apply the kernel trick is done after-the-fact. Our formulation yields some insight on this problem, since the use of the DFT in a kernel algorithm arises as a direct consequence of the circulant structure, just as in the linear case.

Objective functions that ignore the spatial structure, such as Synthetic Discriminant Function (SDF) filters, are easier to kernelize [70, 103]. Unfortunately, ignoring the spatial structure effectively removes the correlation filters' greatest advantage, which is their robustness to image translations.

Another approach is to decouple the two features into separate stages: one that pre-processes the image using spatial information but ignoring the non-linear kernel, and another that applies the kernel trick ignoring the spatial structure [146]. A kernel that applies a whitening operation to the spectrum of its arguments was also proposed [9, Section 3.2.1]. These methods are notable and share some of the benefits of both kernels and correlation filters. However, they are somewhat ad-hoc and do not apply the kernel trick to the optimization objective of correlation filters, which we do on a strong theoretical basis by virtue of the cyclic shifted samples.

4.3 FAST KERNEL REGRESSION

We can now focus on the main derivations of this chapter. They consist of replicating the process done in Section 3.8, which derives a correlation filter from Ridge Regression and cyclically shifted samples, but now for kernel Ridge Regression. After computing the optimal solution in the dual, as is usually done for kernel algorithms (Section 2.3), we will also look at some related computations and how they can be accelerated using the properties of circulant matrices (Sections 4.4 and 4.5).

Let us recall the solution for dual Ridge Regression (Eq. 2.16). It can be computed in closed-form with

$$\boldsymbol{\alpha} = (K + \lambda I)^{-1} \mathbf{y}, \tag{4.1}$$

where K is the kernel matrix and $\boldsymbol{\alpha}$ is the vector of coefficients α_i, that represent the solution in the dual space. Since we want to work in a high-dimensional feature space induced by a kernel function κ, we use it to compute the elements of the kernel matrix, $K_{ij} = \kappa\left(\mathbf{x}'_i, \mathbf{x}'_j\right)$ (Eq. 2.24). We also use the term kernel matrix instead of Gram matrix, and denote it with K, to emphasize the fact that the standard dot-product is computed implicitly by a kernel function.

Now, if we can prove that K is circulant for datasets of cyclic shifts, we can diagonalize Eq. 4.1 and obtain a fast solution as for the linear case. This would seem to be intuitively true, but does not hold in general. The arbitrary non-linear mapping gives us no guarantee of preserving any sort of structure. However, we can impose one condition that will allow K to be circulant. It turns out to be fairly broad, and apply to most useful kernels.

Theorem 4. *Given circulant data $C(\mathbf{x})$, the corresponding kernel matrix K is circulant if the kernel function satisfies $\kappa(\mathbf{x}, \mathbf{x}') = \kappa(M\mathbf{x}, M\mathbf{x}')$, for any permutation matrix M.*

For a proof, see Appendix A.1.2. What this means is that, for a kernel to preserve the circulant structure, it must treat all dimensions of the data equally. Fortunately, this includes most useful kernels (Section 2.3.2).

Example 5. *The following kernels (see Section 2.3.2) satisfy Theorem 4:*

- *Radial Basis Function kernels – e.g., Gaussian.*
- *Dot-product kernels – e.g., linear, polynomial.*

- *Additive kernels – e.g., intersection, χ^2 and Hellinger kernels [138].*
- *Exponentiated additive kernels.*

Checking this fact is easy, since reordering the dimensions of $\mathbf{x}$ and $\mathbf{x}'$ simultaneously does not change $\kappa(\mathbf{x}, \mathbf{x}')$ for these kernels. This applies to any kernel that combines dimensions through a commutative operation, such as sum, product, min and max.

Knowing which kernels we can use to make K circulant, it is possible to diagonalize Eq. 4.1 as in the linear case, obtaining

$$\hat{\boldsymbol{\alpha}} = \frac{\hat{\mathbf{y}}}{\hat{\mathbf{k}}^{\mathbf{xx}} + \lambda}, \tag{4.2}$$

where $\mathbf{k}^{\mathbf{xx}}$ is the first row of the kernel matrix $K = C(\mathbf{k}^{\mathbf{xx}})$, and a hat $\hat{}$ denotes the DFT of a vector. A detailed derivation is in Appendix A.1.3.

To better understand the role of $\mathbf{k}^{\mathbf{xx}}$, we found it useful to define a more general *kernel correlation*. The kernel correlation of two arbitrary vectors, $\mathbf{x}$ and $\mathbf{x}'$, is the vector $\mathbf{k}^{\mathbf{xx}'}$ with elements

$$k_i^{\mathbf{xx}'} = \kappa(\mathbf{x}', P^{i-1}\mathbf{x}). \tag{4.3}$$

Recall that P is the cyclic shift operator (Eq. 3.2), which is a permutation matrix. In words, kernel correlation evaluates the kernel for different relative shifts of its two arguments, and stores them in a vector. Then $\hat{\mathbf{k}}^{\mathbf{xx}}$ is the kernel correlation of $\mathbf{x}$ with itself, in the Fourier domain. We can refer to it as the *kernel auto-correlation*, in analogy with the linear case.

This analogy can be taken further. Since a kernel is equivalent to a dot-product in a high-dimensional space $\varphi(\cdot)$, another way to view Eq. 4.3 is

$$k_i^{\mathbf{xx}'} = \varphi^T(\mathbf{x}')\varphi(P^{i-1}\mathbf{x}), \tag{4.4}$$

which is the cross-correlation of $\mathbf{x}$ and $\mathbf{x}'$ in the high-dimensional space $\varphi(\cdot)$.

Notice how we only need to compute and operate on the kernel auto-correlation, an $s \times 1$ vector, which grows linearly with the number of samples. This is contrary to the conventional wisdom on kernel methods, which requires computing an $s \times s$ kernel matrix, scaling quadratically with the samples. Our knowledge of the exact structure of K allowed us to do better than a generic algorithm.

Finding the optimal $\boldsymbol{\alpha}$ is not the only problem that can be accelerated, due to the ubiquity of translated patches in a tracking-by-

detection setting. Over the next sections we will investigate the effect of the cyclic shift model on the detection phase, and even in computing kernel correlations.

4.4 FAST DETECTION

It is rarely the case that we want to evaluate the regression function $f(\mathbf{z})$ for one image patch in isolation. To detect the object of interest, we typically wish to evaluate $f(\mathbf{z})$ on several image locations, i.e., for several candidate patches. These patches can be modeled by cyclic shifts.

Denote by $K^{\mathbf{z}}$ the (asymmetric) kernel matrix between all training samples and all candidate patches. Since the samples and patches are cyclic shifts of base sample $\mathbf{x}$ and base patch $\mathbf{z}$, respectively, each element of $K^{\mathbf{z}}$ is given by $\kappa(P^{i-1}\mathbf{z},\ P^{j-1}\mathbf{x})$. It is easy to verify that this kernel matrix satisfies Theorem 4, and is circulant for appropriate kernels.

Similarly to Section 4.3, we only need the first row to define the kernel matrix:

$$K^{\mathbf{z}} = C(\mathbf{k}^{\mathbf{xz}}), \tag{4.5}$$

where $\mathbf{k}^{\mathbf{xz}}$ is the *kernel correlation* of $\mathbf{x}$ and $\mathbf{z}$, as defined before.

From Eq. 2.25, we can compute the regression function for all candidate patches with

$$\mathbf{f}(\mathbf{z}) = \left(K^{\mathbf{z}}\right)^T \boldsymbol{\alpha}. \tag{4.6}$$

Notice that $\mathbf{f}(\mathbf{z})$ is a vector, containing the output for *all* cyclic shifts of $\mathbf{z}$, i.e., the full detection response. To compute Eq. 4.6 efficiently, we diagonalize it to obtain

$$\hat{\mathbf{f}}(\mathbf{z}) = \hat{\mathbf{k}}^{\mathbf{xz}} \odot \hat{\boldsymbol{\alpha}}. \tag{4.7}$$

Intuitively, evaluating $f(\mathbf{z})$ at all locations can be seen as a spatial filtering operation over the kernel values $\mathbf{k}^{\mathbf{xz}}$. Each $f(\mathbf{z})$ is a linear combination of the neighboring kernel values from $\mathbf{k}^{\mathbf{xz}}$, weighted by the learned coefficients $\boldsymbol{\alpha}$. Since this is a filtering operation, it can be formulated more efficiently in the Fourier domain. A more detailed proof is given in Appendix A.1.4.

4.5 FAST KERNEL CORRELATION

Even though we have found faster algorithms for training and detection, they still rely on computing one kernel correlation each ($\mathbf{k}^{\mathbf{xx}}$ and $\mathbf{k}^{\mathbf{xz}}$, respectively). Recall that kernel correlation consists of computing the kernel for all relative shifts of two input vectors (Eq. 4.3). This represents the last standing computational bottleneck, as a naive evaluation of s kernels for signals of size s will have quadratic complexity. However, using the cyclic shift model will allow us to efficiently exploit the redundancies in this expensive computation.

4.5.1 *Dot-product and polynomial kernels*

Dot-product kernels have the form $\kappa(\mathbf{x}, \mathbf{x}') = g(\mathbf{x}^T\mathbf{x}')$, for some function g (Section 2.3.2). Then, $\mathbf{k}^{\mathbf{xx}'}$ has elements

$$k_i^{\mathbf{xx}'} = \kappa(\mathbf{x}', P^{i-1}\mathbf{x}) = g\big(\mathbf{x}'^T P^{i-1}\mathbf{x}\big)\,. \tag{4.8}$$

Let g also work element-wise on any input vector. This way we can write Eq. 4.8 in vector form

$$\mathbf{k}^{\mathbf{xx}'} = g(C(\mathbf{x})\,\mathbf{x}')\,. \tag{4.9}$$

This makes it an easy target for diagonalization, yielding

$$\mathbf{k}^{\mathbf{xx}'} = g\big(\mathcal{F}^{-1}\left(\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}}'\right)\big)\,, \tag{4.10}$$

where $\mathcal{F}^{-1}$ denotes the Inverse DFT.

In particular, for a polynomial kernel $\kappa(\mathbf{x}, \mathbf{x}') = \big(\mathbf{x}^T\mathbf{x}' + a\big)^b$,

$$\mathbf{k}^{\mathbf{xx}'} = \big(\mathcal{F}^{-1}\left(\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}}'\right) + a\big)^b\,. \tag{4.11}$$

Then, computing the kernel correlation for these particular kernels can be done using only a few DFT/IDFT and element-wise operations, in $\mathcal{O}\left(s \log s\right)$ time.

4.5.2 *Radial Basis Function and Gaussian kernels*

RBF kernels have the form $\kappa(\mathbf{x}, \mathbf{x}') = h(\|\mathbf{x} - \mathbf{x}'\|^2)$, for some function h (Section 2.3.2). The elements of $\mathbf{k}^{\mathbf{xx}'}$ are

$$k_i^{\mathbf{xx}'} = \kappa(\mathbf{x}', P^{i-1}\mathbf{x}) = h\Big(\big\|\mathbf{x}' - P^{i-1}\mathbf{x}\big\|^2\Big) \tag{4.12}$$

We will show (Eq. 4.14) that this is actually a special case of a dot-product kernel. We only have to expand the norm,

$$k_i^{\mathbf{x}\mathbf{x}'} = h\Big(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\mathbf{x}'^T P^{i-1}\mathbf{x}\Big)\,. \tag{4.13}$$

The permutation P^{i-1} does not affect the norm of $\mathbf{x}$ due to Parseval's Theorem [83]. Since $\|\mathbf{x}\|^2$ and $\|\mathbf{x}'\|^2$ are constant w.r.t. i, Eq. 4.13 has the same form as a dot-product kernel (Eq. 4.8). Leveraging the result from the previous section,

$$\mathbf{k}^{\mathbf{x}\mathbf{x}'} = h\Big(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\mathcal{F}^{-1}\left(\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}}'\right)\Big)\,. \tag{4.14}$$

As a particularly useful special case, for a Gaussian kernel $\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{\sigma^2}\|\mathbf{x} - \mathbf{x}'\|^2\right)$ we get

$$\mathbf{k}^{\mathbf{x}\mathbf{x}'} = \exp\left(-\frac{1}{\sigma^2}\left(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\mathcal{F}^{-1}\left(\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}}'\right)\right)\right)\,. \tag{4.15}$$

As before, we can compute kernel correlation in only $\mathcal{O}\left(s \log s\right)$ time.

4.5.3 *Other kernels*

The approach from the preceding two sections depends on the kernel value being unchanged by unitary transformations, such as the DFT. This does not hold in general for other kernels, e.g. intersection kernel. We can still use the fast training and detection results (Sections 4.3 and 4.4), but kernel correlation must be evaluated by a more expensive sliding window method.

4.6 MULTIPLE CHANNELS

In this section, we will see that working in the dual has the advantage of allowing multiple channels (such as the orientation bins of a HOG descriptor [44]) by simply summing over them in the Fourier domain. A corollary is that linear correlation filter can also easily incorporate multiple channels, if solved in the dual.

4.6.1 *General case*

To deal with multiple channels, in this section we will assume that a vector $\mathbf{x}$ concatenates the individual vectors for m channels (e.g. 31 gradient orientation bins for a HOG variant [44]), as $\mathbf{x} = [\mathbf{x}_1, \dots, \mathbf{x}_m]$.

Notice that all kernels studied in Section 4.5 are based on either dot-products or norms of the arguments. A dot-product can be computed by simply summing the individual dot-products for each channel. By linearity of the DFT, this allows us to sum the result for each channel in the Fourier domain. As a concrete example, we can apply this reasoning to the Gaussian kernel, obtaining the multi-channel analogue of Eq. 4.15,

$$\mathbf{k}^{\mathbf{x}\mathbf{x}'} = \exp\left(-\frac{1}{\sigma^2}\left(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\mathcal{F}^{-1}\left(\textstyle\sum_c^m \hat{\mathbf{x}}_c^* \odot \hat{\mathbf{x}}'_c\right)\right)\right). \tag{4.16}$$

It is worth emphasizing that the integration of multiple channels does not result in a more difficult inference problem – we merely have to sum over the channels when computing kernel correlation.

4.6.2 *Linear kernel*

For a linear kernel $\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$, the multi-channel extension from the previous section simply yields

$$\mathbf{k}^{\mathbf{x}\mathbf{x}'} = \mathcal{F}^{-1}\left(\textstyle\sum_c^m \hat{\mathbf{x}}_c^* \odot \hat{\mathbf{x}}'_c\right). \tag{4.17}$$

We named it the Dual Correlation Filter (DCF). This filter is linear, but trained in the dual space $\boldsymbol{\alpha}$. This enables a simple multi-channel extension, that direct formulations such as MOSSE lack [11].

4.7 EXPERIMENTS

4.7.1 *Tracking pipeline*

We implemented in Matlab two simple trackers based on the proposed Kernelized Correlation Filter (KCF), using a Gaussian kernel, and Dual Correlation Filter (DCF), using a linear kernel. We do not report results for a polynomial kernel as they are virtually identical to those for the Gaussian kernel, and require more parameters. We tested two further variants: one that works directly on the raw pixel values, and another that works on HOG descriptors with a cell size of

Algorithm 4.1 Matlab code for a Kernelized Correlation Filter, using a Gaussian kernel. Multiple channels (third dimension of image patches) are supported. It is possible to further reduce the number of FFT calls. Implementation with GUI available at:
`http://www.isr.uc.pt/~henriques/`

Inputs:

- x: training image patch, size $s_1 \times s_2 \times m$
  (an RGB image or a grid of features such as HOG)
- y: regression target, Gaussian-shaped, size $s_1 \times s_2$
- z: test image patch, size $s_1 \times s_2 \times m$

Output:

- `responses`: detection score for each location, $s_1 \times s_2$

```
function alphaf = train(x, y, sigma, lambda)
  k = kernel_correlation(x, x, sigma);
  alphaf = fft2(y) ./ (fft2(k) + lambda);
end

function responses = detect(alphaf, x, z, sigma)
  k = kernel_correlation(z, x, sigma);
  responses = real(ifft2(alphaf .* fft2(k)));
end

function k = kernel_correlation(x1, x2, sigma)
  c = ifft2(sum(conj(fft2(x1)) .* fft2(x2), 3));
  d = x1(:)'*x1(:) + x2(:)'*x2(:) - 2 * c;
  k = exp(-1 / sigma^2 * abs(d) / numel(d));
end
```

Figure 4.3: Qualitative results for the proposed Kernelized Correlation Filter (KCF), compared with the top-performing Struck and TLD. Best viewed on a high-resolution screen. The chosen kernel is Gaussian, on HOG features. These snapshots were taken at the midpoints of the 50 videos of a recent benchmark [147]. Missing trackers are denoted by an "x". KCF outperforms both Struck and TLD, despite its minimal implementation and running at 172 FPS (see Algorithm 4.1, and Table 4.2).

PARAMETERS	RAW PIXELS	HOG
Feature bandwidth σ	0.2	0.5
Adaptation rate	0.075	0.02
Spatial bandwidth δ	$\sqrt{s_1 s_2}/10$	
Regularization λ	10^{-4}	

Table 4.1: Parameters used in all tracking experiments. The same parameters were used for KCF and for DCF. The parameters differ only between the raw pixels and HOG variants. s_1 and s_2 refer to the height and width of the target, measured in pixels or HOG cells.

4 pixels, in particular Felzenszwalb's variant[1] [44, 36]. Note that our linear DCF is equivalent to MOSSE [11] in the limiting case of a single channel (raw pixels), but it has the advantage of also supporting multiple channels (e.g. HOG). Our tracker requires few parameters, and we report the values that we used, fixed for all videos, in Table 4.1.

The bulk of the functionality of the KCF is presented as Matlab code in Algorithm 4.1. It is prepared to deal with multiple channels, as the 3rd dimension of the input arrays. It implements 3 functions: `train` (Eq. 4.2), `detect` (Eq. 4.7), and `kernel_correlation` (Eq. 4.16), which is used by the first two functions.

The pipeline for the tracker is intentionally simple, and does not include any heuristics for failure detection or motion modeling. In the first frame, we `train` a model with the image patch at the initial position of the target. This patch is larger than the target, to provide some context. For each new frame, we `detect` over the patch at the previous position, and the target position is updated to the one that yielded the maximum value. Finally, we `train` a new model at the new position, and linearly interpolate the obtained values of $\boldsymbol{\alpha}$ and $\mathbf{x}$ with the ones from the previous frame, to provide the tracker with some memory.

4.7.2 *Evaluation*

We put our tracker to the test by using a recent benchmark that includes 50 video sequences [147] (see Fig. 4.3). This dataset collects many videos used in previous works, so we avoid the danger of overfitting to a small subset.

1 The HOG descriptor for each cell contains 31 features, corresponding to the normalized edge magnitude in 9 contrast-insensitive and 18 contrast-sensitive orientations, plus 4 gradient energy features [44]. The features were computed using Piotr's Toolbox [35].

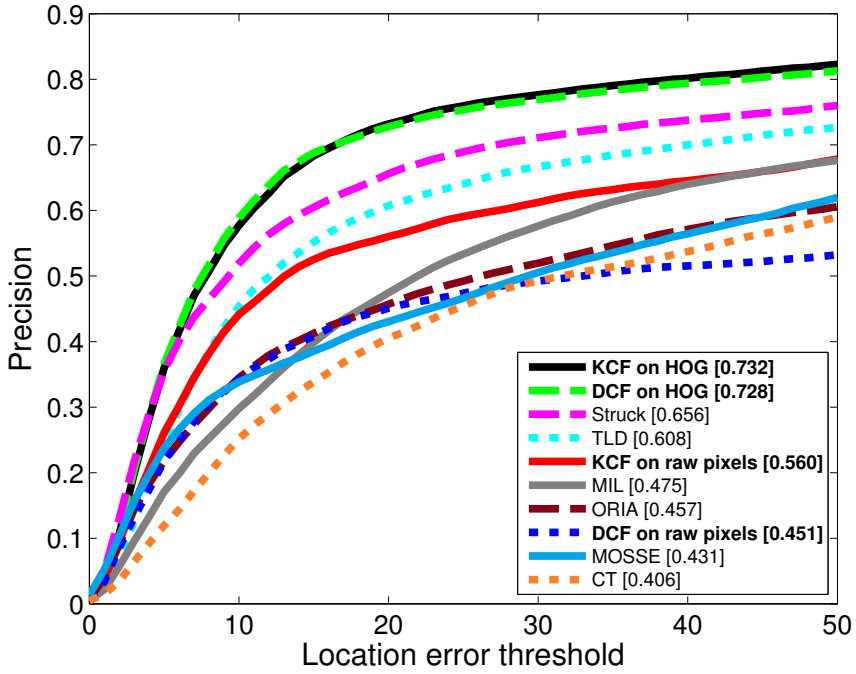


Figure 4.4: Precision plot for all 50 tracking sequences. The proposed trackers (bold) outperform state-of-the-art systems, such as TLD and Struck, which are more complicated to implement and much slower (see Table 4.2). Best viewed in color.

For the performance criteria, we did not choose average location error or other measures that are averaged over frames, since they impose an arbitrary penalty on lost trackers that depends on chance factors (i.e., the position where the track was lost), making them not comparable. A similar alternative is bounding box overlap, which has the disadvantage of heavily penalizing trackers that do not track across scale, even if the target position is otherwise tracked perfectly.

An increasingly popular alternative, which we chose for our evaluation, is the precision curve [147, 6, 61]. A frame may be considered correctly tracked if the predicted target center is within a distance threshold of ground truth. Precision curves simply show the percentage of correctly tracked frames for a range of distance thresholds. Notice that by plotting the precision for all thresholds, no parameters are required. This makes the curves unambiguous and easy to interpret. A higher precision at low thresholds means the tracker is more accurate, while a lost target will prevent it from achieving perfect precision for a very large threshold range. When a representative precision score is needed, the chosen threshold is 20 pixels, as done in previous works [147, 6, 61].

	ALGORITHM	FEATURES	PRECISION	MEAN FPS
Proposed	KCF	HOG	**73.2%**	172
	DCF		**72.8%**	**292**
	KCF	Pixels	56.0%	154
	DCF		45.1%	278
Other algorithms	Struck [57]		65.6%	20
	TLD [73]		60.8%	28
	MOSSE [11]		43.1%	**615**
	MIL [6]		47.5%	38
	ORIA [144]		45.7%	9
	CT [151]		40.6%	64

Table 4.2: Summary of experimental results on the 50 videos dataset [147]. The reported quantities are averaged over all videos. Precision is calculated at a 20px threshold. Reported speeds include feature computation (e.g. HOG).

4.7.3 *Experiments on the full dataset*

We start by summarizing the results over all videos in Table 4.2 and Fig. 4.4. For comparison, we also report results for several other systems [57, 73, 11, 6, 144, 151], including some of the most resilient trackers available – namely, Struck and TLD. Unlike our simplistic implementation (Algorithm 4.1), these trackers contain numerous engineering improvements. Struck operates on many different kinds of features and a growing pool of support vectors. TLD is specifically geared towards re-detection, using a set of structural rules with many parameters.

Despite this asymmetry, our Kernelized Correlation Filter (KCF) can already reach competitive performance by operating on raw pixels alone, as can be seen in Fig. 4.4. In this setting, the rich implicit features induced by the Gaussian kernel yield a distinct advantage over the proposed Dual Correlation Filter (DCF).

We remark that the DCF with single-channel features (raw pixels) is theoretically equivalent to a MOSSE filter [11]. For a direct comparison, we include the results for the authors' MOSSE tracker [11] in Fig. 4.4. The performance of both is very close, showing that any particular differences in their implementations do not seem to matter much. However, the kernelized algorithm we propose (KCF) does yield a noticeable increase in performance.

Replacing pixels with HOG features allows the KCF and DCF to surpass even TLD and Struck, by a relatively large margin (Fig. 4.4). This suggests that the most crucial factor for high performance, com-

pared to other trackers that use similar features, is the efficient incorporation of thousands of negative samples from the target's environment, which they do with very little overhead.

4.7.4 *Timing*

As mentioned earlier, the overall complexity of our closed-form solutions is $\mathcal{O}(s \log s)$, resulting in its high speed (Table 4.2). The speed of the tracker is directly related to the size of the tracked region. This is an important factor when comparing trackers based on correlation filters. MOSSE [11] tracks a region that has the same support as the target object, while our implementation tracks a region that is 2.5 times larger (116x170 pixels on average). Reducing the tracked region would allow us to approach their FPS of 615 (Table 4.2), but we found that it hurts performance, especially for the kernel variants. Another interesting observation from Table 4.2 is that operating on 31 HOG features per spatial cell can be slightly faster than operating on raw pixels, even though we take the overhead of computing HOG features into account. Since each 4x4 pixels cell is represented by a single HOG descriptor, the smaller-sized DFT counterbalance the cost of iterating over feature channels. Taking advantage of all 4 cores of a desktop computer, KCF/DCF take less than 2 minutes to process all 50 videos ($\sim$29,000 frames).

4.7.5 *Experiments with sequence attributes*

The videos in the benchmark dataset [147] are annotated with attributes, which describe the challenges that a tracker will face in each sequence – e.g., illumination changes or occlusions. These attributes are useful for diagnosing and characterizing the behavior of trackers in such a large dataset, without having to analyze each individual video. We report results for 4 attributes in Figure 4.5: non-rigid deformations, occlusions, out-of-view target, and background clutter.

The robustness of the HOG variants of our tracker regarding non-rigid deformations and occlusions is not surprising, since these features are known to be highly discriminative [44]. However, the KCF on raw pixels alone still fares almost as well as Struck and TLD, with the kernel making up for the features' shortcomings.

One challenge for the system we implemented is an out-of-view target, due to the lack of a failure recovery mechanism. TLD performs better than most other trackers in this case, which illustrates its focus on re-detection and failure recovery. Such engineering improvements

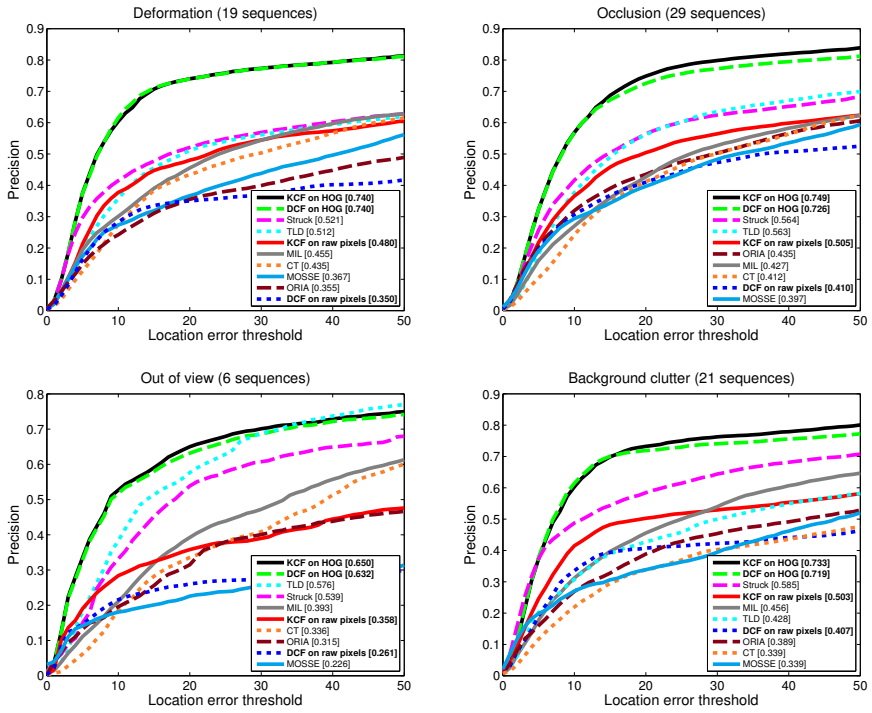


Figure 4.5: Precision plot for sequences with attributes: occlusion, non-rigid deformation, out-of-view target, and background clutter. The HOG variants of the proposed trackers (bold) are the most resilient to all of these nuisances. Best viewed in color.

could probably benefit our trackers, but the fact that KCF/DCF can still outperform TLD shows that they are not a decisive factor.

Background clutter severely affects almost all trackers, except for the proposed ones, and to a lesser degree, Struck. For our tracker variants, this is explained by the implicit inclusion of thousands of negative samples around the tracked object. Since in this case even the raw pixel variants of our tracker have a performance very close to optimal, while TLD, CT, ORIA and MIL show degraded performance, we conjecture that this is caused by their undersampling of negatives.

We also report results for other attributes in Fig. 4.6. Generally, the proposed trackers are the most robust to 6 of the 7 challenges, except for low resolution, which affects equally all trackers but Struck.

4.8 CONCLUSIONS

In this chapter, we leveraged the theory of circulant matrices to efficiently train a kernel learning algorithm with several translated image patches. Our approach provides a theoretically sound solution to the long-standing open problem of how to apply the kernel trick to correlation filters. We thus obtained state-of-the-art trackers that run

at hundreds of FPS, and can be implemented with only a few lines of code.

The proposed method takes into account cyclic shifts of a single base sample, which seems to be sufficient to obtain very competitive performance in tracking. It is possible to extend the method to incorporate cyclic shifts of a set of base samples. Unfortunately, the resulting model would scale linearly with the number of base samples, similarly to other kernel methods [121]. This fact can be mitigated by a number of approaches, such as the Nyström method [143], random Fourier features [108], and explicit feature maps obtained by spectral analysis [138]. Nevertheless, for large-scale learning applications, such as detection and pose estimation, linear models are typically chosen over non-linear kernels. For this reason, over the next chapters we will analyze the behavior of batch learning algorithms under cyclic shifts, and change our focus to linear models.

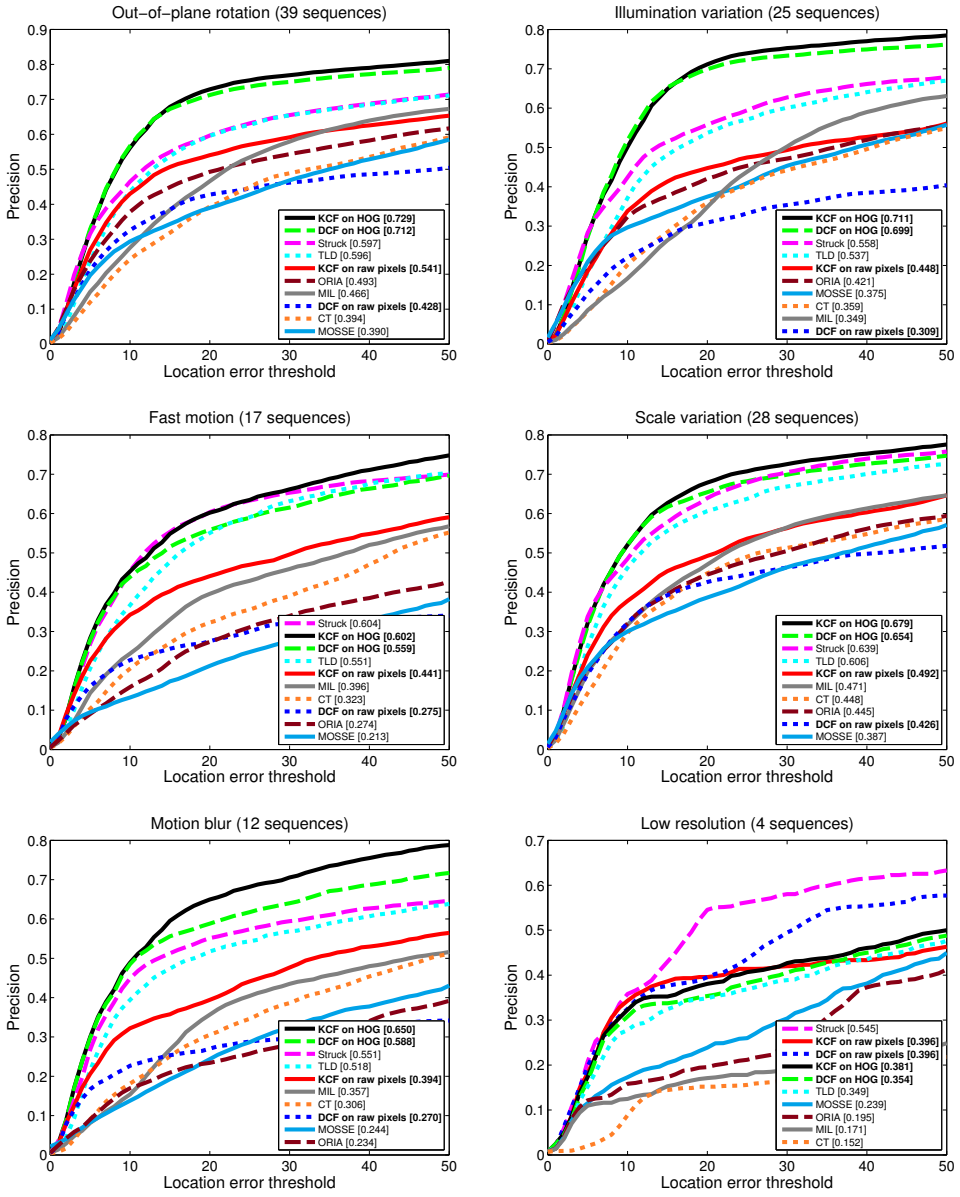


Figure 4.6: Precision plots for 6 attributes of the dataset. Best viewed in color. In-plane rotation was left out due to space constraints. Its results are virtually identical to those for out-of-plane rotation (top-left), since they share almost the same set of sequences.

5

CIRCULANT DECOMPOSITION OF LARGE-SCALE PROBLEMS

There are a few immediate ways in which we can generalize the methodology of the previous chapters. One of them is to consider the cyclic shifts of multiple base samples. In tracking applications (studied in Chapter 4) there is only one new image in each frame, from which we can obtain samples through cyclic shifts. In this chapter, however, we will study detectors for general classes of objects. Object detectors are typically learned offline, using a collection of example images, and all of their cyclic shifts are potentially useful samples, which we must consider jointly. In Section 5.1 we will relate this sampling method to the approach taken in state-of-the-art detector training, called hard-negative mining. From Section 5.2.1 onwards we will devise efficient algorithms that exploit the proposed sampling process.

Another important generalization is to consider more complicated learning algorithms than Ridge Regression, which often attain better performance [30, 44]. The discussion in Chapter 4 was restricted to Ridge Regression, which minimizes the squared error of the model on the training samples (Section 2.1.1). Both Ridge Regression, and other objective functions that are quadratic in the arguments, can be solved in closed-form [14]. This makes them a popular framework in the signal processing literature, and the basis for correlation filters (by minimizing a quadratic error in the Fourier domain) [86, 10, 146]. It is legitimate to ask, however, whether correlation filters can be generalized using more varied loss functions, creating a bridge between classical signal processing and machine learning algorithms. The practical benefit is a family of algorithms that inherit the computational efficiency of the former, and the high performance of the later. These advantages will be demonstrated with detector learning experiments in Section 5.6.

5.1 THE SAMPLING PROBLEM IN DETECTION

Similarly to object trackers, object detectors work by scanning images using learned templates. These templates may model whole objects [30], parts [44], more general mid-level fragments [13, 135] or hierar-

chies of filters capturing increasingly higher-level features [76]. The templates, most often HOG (Histogram of Oriented Gradients) filters [30], are evaluated exhaustively at all locations in an image over a discrete range of scales, using fast convolution implementations.

This type of dense search is very powerful but makes it challenging to learn the filters: learning algorithms for even the simplest models, linear classification and logistic regression, scale to $\sim 10^6$ training examples [42]; however, a handful of test images can contain these many samples. The asymmetry between the resolution of prediction and learning has been tackled by mining for hard negative examples. In this iterative process, an initial model is trained using all positive examples (which are representative of the desired object class) and a randomly selected subset of negative examples. The initial training set is then progressively augmented with false positive examples produced while scanning the images with the model learned so far. Due to the exhaustive scanning, hard negative mining is very expensive, and does not scale up to a large number of object models [88].

Some recent attempts at speeding up learning have fit parametric models (Gaussians) to the background distribution [102, 58, 49], however some of these approaches are specific to particular models such as Linear Discriminant Analysis. Additionally, natural image statistics are known to be characterized by long exponential tails [115] hence may not be very faithfully represented by a Gaussian distribution.

Fourier transforms have long been used to perform fast convolution, and were employed recently to accelerate detectors at test time [38]. They have also been used to accelerate the subgradient computation of a modified SVM solver [39].

In the present chapter we will take a different direction. Leveraging the idea of cyclic shifts (Chapter 3), we will model *all* samples that are contained in the training images. We will then see how standard learning algorithms can take advantage of the inherent structure for maximum efficiency. By taking into account all possible samples from the start, there is no need for an expensive hard-negative mining process.

5.2 DATASETS WITH CYCLIC SHIFTS

Our starting point is a direct extension of the datasets considered in previous chapters. Recall that s virtual samples can be obtained by translation from a base sample $\mathbf{x}$ as (Eq. 3.4), yielding the dataset

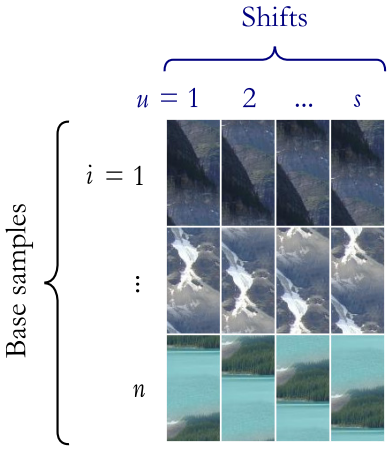


Figure 5.1: Illustration of a dataset augmented using s cyclic shifts of the original n base samples. Such a dataset can be used to model the training of a classifier with several subwindows from a set of images.

$$\mathcal{X}' = \left\{ P^{u-1}\mathbf{x} \,|\, u = 1, \ldots s \right\}. \tag{5.1}$$

The samples can be concatenated into the rows of a data matrix, which will then be circulant: $X' = C(\mathbf{x})$ (Section 3.2).

Now, if we consider n base samples instead, denoted $\mathbf{x}_i$, we get an augmented dataset

$$\mathcal{X} = \left\{ P^{u-1}\mathbf{x}_i \,|\, i = 1, \ldots, n;\ u = 1, \ldots, s \right\}. \tag{5.2}$$

The total number of samples is now sn, which can exceed the base samples by a large factor. Another view of the same dataset is by concatenating the samples into the rows of a $sn \times s$ data matrix,

$$X = \begin{bmatrix} C(\mathbf{x}_1) \\ \vdots \\ C(\mathbf{x}_n) \end{bmatrix} \tag{5.3}$$

which contains one circulant block for each base sample.

The model of Eq. 5.2-5.3, if exploited correctly, can have a tremendous impact in practice. The bottleneck of detector learning is the large amount of potentially useful negative samples, extracted from example images by hard-negative mining. Eq. 5.2, on the other hand, can describe all the potential negative samples at once. Because of the wrap-around effect at the borders, the approximation quality of cyclic shifts degrades for very large translations, when compared to non-cyclic translation. As such, in practice we collect base samples in a grid pattern, at large intervals, which models large translations.

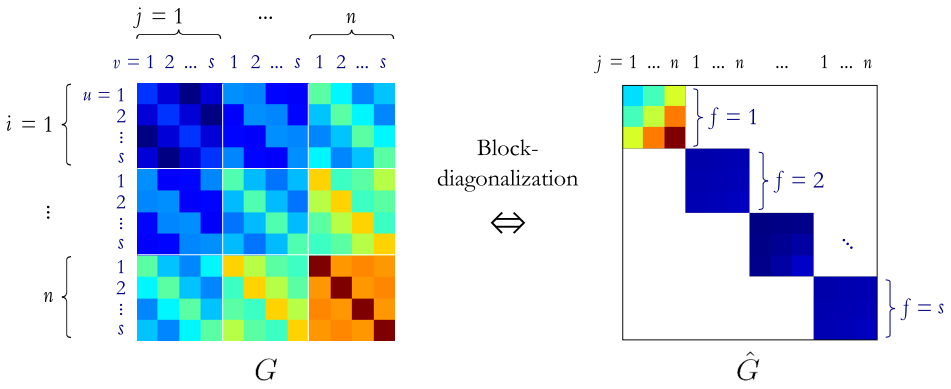


Figure 5.2: Illustration of a Gram matrix with circulant blocks (left) and its block-diagonalization (right). Forming the $ns \times ns$ Gram matrix G from the dataset illustrated in Fig. 5.1, it is clear that there is some structure at the block level. We show that G has circulant blocks (Sec. 5.2.1). Transforming by V block-diagonalizes the Gram matrix, resulting in s smaller and independent sub-problems (one per block). Each sub-problem will correspond to a distinct frequency of the Fourier transform.

The cyclic shifts in Eq. 5.2 then model the finer translations. Fig. 5.1 illustrates this idea.

5.2.1 *Influence of cyclic shifts on a learning problem*

In order to understand the role that such datasets play in a learning algorithm, probably the most direct quantities that we can analize are the uncentered covariance matrix $X^T X$, and the Gram matrix XX^T. The covariance matrix is somewhat limited in usefulness, as it appears only in the expression for Ridge Regression (Eq. 2.4)[1]. However, the Gram matrix is more pervasive: every algorithm presented in Chapter 2 can be expressed in the dual, and access the data solely through the Gram matrix (Eq. 2.16, 2.18, 2.19, 2.20). It is due to this wider availability that we will focus our efforts on the Gram matrix G.

Inspecting the rows of the data matrix X in Eq. 5.3, we can observe that they are partitioned into n blocks, one per base sample, and each block contains s samples (obtained by cyclic shifts). Since X is a matrix with n vertical blocks, the Gram matrix $G = XX^T$ must also be a block matrix. To compute its values we can use the rules of block-matrix multiplication, which are essentially the same as those of stan-

1 Whether the covariance matrix is used at all depends, of course, on the optimization method. Newton methods rely on the Hessian matrix (second-order derivatives of the objective) that often shares similarities with the covariance matrix, but in general it does not exhibit the properties we discuss here [80, 42].

dard multiplication but replacing matrix elements with matrix blocks [52, Section 1.3]. As a result, the n^2 blocks of G are simply

$$G(i,j) = C(\mathbf{x}_i)C^T(\mathbf{x}_j), \tag{5.4}$$

where each $G(i,j)$ is $s \times s$. By stacking the blocks $G(i,j)$ we obtain the full Gram matrix G, which is $sn \times sn$. Note that $G(i,j)$ is the product of two circulant matrices, and as such it must also be circulant (Section 3.7).

5.2.1.1 *Block-circulant vs. circulant blocks*

As shown earlier, the Gram matrix G is a matrix with circulant blocks (Eq. 5.4). Note that it is always possible to reorder its rows and columns so it becomes block-circulant (i.e., the elements of the blocks are arbitrary, but the blocks themselves follow a circulant pattern). This is explored in detail in Appendix A.2.2. Both forms of the Gram matrix are equivalent, because only the order of the samples is different.

5.2.2 *Block-diagonalization*

Since each block of the Gram matrix $G(i,j)$ is circulant, we can diagonalize it by Theorem 3:

$$\begin{aligned} G(i,j) &= U \operatorname{diag}(\hat{\mathbf{x}}_i)\ U^*U \operatorname{diag}(\hat{\mathbf{x}}_j)\ U^* && (5.5)\\ &= U \operatorname{diag}(\hat{\mathbf{x}}_i \odot \hat{\mathbf{x}}_j)\ U^* && (5.6)\\ &= U\, G'(i,j)\, U^*, && (5.7) \end{aligned}$$

where $\odot$ is the element-wise product, and we defined a diagonalized block as $G'(i,j) = \operatorname{diag}(\hat{\mathbf{x}}_i \odot \hat{\mathbf{x}}_j)$.

The fact that we can diagonalize each block $G(i,j)$ hints at a possible diagonalization of the *full* Gram matrix G. A nearly-diagonal Gram matrix G would be extremely sparse, which would simplify significantly most learning problems in the dual (Section 2.2). We will now focus on achieving this goal.

Concatenating the blocks of Eq. 5.7 into a full matrix G, and doing the same for G', we can collect the U and U^* factors outside of G', yielding

$$G = (I \otimes U)\, G'\, (I \otimes U)^*, \tag{5.8}$$

where $\otimes$ is the Kronecker product (Section A.2.3).

There is a unique permutation R that changes any matrix with diagonal blocks, such as G', into a block-diagonal matrix,

$$G' = R^T \hat{G} R, \qquad \hat{G} = \begin{bmatrix} \hat{G}(1) & & \\ & \ddots & \\ & & \hat{G}(s) \end{bmatrix}. \tag{5.9}$$

For the purposes of this derivation we only need to know that such a permutation exists, however it is explored in more detail in Appendix A.2. We point out that this permutation R is unique and is known in the literature as the *commutation matrix* [85]. The blocks of $\hat{G}$ can then be computed explicitly by drawing from the elements of G',

$$\hat{G}_{ij}(f) = \hat{x}_{if}\hat{x}_{jf}, \tag{5.10}$$

where $\hat{x}_{if}$ denotes the fth element of $\hat{\mathbf{x}}_i$. A block $\hat{G}(f)$ can be interpreted as the Gram matrix for a Fourier frequency f: it is composed of the dot-products between the frequency f from all pairs of samples.

It is possible to simplify Eq. 5.8, by replacing Eq. 5.9 and using the fact that the commutation matrix R commutes the arguments of Kronecker products [85] (Appendix A.2.3).

$$G = (U \otimes I)\, \hat{G}\, (U \otimes I)^* . \tag{5.11}$$

5.2.3 *Unitarity revisited*

For convenience, we can denote $U \otimes I = V$. It can be easily verified that the diagonalization matrix V is unitary, a property inherited from the unitary DFT matrix U and the identity I. In other words, $V^{-1} = V^H$ (where $V^H = (V^*)^T$ is the Hermitian transpose, see Section 3.4).

The interest in unitary transformations lies in the fact that they preserve dot-products. Given two vectors $\mathbf{a}$ and $\mathbf{b}$, and denoting $\hat{\mathbf{a}} = V\mathbf{a}$ and $\hat{\mathbf{b}} = V\mathbf{b}$, then we have $\hat{\mathbf{a}}^H\hat{\mathbf{b}} = \mathbf{a}^H\mathbf{b}$. Since L^2-norms are dot-products, they are also preserved: $\mathbf{a}^H\mathbf{a} = \|\mathbf{a}\|^2 = \|\hat{\mathbf{a}}\|^2$. These properties will be very useful in the next section.

5.3 SEPARABILITY OF LEARNING PROBLEMS

We now need to prove the intuition that the blocks of a block-diagonal Gram matrix $\hat{G}$ indeed represent independent learning problems. This can yield considerable computation and storage savings, since

the number of elements under consideration is $(s-1)\,s$ times smaller than for the full G. The exact partitioning into sub-problems is also suitable for parallel implementations that take full advantage of modern architectures.

As mentioned earlier, V preserves dot-products and norms. If the dual of an algorithm can be represented exactly using dot-products and norms of the data in the dual space, then transforming this space by V will not affect the result. Additionally, both a block-diagonal $\hat{G}$ and any dot-products and norms can be decomposed into sums of their respective blocks. This means that the blocks can be optimized separately, proving our initial intuition. The result is formalized in the following theorem.

Theorem 6. *Consider a general learning problem, expressed in the dual variables $\boldsymbol{\alpha}$, with labels $\mathbf{y}$ and Gram matrix G:*

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^H G\boldsymbol{\alpha} + \sum_i^n D\left(\alpha_i, y_i\right). \tag{5.12}$$

The function D can vary depending on the chosen algorithm. Given a unitary matrix V such that $\hat{G} = V^*GV$ is block-diagonal, with s blocks $\hat{G}_f$, then Eq. 5.12 can be decomposed into s sub-problems

$$\min_{\hat{\boldsymbol{\alpha}}_f} \frac{1}{2}\hat{\boldsymbol{\alpha}}_f^H \hat{G}_f \hat{\boldsymbol{\alpha}}_f + \sum_i^n D\left(\hat{\alpha}_{fi}, \hat{y}_{fi}\right), \quad f = 1, \ldots, s, \tag{5.13}$$

with the transformed variables $\hat{\boldsymbol{\alpha}} = V^*\boldsymbol{\alpha}$ and $\hat{\boldsymbol{y}} = V^*\boldsymbol{y}$. Both $\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{y}}$ are partitioned into s blocks $\hat{\boldsymbol{\alpha}}_f$ and $\hat{\boldsymbol{y}}_f$, each with n elements $\hat{\alpha}_{fi}$ and $\hat{y}_{fi}$.

This relation is exact if the function D only depends on dot-products of its arguments, and approximate otherwise.

The proof is given in Appendix A.3.1. We are now ready to explore the implications for a number of learning algorithms.

5.3.1 *Ridge Regression*

Ridge regression (RR) is a regularized form of least-squares, with loss function (Section 2.1.1)

$$L\left(\mathbf{w}^T\mathbf{x}, y_i\right) = \left(\mathbf{w}^T\mathbf{x} - y_i\right)^2. \tag{5.14}$$

Although we can compute the dual solution in closed form (Section 2.2.1), it is easier to consider it as a minimization problem, which conforms to Eq. 5.12. In the case of Ridge Regression, the dual has

$D(\alpha_i, y_i) = \frac{\lambda}{2}\alpha_i^2 - \lambda\alpha_i y_i$ [112]. Since its terms are all dot-products, the decomposition is exact.

5.3.2 *Support Vector Regression*

L^2-SVR penalizes errors with the squared epsilon-insensitive loss (Section 2.1.3)

$$L(f(\mathbf{x}), y) = \left|\mathbf{w}^T\mathbf{x} - y\right|_\epsilon^2 = \max\left(0,\ \left|\mathbf{w}^T\mathbf{x} - y\right| - \epsilon\right)^2. \tag{5.15}$$

The dual has $D(\alpha_i, y_i) = \frac{\lambda}{2}\alpha_i^2 - \alpha_i y_i + \epsilon|\alpha_i|$ [137, Section 6.2.2]. Only the last term, an L^1-norm, is not a dot-product. As such, the approximation error is bounded by $\epsilon\left|\|\bar{\boldsymbol{\alpha}}\|_1 - \|\boldsymbol{\alpha}\|_1\right|$.

5.3.3 *General case*

The same analysis applies to L^1-SVR, Logistic Regression and other dual formulations, with varying degrees of approximation. It is also possible to characterize the transformations that preserve L^1 and L^2-norms exactly, but such a restriction makes them less useful (this is explored in Appendix A.3.2). We do not consider SVM because it restricts the labels to $\{-1, 1\}$, and a unitary transformation of the labels may fall outside this set.

5.4 EXPLICIT DATA MATRIX

The transformed Gram matrix for each of the s sub-problems, $\hat{G}(f)$, can be used directly in a dual solver (e.g., `libsvm` [42]). However, if n is large, an explicit description of the transformed data matrix that generates such a Gram matrix (through Eq. 2.17) would be more desirable.

By inspecting Eq. 5.10, it can be seen that each block $\hat{G}(f)$ corresponds to a distinct Fourier frequency f, and each of its elements (i, j) is simply the product of frequency f of the Fourier transforms of samples $\mathbf{x}_i$ and $\mathbf{x}_j$.

Because it is a simple product, it can be factorized into

$$\hat{G}(f) = \hat{X}(f)\hat{X}^H(f), \tag{5.16}$$

Algorithm 5.1 Matlab code for the Circulant Decomposition with multiple base samples. See Section 5.6.2.1 for details. This algorithm is equivalent to solving a regression with all spatial translations of the given samples. The full test suite can be downloaded at: `http://www.isr.uc.pt/~henriques/`

Inputs:
- X: $s_1 \times s_2 \times m \times n$ data matrix with n samples, each with m features on a $s_1 \times s_2$ grid
- Y: $s_1 \times s_2 \times n$ labels matrix
- `regression`: a complex-valued linear regression function

Output:
- W: $s_1 \times s_2 \times m$ weights matrix

```
X = fft2(X);
Y = fft2(Y);
Y(1,1,:) = 0;
X = permute(X, [4, 3, 1, 2]);
Y = permute(Y, [3, 1, 2]);
for f1 = 1:s1
  for f2 = 1:s2
    W(f1,f2,:) = regression(X(:,:,f1,f2), Y(:,f1,f2));
  end
end
W = real(ifft2(W));
```

where $\hat{X}(f)$ is an $n \times 1$ vector with the Fourier frequency f of each sample. This explicit description of the data matrix $\hat{X}(f)$ allows us to use a fast primal solver such as `liblinear` [42].

5.4.0.1 *Extension to two dimensions*

All properties of the Fourier transform (FT) and circulant matrices we used have direct equivalents in 2D, i.e., when samples contain $s_1 \times s_2$ spatial cells. We just have to replace the 1D FT with the 2D FT, and set $s = s_1 s_2$.

5.4.0.2 *Extension for multiple features per cell*

We can simply extend $\hat{X}(f)$ to be a $n \times m$ matrix with the Fourier frequency f of m features. By additivity of the dot-product, the Gram matrix obtained this way is the sum of Gram matrices over all m features, and all properties are preserved.

5.5 COMPLEX-VALUED REGRESSION

The fact that the data matrices of Eq. 5.16 are complex may apparently present some difficulties, since regression is usually real-valued. In this section we discuss some solutions. We consider a generic data matrix X and regression targets $\mathbf{y}$.

The simple algorithm of Ridge Regression is already prepared to deal with complex values:

$$\mathbf{w}^* = \left(X^H X + \lambda I\right)^{-1} X^H \mathbf{y}. \tag{5.17}$$

The only difference from the real case is that the solution must be complex-conjugated.

The SVR can deal with the complex case by extending its loss function (Eq. 5.15) from the real line to the complex plane:

$$L\left(\mathbf{w}^H \mathbf{x}, y\right) = \left|\mathrm{Re}\left(\mathbf{w}^H \mathbf{x} - y\right)\right|_\epsilon^2 + \left|\mathrm{Im}\left(\mathbf{w}^H \mathbf{x} - y\right)\right|_\epsilon^2, \tag{5.18}$$

where $\mathrm{Re}\left(\cdot\right)$ extracts the real part of a complex number, and $\mathrm{Im}\left(\cdot\right)$ the imaginary part. For real arguments, this reduces to the simple SVR. We can prove (Appendix A.3.3) that this is equivalent to a real SVR with the augmented data matrix X' and augmented targets $\mathbf{y}'$,

$$X' = \begin{bmatrix} \mathrm{Re}\left(X\right) & \mathrm{Im}\left(X\right) \\ \mathrm{Im}\left(X\right) & -\mathrm{Re}\left(X\right) \end{bmatrix}, \quad \mathbf{y}' = \begin{bmatrix} \mathrm{Re}\left(\mathbf{y}\right) \\ \mathrm{Im}\left(\mathbf{y}\right) \end{bmatrix}, \tag{5.19}$$

where the rows of X' are the samples of the new real SVR problem, and the elements of $\mathbf{y}'$ are its regression targets. The complex solution $\mathbf{w}$ can then be reconstructed from the augmented real solution $\mathbf{w}'$,

$$\mathbf{w}' = \begin{bmatrix} \mathrm{Re}\left(\mathbf{w}\right) \\ \mathrm{Im}\left(\mathbf{w}\right) \end{bmatrix}. \tag{5.20}$$

Note that this is not the same as simply concatenating the real and imaginary parts as features. The structure of X' ensures that the properties of complex numbers are respected (e.g., $i.i = -1$, with i the pure-imaginary unit).

5.6 EXPERIMENTS

We tested the proposed decomposition on a number of detection tasks. Recall that the proposed method can replace the traditional hard negative mining steps with a single learning phase. As such, the goal of these experiments is *not* to show greater accuracy, but to

Figure 5.3: Example detections in the Inria Pedestrians dataset. The opacity of the bounding boxes is proportional to the confidence in each detection. Best viewed in color.

verify that we can achieve accuracy that is competitive with several rounds of hard negative mining, *all other components being constant*. If we introduced different variations (features, etc), results would be less conclusive. We chose the task of learning a single HOG filter from full object exemplars, which captures the core component shared by many modern object detectors [44, 13]. Note that these contributions are orthogonal to other recent advances in object detection, such as the use of part filters and multiple object components [44, 13, 135].

In our tests, we will apply the Circulant Decomposition (CD) to an SVR solver, as we found SVR to perform similarly to regular SVM in practical object detection tasks. The experiments will evaluate several aspects: 1) how detection performance with the CD single-batch learning relates to SVM learning as the number of rounds of hard negative mining increases, 2) the implicit ability of CD to enlarge the training set with many translations, and its impact on datasets with few positive samples and 3) the computational savings of CD, compared to hard negative mining, which is generally considered expensive.

5.6.1 *Detector evaluation metrics*

Probably the most well-known detector evaluation metrics are Precision-Recall curves (for a visual comparison), and Average Precision (for a summary statistic), made popular by the Pascal VOC Challenge [41, 44]. An important feature is that these metrics are threshold independent. The output of a detector is typically a list of bounding boxes

(object locations obtained during scanning), and each has an associated confidence score. In the case of a discriminative classifier, the output of the model $f(\mathbf{z}) \in \mathbb{R}$ at the bounding box's location gives a measure of confidence. Thresholding $f(\mathbf{z})$ at a given confidence level will separate the object detections from the background.

However, this threshold depends heavily on the tradeoffs associated with a particular application, where different kinds of errors may be acceptable. A low threshold means that more candidates are considered objects, so there are few missing detections (false negatives), but a higher probability of spurious detections (false positives). A high threshold will filter the candidates aggressively, with fewer false positives, but may miss more detections (false negatives).

It is desirable to measure performance in an application-independent (and thus threshold-independent) manner, so methods are compared using a plot of the precision and recall over all possible thresholds (see Fig. 5.4). For a given threshold, precision is computed as $TP/(TP + FP)$, where TP is the number of true positives (correct detections) and FP the number of false positives. Recall is computed as $TP/(TP + FN)$, where FN is the number of false negatives. This allows us to plot a Precision-Recall curve for all thresholds, where methods with higher curves generally fare better. The average value of the precisions in the curve (AP) is typically used as a single performance statistic, with higher values corresponding to a more accurate detector.[2]

5.6.2 *Pedestrian detection*

We experimented with pedestrian detection on two standard datasets: the well-known INRIA Pedestrians [30] and the more recent Caltech Pedestrian Detection Benchmark [34]. Example detections obtained using the proposed method are shown in Fig. 5.3 and 5.8.

Before delving into the details of the detector implementations, we will show the main results of the experimental section – that a Circulant Decomposition is equivalent to training with all negative windows, a feat that can only be approximated by several rounds of hard negative mining. Fig. 5.4 shows a comparison of CD and different numbers of hard negative rounds for INRIA Pedestrians and Fig. 5.5 shows the same comparison for the Caltech Pedestrian Detection

2 The criteria for considering a detection correct (to compute TP, FP and FN) is that it overlaps by more than 50% with a ground truth bounding box. Their relative overlap is defined as the area of the intersection of the two bounding boxes, divided by the area of their union.

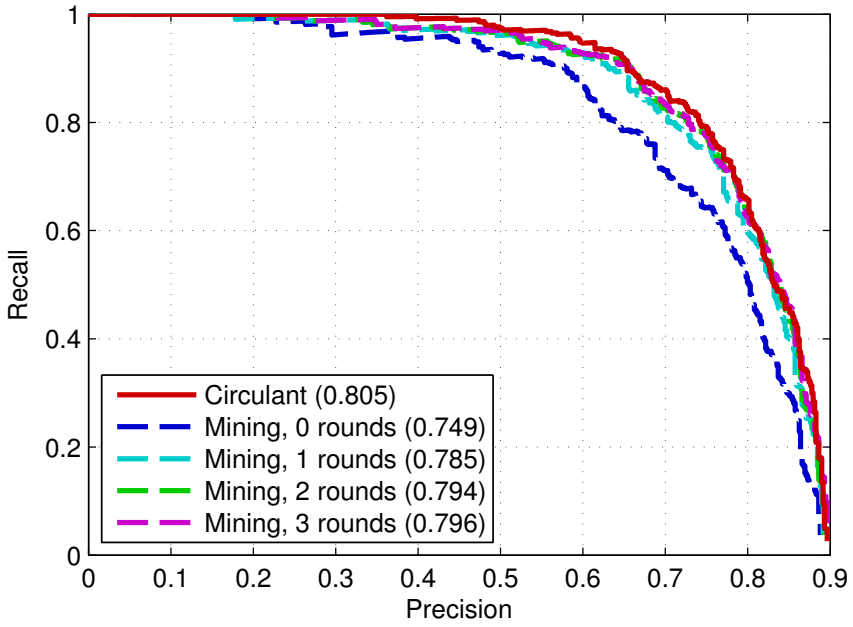


Figure 5.4: Performance on the test set of INRIA Pedestrians using a HOG detector. It takes several rounds of hard negative mining to converge to the same results of the proposed Circulant Decomposition, which is trained on the full set of negative windows. The Circulant Decomposition allows training on the full set in one go. AP is shown in brackets.

Benchmark. The Average Precision (AP) is shown in brackets. The results suggest that the Circulant Decomposition performs on par with the slower and intrinsically less complete process of learning with hard negative mining.

5.6.2.1 *Implementation*

We followed the original implementation of the HOG pedestrian detector [30]. On INRIA Pedestrians the baseline classifier is trained with 12180 random negative windows, before mining hard negative examples from the set of 1218 negative images, which contains $\sim 10^8$ potential windows. For our method, which can train with all $\sim 10^8$ windows in this set, we consider a total of $\sim 10^5$ base samples. The finer translations within each patch are implicitly dealt with by the Circulant Decomposition, which is the main advantage of our approach. We proceed similarly on the "reasonable" subset of the Caltech Pedestrians dataset [34], composed of 4250 training images obtained every 30 frames, of which 2217 are negative images, without pedestrians. All tests were done on a quad-core 3.0Ghz desktop computer. Both CD and mining implementations are parallelized, providing a fair representation of a modern set-up.

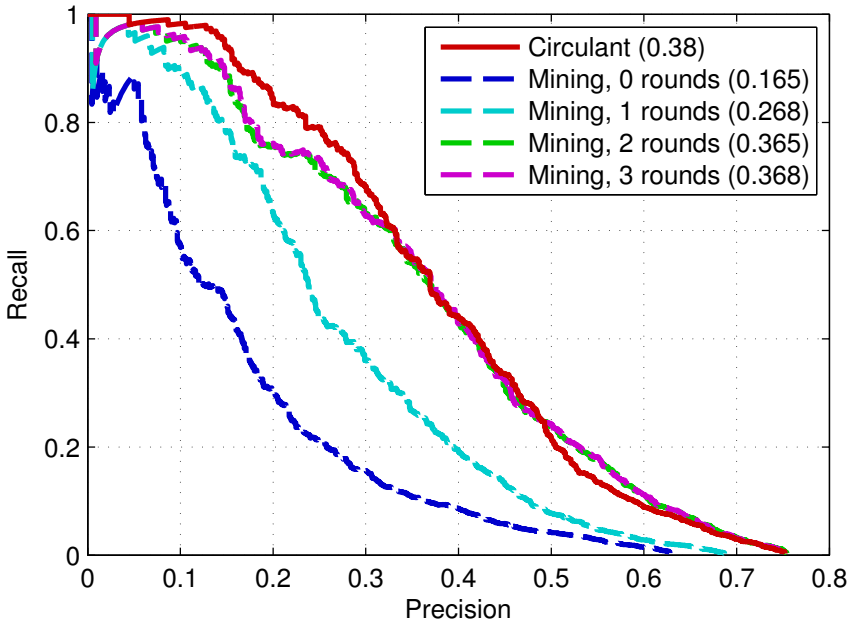


Figure 5.5: Performance on the test set of the Caltech Pedestrians Detection Benchmark (AP is shown in brackets) using a HOG detector. The Circulant Decomposition is competitive with 3 expensive rounds of hard negative mining.

An implementation of the proposed method using this data matrix is given in Algorithm 5.1. For HOG descriptors, m is the number of orientation bins, over an $s_1 \times s_2$ grid, and n is the number of base samples. After transforming X to the Fourier domain, the built-in Matlab function `permute` will reorder the dimensions of the data matrix X, from $s_1 \times s_2 \times m \times n$ to $n \times m \times s_1 \times s_2$. Extracting a $n \times m$ slice from this array yields the data matrix for one of the $s_1 s_2$ regression subproblems. The results of the individual regressions are stored into $\mathbf{w}$, which is then transformed back to the spatial domain to yield the solution.

Additionally, we verified experimentally that it is necessary for the regression targets to have no DC component (line 3 of Algorithm 5.1.) A possible reason is that the DC component of the data, which corresponds to the mean in the spatial domain, tipically has values several orders of magnitude larger than the remaining frequencies.

5.6.2.2 *Cyclic shifts as a model for translation*

Since samples must have the same support as the learned template $\mathbf{w}$, cyclic shifts of a template-sized sample are less accurate for large translations, due to wrap-around effects. Thus in practice we collect base samples $\mathbf{x}_i$ from negative images in a grid, at regular intervals

		MINING				CIRCULANT
ROUNDS		0	1	2	3	**0**
Time (s)	INRIA	7	159	312	463	**35**
	Caltech	12	646	1272	1901	**139**
AP	INRIA	0.749	0.785	0.794	0.796	**0.805**
	Caltech	0.165	0.268	0.365	0.368	**0.380**

Table 5.1: Performance and timing of the proposed method on the INRIA and Caltech Pedestrians datasets, and the classical approach using increasing numbers of hard negative mining rounds. The proposed method converges on the solution in one go, requiring much less time than even a *single* round of mining.

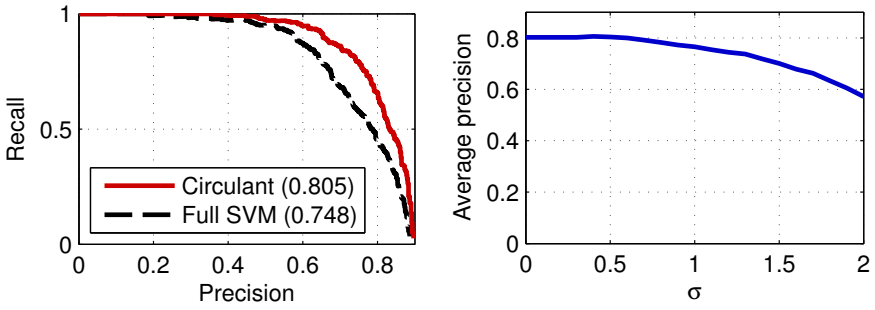
of 2/3 of the template size, which accounts for large translations. We verified experimentally that there is little impact in performance if we assume cyclic shifts are accurate up to $\sim$1/3 of the template size in all directions. The cyclic shifts $P^{u-1}\mathbf{x}_i$ model the finer translations, which comes at no additional cost when using the Circulant Decomposition. This allows us to effectively model a sliding window, while collecting only a few dozen base samples per negative image.

To verify that the gain in performance is truly due to the Circulant Decomposition and not the grid sampling scheme, we trained a full SVM classifier with the same base samples. The results in Fig. 5.6-a show that performance drops significantly without the proposed decomposition, which also makes training faster and easier to parallelize.

5.6.2.3 *Positive samples*

Another aspect of our method is that we must choose labels for translations of positive samples, since they are implicitly accounted for during training.

A translation of a positive sample at some point becomes a false positive, so a simple choice is to assign it the label +1 if $t = 0$ (aligned), and -1 if $t \neq 0$ (misaligned). Alternatively, and taking advantage of the fact that regression allows labels outside the set $\{-1, +1\}$, we can use a Gaussian function to interpolate smoothly between the two, according to a Gaussian bandwidth σ. For $\sigma = 0$ the function looks like a single peak, and we recover the first choice we mentioned. This function plays a similar role to the regression targets in correlation filters (Appendix A.1.1).



(a) (b)

Figure 5.6: INRIA Pedestrians. (a) Comparison of Circulant Decomposition and a full SVM (not decomposed) using the same base samples. (b) Performance as spatial bandwidth σ is varied (see text). Small values offer better performance because high localization in detection tasks can be understood as suppressing off-center detections.

Performance on INRIA Pedestrians as this parameter varies is shown in Fig. 5.6-b. Larger bandwidths seem to degrade performance, and the best performance is attained near $\sigma = 0$. Since good localization (suppressing misaligned detections) is important in detection tasks, this result agrees with intuition.

5.6.3 *ETHZ Shapes*

As mentioned earlier, translations of positive samples are also considered. It is possible that these "virtual samples" help regularize the solution in settings with a scarcity of positive samples. To test this hypothesis, we followed the same methodology as before but on the ETHZ Shapes dataset [45]. This dataset has only between 22 and 45 positive examples for each of its 5 categories (Mugs, Bottles, Swans, Giraffes and Apple Logos), evenly split into training and testing sets. Example detections of each class are shown in Fig. 5.9. Unlike on the pedestrian detection benchmarks, where positive examples abound, here results are markedly improved using the proposed method, as visible in Fig. 5.7. We conjecture, based on these results, that our approach could also benefit applications such as one-shot learning and transfer learning.

5.7 CONCLUSIONS

Supported by the observation that the Gram matrix of the set of training images and their translations has circulant blocks, we have

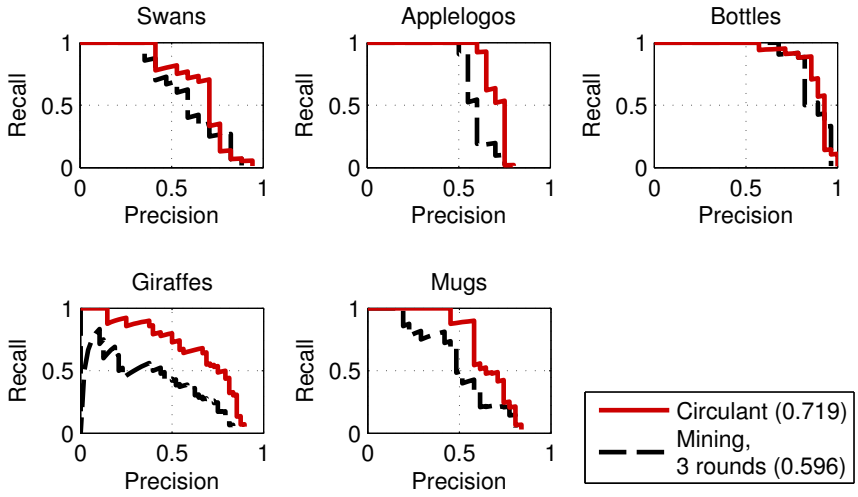


Figure 5.7: Performance on ETHZ Shapes, a dataset with scarce training data (between 22 and 45 positive examples per class, with an equal train-test split). In addition to allowing models to be trained faster, the Circulant Decomposition achieves higher performance in this case, perhaps due to the implicit inclusion of translated positive samples. (Mean AP is shown in brackets.)

derived a closed-form decomposition that allows for popular filter-based detectors to be efficiently learned from all subwindows of a given size in a single batch, on datasets with a few thousand training images. This is surprising since the number of subwindows is in the order of 10^8, which precludes even loading the data matrix into a current computer's memory, but is feasible using our proposed embedding of the learning problem into the Fourier domain. This methodology is likely to have broad applicability, as it allows for both a much more efficient and more complete learning process than iterative hard negative mining, used for learning many modern object detectors.

After exploiting the nature of image translations to accelerate learning algorithms, we will now turn to an even more challenging problem: to exploit the structure of arbitrary image transformations, such as rotations or image scaling.

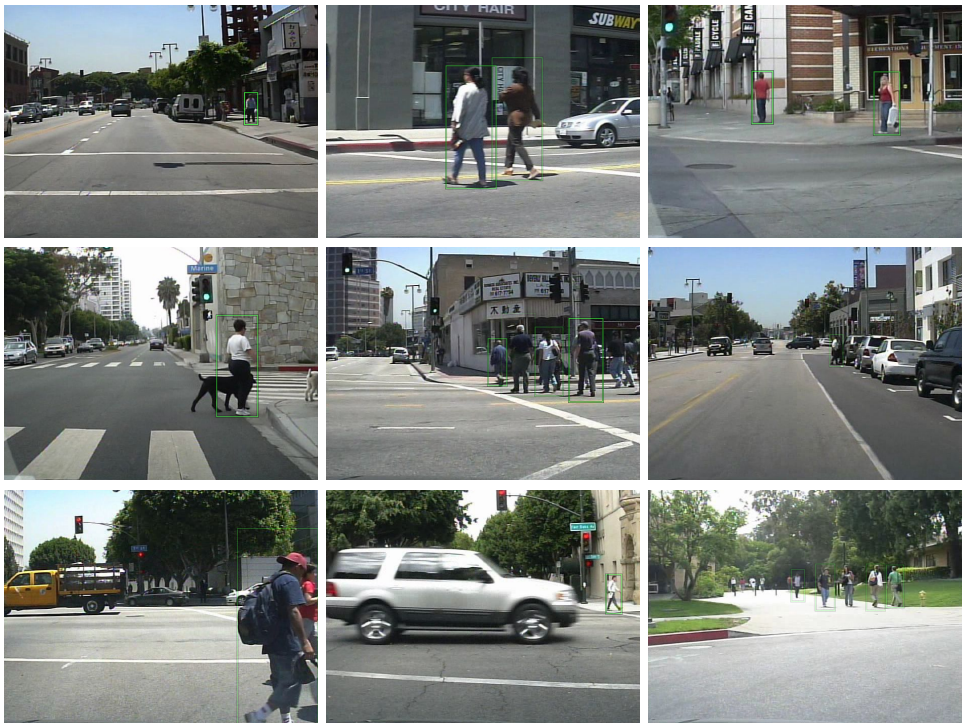


Figure 5.8: Example detections in the Caltech Pedestrians dataset. The opacity of the bounding boxes is proportional to the confidence in each detection. Best viewed in color.

Figure 5.9: Example detections in the ETHZ Shapes dataset, one per object class. The opacity of the bounding boxes is proportional to the confidence in each detection. Best viewed in color.

6

GENERAL GEOMETRIC TRANSFORMATIONS

In many datasets, the samples are related by a known image transformation, such as rotation, or a repeatable non-rigid deformation. This applies to both datasets with the same objects under different viewpoints, and datasets augmented with virtual samples. Such datasets possess a high degree of redundancy, because geometrically-induced transformations should preserve intrinsic properties of the objects. Likewise, ensembles of classifiers used for pose estimation should also share many characteristics, since they are related by a geometric transformation. This represents a significant generalization of the results from the previous chapters, which considered the same problem but for image translation only. In this chapter we will show that, under the assumption that the transformation is norm-preserving and cyclic, a simple closed-form solution in the Fourier domain can eliminate most redundancies. It can leverage off-the-shelf solvers with no modification (e.g. `libsvm` [23]), and train several pose classifiers simultaneously at no extra cost. The experiments will show that training a sliding-window object detector and pose estimator can be sped up by orders of magnitude, for transformations as diverse as planar rotation, the walking motion of pedestrians, and out-of-plane rotations of cars.

6.1 INTRODUCTION

To cope with the rich variety of transformations in natural images, recognition systems require a representative sample of possible variations. Some of those variations must be learned from data (e.g. non-rigid deformations), while others can be virtually generated (e.g. translation or rotation, see Fig. 6.1). Recently, there has been a renewed interest in augmenting datasets with virtual samples, both in the context of supervised [104, 76] and unsupervised learning [37]. This augmentation has the benefits of regularizing high-capacity classifiers [37], while learning the natural invariances of the visual world.

Some kinds of virtual samples can actually make learning easier – for example, with horizontally-flipped virtual samples [44, 30, 76], half of the weights of the template in the Dalal-Triggs detector [30] become redundant by horizontal symmetry. The developments of

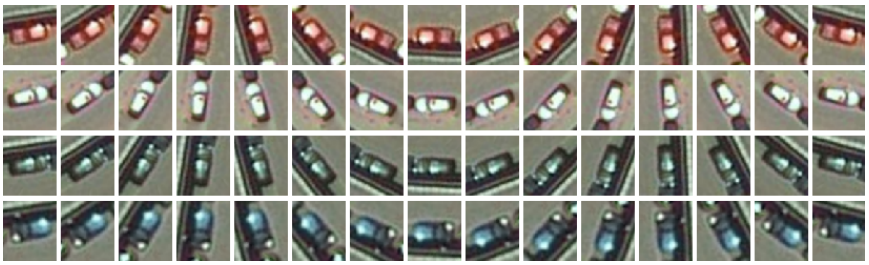


Figure 6.1: Virtual samples generated by rotation of 4 cars from aerial images. The deterministic rotations create an intrinsic pattern in the dataset, that we propose to exploit.

Chapters 4 and 5 have shown that cyclically translated virtual samples *also* constrain learning problems, to obtain substantial gains in computational efficiency. In this chapter, we will show that the "Fourier trick" responsible for these gains is not unique to cyclic translation, but can be generalized to other cyclic transformations. The proposed model captures a wide range of useful image transformations, yet retains the ability to accelerate training with the DFT. As it is only implicit, we can accelerate training in both datasets of virtual samples and natural datasets with pose annotations.

Also due to the geometrically-induced structure of the training data, the proposed algorithm can obtain several transformed pose classifiers simultaneously. Some of the best object detection and pose estimation systems currently learn classifiers for different poses independently [47, 44, 88], and we show how joint learning of these classifiers can dramatically reduce training times.

6.1.1 *Related work*

There is a vast body of works on image transformations and invariances, of which we can only mention a few. Much of the earlier computer vision literature focused on finding viewpoint-invariant patterns [97]. They were based on image or scene-space coordinates, on which geometric transformations can be applied directly, however they do not readily apply to modern appearance-based representations. To relate complex transformations with appearance descriptors, a classic approach is to use tangent vectors [24, 124, 72], which represent a first-order approximation. However, the desire for more expressiveness has motivated the search for more general models.

Recent works have begun to approximate transformations as matrix-vector products, and try to estimate the transformation matrix ex-

plicitly. Tamaki et al. [131] do so for blur and affine transformations in the context of Linear Discriminant Analysis (LDA), while Miao et al. [94] approximate affine transformations with an Expectation-Maximization (E-M) algorithm, based on a Lie group formulation. They estimate a basis for the transformation operator or the transformed images, which is a hard analytical/inference problem in itself. The involved matrices are extremely large for moderately-sized images, necessitating dimensionality reduction techniques such as Principal Component Analysis (PCA), which may be suboptimal.

Several works focus on rotation alone [119, 81, 134, 22], most of them speeding up computations using Fourier analysis, but they all explicitly estimate a reduced basis on which to project the data. Another approach is to learn a transformation from data, using more parsimonious factored or deep models [93]. In contrast, our method generalizes to other transformations and avoids a potentially costly transformation model or basis estimation.

6.2 THE CYCLIC ORTHOGONAL MODEL FOR IMAGE TRANSFORMATIONS

Consider the $m \times 1$ vector $\mathbf{x}$, obtained by vectorizing an image (Section A.2). The image may be a 3-dimensional array that contains multiple channels, such as RGB, or the values of a densely-sampled image descriptor.

We wish to quickly train a classifier or regressor with transformed versions of sample images, to make it robust to those transformations. The model we will use is an $m \times m$ orthogonal matrix Q, which will represent an incremental transformation of an image as $Q\mathbf{x}$ (for example, a small translation or rotation, see Fig. 6.2-a and 6.2-b). We can traverse different poses w.r.t. that transformation, $p \in \mathbb{Z}$, by repeated application of Q with a matrix power, $Q^p\mathbf{x}$. Note that Q is a generalization of the cyclic shift matrix P we used earlier, but unlike P we do not constrain Q to be a particular matrix, or even require it to represent a permutation.

In order for the number of poses to be finite, we must require the transformation to be *cyclic*, $Q^s = Q^0 = I$, with some period s. This allows us to store all versions of $\mathbf{x}$ transformed to different poses as the rows of an $s \times m$ matrix,

$$C_Q(\mathbf{x}) = \begin{bmatrix} (Q^0\mathbf{x})^T \\ (Q^1\mathbf{x})^T \\ \vdots \\ (Q^{s-1}\mathbf{x})^T \end{bmatrix} \tag{6.1}$$

Due to Q being cyclic, any pose $p \in \mathbb{Z}$ can be found in the row $(p \bmod s) + 1$. Note that the first row of $C_Q(\mathbf{x})$ contains the untransformed image $\mathbf{x}$, since Q^0 is the identity I. For the purposes of training a classifier, $C_Q(\mathbf{x})$ can be seen as a data matrix, with one sample per row.

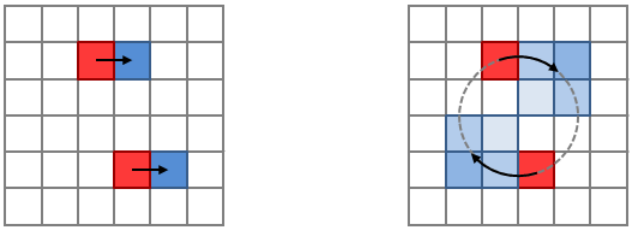
The cyclic orthogonal model can be related to our earlier results by setting $Q = P$, the cyclic shift matrix (Eq. 3.2). In this case Q represents (cyclic) image translations, and $C_Q(\mathbf{x})$ is reduced to a simple circulant matrix $C(\mathbf{x})$. An important open question was whether the diagonalization of standard circulant matrices (Theorem 3) can be reused for image transformations other than translation. We will show that this is true, using the model from Eq. 6.1.

We must contrast this model to tangent vectors and subspace methods, as used in several prior works [24, 131], which are only valid in a small neighborhood of p around the original image $\mathbf{x}$. Outside that scope, their predicted values may diverge to zero or to infinity. By virtue of the orthogonality of Q, $Q^p\mathbf{x}$ is guaranteed to preserve the norm of the original image $\mathbf{x}$ for any pose values p, so our model is inherently stable.

Although the cyclic orthogonal model is conceptually simple, we will show through experiments that it can accurately capture a variety of natural transformations (Section 6.5.2). More importantly, we will show that *Q never has to be created explicitly*. The algorithms we develop will be entirely data-driven, using an implicit description of Q from a structured dataset, either composed of virtual samples (e.g., by image rotation), or natural samples (e.g. using pose annotations).

6.3 FAST TRAINING WITH TRANSFORMATIONS OF A SINGLE IMAGE

We will now focus on the main derivations, which allow us to quickly train a classifier with virtual samples generated from an image $\mathbf{x}$ by repeated application of the transformation Q. This section assumes only a single image $\mathbf{x}$ is given for training, which makes the presentation simpler and we hope will give valuable insight into the core of the technique. Section 6.4 will expand it to full generality, with training sets of an arbitrary number of images, all transformed by Q.



(a) (b)

Figure 6.2: (a) The horizontal translation of a 6×6 image, by 1 pixel, can be achieved by a 36×36 permutation matrix P that reorders elements appropriately (depicted is the reordering of 2 pixels). (b) Rotation by a fixed angle, with linearly-interpolated pixels, requires a more general matrix Q. By studying its influence on a dataset of rotated samples, we show how to accelerate learning in the Fourier domain. Our model can also deal with other transformations, including non-rigid.

The first step is to show that some aspect of the data is diagonalizable by the DFT, which we do in the following theorem.

Theorem 7. *Given an orthogonal cyclic matrix Q, i.e. satisfying $Q^T = Q^{-1}$ and $Q^s = Q^0$, then the $s \times m$ matrix $X = C_Q(\mathbf{x})$ (from Eq. 6.1) verifies the following:*

- The data matrix X and the uncentered covariance matrix $X^T X$ are not circulant in general, unless $Q = P$ (from Eq. 3.2).
- The Gram matrix $G = XX^T$ is always circulant.

Proof. See Appendix A.4.1. □

Theorem 7 implies that the learning problem in its original form is not diagonalizable by the DFT basis. However, the same diagonalization is possible for the dual problem, defined by the Gram matrix G.

Because G is circulant, it has only s degrees of freedom and is fully specified by its first row $\mathbf{g}$ [56], $G = C(\mathbf{g})$. By direct computation from Eq. 6.1, we can verify that the elements of the first row $\mathbf{g}$ are given by $g_p = \mathbf{x}^T Q^{p-1} \mathbf{x}$. One interpretation is that $\mathbf{g}$ contains the autocorrelation of $\mathbf{x}$ through pose-space, i.e., the inner-product of $\mathbf{x}$ with itself as the transformation Q is applied repeatedly.

6.3.1 *Dual Ridge Regression*

For now we will restrict our attention to Ridge Regression (RR), since it has the appealing property of having a solution in closed form,

which we can easily manipulate. Section 6.4.1 will show how to extend these results to Support Vector Regression.

Since we have s samples in the data matrix under consideration (Eq. 6.1), there are s dual variables, stored in a vector $\boldsymbol{\alpha}$. The RR solution is given by $\boldsymbol{\alpha} = (G + \lambda I)^{-1} \mathbf{y}$ (Eq. 2.16), where $G = XX^T$ is the $s \times s$ Gram matrix, $\mathbf{y}$ is the vector of s labels (one per pose), and λ is the regularization parameter. The dual form of RR is usually associated with non-linear kernels (Section 2.3), but since this is not our case we can compute the explicit primal solution with $\mathbf{w} = X^T \boldsymbol{\alpha}$, yielding

$$\mathbf{w} = X^T (G + \lambda I)^{-1} \mathbf{y}. \tag{6.2}$$

Applying the circulant eigendecomposition (Theorem 3) to G, and substituting it in Eq. 6.2,

$$\mathbf{w} = X^T \left(U \operatorname{diag}(\hat{\mathbf{g}}) U^* + \lambda U U^*\right)^{-1} \mathbf{y} = X^T U \left(\operatorname{diag}(\hat{\mathbf{g}} + \lambda)\right)^{-1} U^* \mathbf{y}, \tag{6.3}$$

where we introduce the shorthand $\hat{\mathbf{g}} = \mathcal{F}(\mathbf{g})$, and similarly $\hat{\mathbf{y}} = \mathcal{F}(\mathbf{y})$. Since inversion of a diagonal matrix can be done element-wise, and its multiplication by the vector $U^*\mathbf{y}$ amounts to an element-wise product, we obtain

$$\mathbf{w} = X^T \mathcal{F}^{-1} \left(\frac{\hat{\mathbf{y}}}{\hat{\mathbf{g}} + \lambda} \right), \tag{6.4}$$

where $\mathcal{F}^{-1}$ denotes the inverse DFT, and the division is taken element-wise. This formula allows us to replace a costly matrix inversion with fast DFT and element-wise operations. We also do not need to compute and store the full G, as the auto-correlation vector $\mathbf{g}$ suffices. As we will see in the next section, there is a simple modification to Eq. 6.4 that turns out to be very useful for pose estimation.

6.3.2 *Training several components simultaneously*

A relatively straightforward way to estimate the object pose in an input image $\mathbf{x}$ is to train a classifier for each pose (which we call *components*), evaluate all of them and take the maximum, i.e.

$$f_{\text{pose}}(\mathbf{x}) = \arg\max_p \mathbf{w}_p^T \mathbf{x}. \tag{6.5}$$

This can also be used as the basis for a pose-invariant classifier, by replacing argmax with max [47]. Each pose classifier $\mathbf{w}_p$ could be trained individually using the method from Section 6.3.1, which can

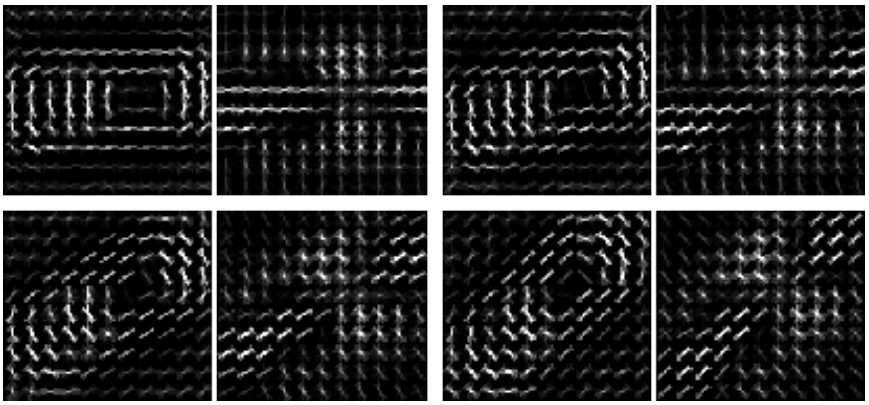


Figure 6.3: Example HOG template (a car from the Google Earth dataset) at 4 rotations learned by the proposed model. Positive weights are on the first and third column, others are negative.

quickly become expensive as we need to evaluate Eq. 6.4 a total of s times. However, we can exploit the fact that these training problems become tightly related when the training set contains transformed images.

Recall that $\mathbf{y}$ specifies the labels for a training set of s transformed images, one label per pose. Without any loss of generality, suppose that the label is 1 for a given pose t and 0 for all others, i.e. $\mathbf{y}$ contains a single peak at element t. Then by shifting the peak with $P^p\mathbf{y}$, we will train a classifier for pose $t+p$. In this manner we can train classifiers for all poses simply by varying the labels $P^p\mathbf{y}$, with $p = 0, \ldots, s-1$.

Based on Eq. 6.4, we can concatenate the solutions for all s components into a single $m \times s$ matrix,

$$W = \left[\ \mathbf{w}_0 \ \middle|\ \cdots \ \middle|\ \mathbf{w}_{s-1}\ \right] = X^T \left(G + \lambda I\right)^{-1} \left[\ P^0\mathbf{y} \ \middle|\ \cdots \ \middle|\ P^{s-1}\mathbf{y}\ \right] \tag{6.6}$$

$$= X^T \left(G + \lambda I\right)^{-1} C^T\left(\mathbf{y}\right). \tag{6.7}$$

Diagonalization yields

$$W^T = \mathcal{F}^{-1}\left(\operatorname{diag}\left(\frac{\hat{\mathbf{y}}^*}{\hat{\mathbf{g}} + \lambda}\right)\mathcal{F}\left(X\right)\right). \tag{6.8}$$

Since their arguments are matrices, the DFT/IDFT operations here work along each column. The product of $\mathcal{F}\left(X\right)$ by the diagonal matrix simply amounts to multiplying each of its rows by a scalar factor, which is inexpensive. Eq. 6.8 has nearly the same computational cost as Eq. 6.4, which trains a single classifier.

6.4 TRANSFORMATION OF MULTIPLE IMAGES

The training method described in the previous section would find little applicability for modern recognition tasks if it remained limited to transformations of a single image. Naturally, we would like to use n images $\mathbf{x}_i$. We now have a dataset of ns samples, which can be divided into n sample groups $\{Q^{p-1}\mathbf{x}_i | p = 1, \ldots, s\}$, each containing the transformed versions of one image.

This case becomes somewhat complicated by the fact that the data matrix X now has three dimensions – the m features, the n sample groups, and the s poses of each sample group. In this $m \times n \times s$ array, each column vector (along the first dimension) is defined as

$$X_{\bullet ip} = Q^{p-1}\mathbf{x}_i, \qquad i = 1, \ldots, n;\ p = 1, \ldots, s, \tag{6.9}$$

where we have used $\bullet$ to denote a one-dimensional slice of the three-dimensional array X.[1] A two-dimensional slice will be denoted by $X_{\bullet\bullet p}$, which yields a $m \times n$ matrix, one for each $p = 1, \ldots, s$.

Through a series of block-diagonalizations and reorderings, we can show (Appendix A.4.2-A.4.5) that the solution W, of size $m \times s$, describing all s components (similarly to Eq. 6.8), is obtained with

$$\hat{W}_{\bullet p} = \hat{X}_{\bullet\bullet p} \left(\hat{\mathbf{g}}_{\bullet\bullet p} + \lambda I\right)^{-1} \hat{Y}^*_{\bullet p}, \qquad p = 1, \ldots, s, \tag{6.10}$$

where a hat $\hat{}$ over an array denotes the DFT along the dimension that has size s (e.g. $\hat{X}$ is the DFT of X along the third dimension), Y_{ip} specifies the label of the sample with pose p in group i, and $\mathbf{g}$ is the $n \times n \times s$ array with elements

$$\mathbf{g}_{ijp} = \mathbf{x}_i^T Q^{p-1}\mathbf{x}_j = X_{\bullet i1}^T X_{\bullet jp}, \quad i, j = 1, \ldots, n;\ p = 1, \ldots, s. \tag{6.11}$$

It may come as a surprise that, after all these changes, Eq. 6.10 still essentially looks like a dual Ridge Regression (RR) problem (compare it to Eq. 6.2). Eq. 6.10 can be interpreted as splitting the original problem into s smaller problems, one for each Fourier frequency, which are independent and can be solved in parallel. A Matlab implementation is given in Algorithm 6.1.

1 For reference, our slice notation $\bullet$ works the same way as the slice notation `:` in Matlab or NumPy. For example, $X_{\bullet ip}$ would be represented as `X(:,i,p)`.

6.4.1 *Support Vector Regression*

Given that we can decompose such a large RR problem into s smaller RR problems, by applying the DFT and slicing operators (Eq. 6.10), it is natural to ask whether the same can be done with other algorithms. Leveraging the results of Chapter 5, where this was done for image translation, the same steps can be repeated for the dual formulation of other algorithms, such as Support Vector Regression (SVR). Although RR in the dual can deal with complex data, SVR requires an extension of the dual solution to the complex domain, which we show in Appendix A.4.6. We give a Matlab implementation in Algorithm 6.3, which can use any off-the-shelf SVR solver without modification.

6.4.2 *Efficiency*

Naively training one detector per pose would require solving s large $ns \times ns$ systems (either with RR or SVR). In contrast, our method learns *jointly* all detectors using s much smaller $n \times n$ subproblems. The computational savings can be several orders of magnitude for large s. Our experiments seem to validate this conclusion, even in relatively large recognition tasks (Section 6.6).

6.5 ORTHOGONAL TRANSFORMATIONS IN PRACTICE

Until now, we avoided the question of how to compute a transformation model Q. This may seem like a computational burden, not to mention a hard estimation problem – for example, what is the cyclic orthogonal matrix Q that models planar rotations with period s?

Inspecting the relevant equations (Eq. 6.10-6.11), however, reveals that we do not need to form Q explicitly, but can work with just a data matrix X of transformed images. From there on, we exploit the knowledge that this data was obtained from *some* matrix Q, and that is enough to allow fast training in the Fourier domain. This allows a great deal of flexibility in implementation.

6.5.1 *Virtual transformations*

One way to obtain a structured data matrix X is with virtual samples. From the original dataset of n samples, we can generate ns virtual samples using a standard image operator (e.g. planar rotation). However, we should keep in mind that the accuracy of the proposed method will be affected by how much the image operator resembles

Figure 6.4: Example detections and estimated poses in the Google Earth dataset. In this dataset, an object's pose is determined by the angle of rotation on the image plane, illustrated as a rotated bounding box. Best viewed in color.

a pure cyclic orthogonal transformation. The different properties that the image operator must have will now be analysed in more detail.

6.5.1.1 *Linearity*

Many common image transformations, such as rotation or scale, are implemented by nearest-neighbor or bilinear interpolation. For a fixed amount of rotation or scale, these functions are linear functions in the input pixels, i.e. each output pixel is a fixed linear combination of some of the input pixels. As such, they fulfill the linearity requirement.

6.5.1.2 *Orthogonality*

For an operator to be orthogonal, it must preserve the L^2 norm of its inputs. At the expense of introducing some non-linearity, we simply renormalize each virtual sample to have the same norm as the original sample, which seems to work well in practice (Section 6.6).

6.5.1.3 *Cyclicity*

We conducted some experiments with planar rotation on aerial imagery (Section 6.6.1). A rotation of $360/s$ degrees is cyclic with period s. In the future, we plan to experiment with non-cyclic operators (similar to how cyclic shifts were used to approximate image translation in Chapters 4 and 5).

Figure 6.5: Example detections and visualization of estimated poses in the TUD-Crossing dataset. The pose is determined by the estimated phase of the walk cycle of a pedestrian, illustrated using a wire-frame model. Best viewed in color.

6.5.2 *Natural transformations*

Another interesting possibility is to use pose annotations to create a structured data matrix. This data-driven approach allows us to consider more complicated transformations than those associated with virtual samples. Given s views of n objects under different poses, we can build the $m \times n \times s$ data matrix X and use the same methodology as before. In Section 6.6 we describe experiments with the walk cycle of pedestrians, and out-of-plane rotations of cars in street scenes. These transformations are cyclic, though highly non-linear, and we use the same renormalization as in Section 6.5.1.

6.5.3 *Negative samples*

One subtle aspect is how to obtain a structured data matrix from negative samples. This is simple for virtual transformations, but not for natural transformations. For example, with planar rotation we can easily generate rotated negative samples with arbitrary poses. However, the same operation with walk cycles of pedestrians is not defined. How do we advance the walk cycle of a non-pedestrian? As a

DATASET	METHOD	CLASSIFIER	TIME (S)	AP	POSE
Google Earth	Proposed	SVR	4.5	73.0	9.4
		RR	3.7	71.4	10.0
	Standard	SVR	130.7	73.2	9.8
		RR	399.3	72.7	10.3

Table 6.1: Experimental results for pose detectors trained with Support Vector Regression (SVR) and Ridge Regression (RR), on the Google Earth dataset. The table shows the total training time, Average Precision (AP) and pose error (both in percentage).

pragmatic solution, we consider that negative samples are unaffected by natural transformations, so a negative sample is constant for all s poses. Because the DFT of a constant signal is 0, except for the DC value (the first frequency), we can ignore untransformed negative samples in all subproblems for $p \neq 1$ (Eq. 6.10). This simple observation can result in significant computational savings.

6.6 EXPERIMENTS

To demonstrate the generality of the proposed model, we conducted object detection and pose estimation experiments on 3 widely different settings, which will be described shortly. We implemented a detector based on Histogram of Oriented Gradients (HOG) templates [30] with multiple components [44]. This framework forms the basis on which several recent advances in object detection are built [88, 47, 44]. The baseline algorithm independently trains s classifiers (components), one per pose, enabling pose-invariant object detection and pose prediction (Eq. 6.5). Components are then calibrated, as usual for detectors with multiple components [44, 88]. The proposed method does not require any ad-hoc calibration, since the components are jointly trained and related by the orthogonal matrix Q, which preserves their L^2 norm.

For the performance evaluation, we use the same procedure as in Section 5.6.1. We measure average precision (AP) and pose error (as e_{pose}/s, where e_{pose} is the discretized pose difference, taking wrap-around into account). We tested two variants of each method, trained with both RR and SVR. Although parallelization is trivial, here we report timings for single-core implementations, which accurately reflect the total CPU load.

6.6.1 *Planar rotation in aerial images (Google Earth)*

Our first test will be on a car detection task on aerial imagery [60], which has been used in several works that deal with planar rotation [119, 81]. We annotated the orientations of 697 objects over half the 30 images of the dataset. The first 7 annotated images were used for training, and the remaining 8 for validation. We created a structured data matrix X by augmenting each sample with 30 virtual samples, using 12° rotations. A visualization of trained weights is shown in Fig. 6.3 and Appendix B. Experimental results are presented in Table 6.1. Recall that our primary goal is to demonstrate faster training, *not* to improve detection performance, which is reflected in the results. Nevertheless, the two proposed fast Fourier algorithms are 29 to 107$\times$ faster than the baseline algorithms.

6.6.2 *Walk cycle of pedestrians (TUD-Campus and TUD-Crossing)*

We can consider a walking pedestrian to undergo a cyclic non-rigid deformation, with each period corresponding to one step. Because this transformation is time-dependent, we can learn it from video data. We used TUD-Campus for training and TUD-Crossing for testing (see Fig. 6.5) [4]. We annotated a key pose in all 272 frames, so that the images of a pedestrian between two key poses represent a whole walk cycle. Sampling 10 images per walk cycle (corresponding to 10 poses), we obtained 10 sample groups for training, for a total of 100 samples.

From Table 6.2, the proposed algorithms seem to slightly outperform the baseline, showing that these non-rigid deformations can be accurately accounted for. However, they are over 2 orders of magnitude faster. In addition to the speed benefits observed in Section 6.6.1, another factor at play is that for natural transformations we can ignore the negative samples in $s - 1$ of the subproblems (Section 6.5.3), whereas the baseline algorithms must consider them when training each of the s components.

6.6.3 *Out-of-plane rotations of cars in street scenes (KITTI)*

For our final experiment, we will attempt to demonstrate that the speed advantage of our method still holds for difficult out-of-plane rotations. We chose the very recent KITTI benchmark [48], which includes an object detection set of 7481 images of street scenes (see Fig. 6.6). The facing angle of cars (along the vertical axis) is provided,

DATASET	METHOD	CLASSIFIER	TIME (S)	AP	POSE
TUD Campus / Crossing	Proposed	SVR	0.1	81.5	9.3
		RR	0.08	82.2	8.9
	Standard	SVR	40.5	80.2	9.5
		RR	45.8	81.6	9.4

Table 6.2: Experimental results for pose detectors on the TUD Campus/Crossing dataset. The columns are as in Table 6.1.

DATASET	METHOD	CLASSIFIER	TIME (S)	AP	POSE
KITTI	Proposed	SVR	15.0	53.5	14.9
		RR	15.5	53.4	15.0
	Standard	SVR	454.2	56.5	13.8
		RR	229.6	54.5	14.0

Table 6.3: Experimental results for pose detectors on the KITTI dataset. The columns are as in Table 6.1.

which we bin into 15 discrete poses. We performed an 80-20% train-test split of the images, considering cars of "moderate" difficulty [48], and obtained 73 sample groups for training with 15 poses each (for a total of 1095 samples).

Table 6.3 shows that the proposed method achieves competitive performance, but with a dramatically lower computational cost. The results agree with the intuition that out-of-plane rotations strain the assumptions of linearity and orthogonality, since they result in large deformations of the object. Nevertheless, the ability to learn a useful model under such adverse conditions shows great promise.

6.7 CONCLUSIONS

In this chapter, we derived new closed-form formulas to quickly train several pose classifiers at once, and take advantage of the structure in datasets with pose annotation or virtual samples. The proposed implicit transformation model seems to be surprisingly expressive, and in future work we would like to experiment with other transformations, including non-cyclic. Other interesting directions include larger-scale variants and the composition of multiple transformations.

Figure 6.6: Example detections and visualization of estimated poses in the KITTI dataset. The pose is determined by the facing angle of a car along the vertical axis, representing an out-of-plane rotation. An illustration of the predicted pose is presented as a rotated 3D bounding box. Best viewed in color.

Algorithm 6.1 Matlab code for fast Fourier training of several pose detectors. X contains sample groups, with s transformed samples each (e.g. rotations of the same object). Note that the transformation should be cyclic. The samples within a group must also have the same Euclidean norm.

Inputs:

- X ($m \times n \times s$ data matrix with m features, n sample groups and s transformations)
- Y ($n \times s$ labels matrix)
- λ (scalar, regularization parameter)
- `regression` (a dual complex regression – Alg. 6.2 or Alg. 6.3)

Output:

- W ($m \times s$ weights matrix – one pose detector per column)

```
g = zeros(n, n, s);
for p = 1:s
  g(:,:,p) = X(:,:,1).' * X(:,:,p);              %Eq. 6.11
end
X = fft(X, [], 3);
g = fft(g, [], 3);
Y = conj(fft(Y, [], 2));
for f = 1:s                                      %Eq. 6.10
  W(:,f) = X(:,:,f) * regression(g(:,:,f), Y(:,f),lambda);
end
W = real(ifft(W, [], 2));
```

Algorithm 6.2 Complex-valued Dual Ridge Regression (can replace `regression` in Alg. 6.1).

```
function alphas = regression(G, y, lambda)
  alphas = (G + lambda * eye(size(G,1))) \ y;     %Eq. 6.2
end
```

Algorithm 6.3 Complex-valued Dual Support Vector Regression (can replace `regression` in Alg. 6.1). The `real_svr` function can be any off-the-shelf SVR solver that accepts a "custom kernel matrix" (Gram matrix), e.g. `libsvm` [23].

```
function alphas = svr(G, y, lambda)
  G = [real(G), imag(G).'; imag(G), real(G)];    %Eq. A.64
  y = [real(y); imag(y)];
  alphas = real_svr(G, y, lambda);
  alphas = alphas(1:end/2) + 1i * alphas(end/2+1:end);
end
```

7

CONCLUSIONS

In the course of this thesis, we have explored how cyclic shifts, and related models of image transformations, affect the learning process of various discriminative algorithms. This represents a significant departure from previous works, which usually regard the input data as unstructured and independently drawn from an unknown distribution. Far from this implicit assumption, we have found that such an analytical model can bring order to an otherwise difficult problem, and manifest itself in the emergence of a circulant structure.

As a side-effect of this exploration of circulant structures, we uncovered an explicit link between Fourier-domain solutions, widely used in the signal processing community, and several machine learning algorithms, which are more commonly used in computer vision. These two approaches to image recognition have evolved side-by-side over the years, under very different views. Circulant matrices thus cast an illuminating new perspective on where they overlap. This creates fertile ground for devising new techniques that take the best of both worlds – the speed of Fourier techniques and the generalizability of machine learning. This thesis shows several applications, but the two fields are so vast that it surely does not exhaust all the possibilities.

It is interesting to note that the analysis of the circulant structure can actually simplify learning problems. The Kernelized Correlation Filter proposed in Chapter 4 requires only the Fast Fourier Transform and element-wise operations, while a general kernel regression would require a more sophisticated solver. Likewise, the general regression problems considered in Chapters 5 and 6 are broken down into smaller regression problems, one for each Fourier frequency, without changing the nature of the underlying regression algorithm. It can be concluded that, though the analysis may involve some non-trivial steps, accounting for the regularities of the data can lead to simplified algorithms, not more complicated ones.

This simplification, along with the expected computational benefits, does not necessarily entail the sacrifice of recognition accuracy. We were able to obtain state-of-the-art results in tracking (Chapter 4), detection (Chapter 5), and pose estimation (Chapter 6). In order to

aid reproducibility, free open-source implementations of these algorithms have been made available to the community[1].

Another conclusion is that, under the right conditions, the Fourier-domain solutions obtained by exploiting circulant data can actually give a significant performance boost. This is the case of data-starved settings, where we have to work with very few examples, such as tracking (Section 4.7) and small detection datasets like ETHZ Shapes (Section 5.6.3). Enriching the dataset with virtual samples (cyclic shifts) can make up for the shortcomings of the dataset, and provide valuable information that a standard learning algorithm does not consider. Cyclic shifts can thus be seen as a strong regularizer, capable of encoding prior knowledge that is common to many computer vision tasks.

7.1 DIRECTIONS FOR FUTURE WORK

There are several ideas that could not be pursued within the scope of this thesis, but which are very promising directions for further investigation. Some of them have already been explored, based on the preliminary publications made over the course of this thesis.

An interesting direction is to relax the assumption of periodic boundaries, which would make for a more accurate model of image translation. A data matrix obtained from (non-cyclically) shifted samples becomes a Toeplitz matrix [56], which is a generalization of a circulant matrix. Although Toeplitz matrices are not invertible in closed form, Valmadre et al. [136] have proposed an iterative method that takes advantage of the close relationship between circulant matrices and Toeplitz matrices. Exploiting the Toeplitz structure provides better performance with only a modest speed penalty, which is still much faster than standard learning. Using the same structure, they also show that the statistics of negative images can be gathered once and then re-used to train different detectors, even when they have different sizes [136].

The proposed methods address the classical inefficiency of virtual samples [33]. Intuitively, assigning the same label to virtual samples obtained by a transformation (e.g. rotated images) should make a classifier invariant to that transformation. There is some early work establishing this connection [98], but much more can be done to put it on a stronger formal basis. Another interesting avenue for theoretical development is to frame cyclic shifts and circulant matrices in the language of Lie groups, which naturally describe continuous

1 http://www.isr.uc.pt/~henriques/

transformations and invariances [94]. There are interesting parallels between them, since both study the successive application of a transformation matrix through a matrix power (e.g. Eq. 3.4, Eq. 6.1). The works on image transformations with Lie group theory can be seen as a continuous analogue of the methodology developed in this thesis [94]. Cohen and Welling [26] have recently shown that the Continuous Fourier Transform diagonalizes cyclically shifted samples, when the shifts are uniformly distributed, which can be seen as a continuous analogue for the settings of Chapter 5. Such an analogy between the discrete and continuous cases can bear further insights into the fundamental properties of image transformations.

There are other areas within computer vision that may benefit from the inclusion of circulant data, but were not explored here. In video streams, virtual samples can be obtained by time translation, which can also be modeled with cyclic shifts, but along the time axis. Revaud et al. [111] have proposed such a method, significantly improving the state-of-the-art of video retrieval. Another interesting direction is to explore these ideas within fine-grained categorization [149] and one-shot learning [43], settings where very few samples are available, and that can benefit from the strong regularization properties of Fourier training.

Finally, a very promising avenue of research is to study the influence of image transformations on deep neural networks [76]. Circulant matrices have recently been proposed in this context as a fast linear layer [25], enjoying better performance and regularization than standard linear layers. It remains to be explored, however, how to exploit the regularities of the data in this setting, similarly to what has been proposed in this thesis. Such advances could hold the key for vastly more efficient deep learning, which is currently both computationally expensive and data-hungry.

A

PROOFS AND IMPLEMENTATION DETAILS

A.1 KERNELIZED CORRELATION FILTERS

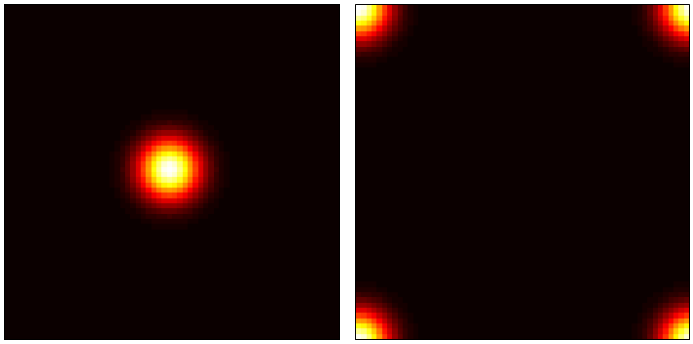
A.1.1 *Implementation details*

As is standard with correlation filters, the input patches (either raw pixels or extracted feature channels) are weighted by a cosine window, which smoothly removes discontinuities at the image boundaries caused by the cyclic assumption [11, 83]. The tracked region has 2.5 times the size of the target, to provide some context and additional negative samples.

Recall that the training samples consist of shifts of a base sample, so we must specify a regression target for each one in $\mathbf{y}$. The regression targets $\mathbf{y}$ simply follow a Gaussian function, which takes a value of 1 for a centered target, and smoothly decays to 0 for any other shifts, according to a spatial bandwidth δ. Gaussian targets are smoother than binary labels, and have the benefit of reducing ringing artifacts in the Fourier domain [83].

A subtle issue is determining which element of $\mathbf{y}$ is the regression target for the centered sample, on which we will center the Gaussian function. Although intuitively it may seem to be the middle of the output plane (Fig. A.1-a), it turns out that the correct choice is the top-left element (Fig. A.1-b). The explanation is that, after computing a cross-correlation between two images in the Fourier domain and converting back to the spatial domain, it is the top-left element of the result that corresponds to a shift of zero [83]. Of course, since we always deal with cyclic signals, the peak of the Gaussian function must wrap around from the top-left corner to the other corners, as can be seen in Fig. A.1-b. Placing the Gaussian peak in the middle of the regression target is common in some filter implementations, and leads the correlation output to be unnecessarily shifted by half a window, which must be corrected post-hoc[1].

1 This is usually done by switching the quadrants of the output, e.g. with the Matlab built-in function `fftshift`. It has the same effect as shifting Fig. A.1-b to look like Fig. A.1-a.



(a) (b)

Figure A.1: Regression targets $\mathbf{y}$, following a Gaussian function with spatial bandwidth s (white indicates a value of 1, black a value of 0). (a) Placing the peak in the middle will unnecessarily cause the detection output to be shifted by half a window (discussed in Section A.1.1). (b) Placing the peak at the top-left element (and wrapping around) correctly centers the detection output.

A.1.2 *Proof of Theorem 4*

Under the theorem's assumption that $\kappa(\mathbf{x}, \mathbf{x}') = \kappa(M\mathbf{x}, M\mathbf{x}')$, for any permutation matrix M, then

$$\begin{aligned} K_{ij} &= \kappa(P^i\mathbf{x},\ P^j\mathbf{x}) \\ &= \kappa(P^{-i}P^i\mathbf{x},\ P^{-i}P^j\mathbf{x}). \end{aligned}$$

Using known properties of permutation matrices, this reduces to

$$K_{ij} = \kappa(\mathbf{x},\ P^{j-i}\mathbf{x}). \tag{A.1}$$

By the cyclic nature of P, it repeats every sth power, i.e. $P^s = P^0$. As such, Eq. A.1 is equivalent to

$$K_{ij} = \kappa(\mathbf{x},\ P^{(j-i) \bmod s}\,\mathbf{x}), \tag{A.2}$$

where mod is the modulus operation (remainder of division by s).

We now use the fact the elements of a circulant matrix $X = C(\mathbf{x})$ (Eq. 3.1) satisfy

$$X_{ij} = x_{((j-i) \bmod s)+1}, \tag{A.3}$$

that is, a matrix is circulant if its elements only depend on $(j - i)$ mod s. It is easy to check that this condition is satisfied by Eq. 3.1, and in fact it is often used as the definition of a circulant matrix [56].

Because K_{ij} also depends on $(j-i)$ mod s, we must conclude that K is circulant as well, finishing our proof.

A.1.3 *Kernel Ridge Regression with Circulant data*

We start by replacing $K = C(\mathbf{k^{xx}})$ in the formula for Kernel Ridge Regression, Eq. 4.1, and diagonalizing it

$$\begin{aligned}\boldsymbol{\alpha} &= \left(C(\mathbf{k^{xx}}) + \lambda I\right)^{-1}\mathbf{y} \\ &= \left(U\text{diag}\left(\hat{\mathbf{k}}^{\mathbf{xx}}\right)U^* + \lambda I\right)^{-1}\mathbf{y}.\end{aligned}$$

By simple linear algebra, and the unitarity of U (Section 3.4),

$$\begin{aligned}\boldsymbol{\alpha} &= \left(U\text{diag}\left(\hat{\mathbf{k}}^{\mathbf{xx}}\right)U^* + \lambda U I U^*\right)^{-1}\mathbf{y} \\ &= U\text{diag}\left(\hat{\mathbf{k}}^{\mathbf{xx}} + \lambda\right)^{-1}U^*\mathbf{y},\end{aligned}$$

which is equivalent to

$$U^*\boldsymbol{\alpha} = \text{diag}\left(\hat{\mathbf{k}}^{\mathbf{xx}} + \lambda\right)^{-1}U^*\mathbf{y}. \tag{A.4}$$

Using U as the unitary DFT operator, we can change the vectors to the Fourier domain, and complex-conjugate the whole expression to obtain

$$\hat{\boldsymbol{\alpha}} = \text{diag}\left(\frac{1}{\hat{\mathbf{k}}^{\mathbf{xx}} + \lambda}\right)\hat{\mathbf{y}}. \tag{A.5}$$

Finally, because the product of a diagonal matrix and a vector is simply their element-wise product,

$$\hat{\boldsymbol{\alpha}} = \frac{\hat{\mathbf{y}}}{\hat{\mathbf{k}}^{\mathbf{xx}} + \lambda}. \tag{A.6}$$

A.1.4 *Derivation of fast detection formula*

To diagonalize Eq. 4.6, we use the same properties as in the previous section. We have

$$\begin{aligned}\mathbf{f}(\mathbf{z}) &= \left(C\left(\mathbf{k}^{\mathbf{xz}}\right)\right)^T \boldsymbol{\alpha} \\ &= \left(U\mathrm{diag}\left(\hat{\mathbf{k}}^{\mathbf{xz}}\right) U^*\right)^T \boldsymbol{\alpha} \\ &= U^*\mathrm{diag}\left(\hat{\mathbf{k}}^{\mathbf{xz}}\right) U\boldsymbol{\alpha}\end{aligned}$$

which is equivalent to

$$U\mathbf{f}(\mathbf{z}) = \mathrm{diag}\left(\hat{\mathbf{k}}^{\mathbf{xz}}\right) U\boldsymbol{\alpha}. \tag{A.7}$$

Replicating the same final steps from Section A.1.3,

$$\hat{\mathbf{f}}(\mathbf{z}) = \hat{\mathbf{k}}^{\mathbf{xz}} \odot \hat{\boldsymbol{\alpha}}. \tag{A.8}$$

A.1.5 *Linear Ridge Regression with Circulant data*

This is a more detailed version of the steps from Section 3.8. It is very similar to the kernel case. We begin by replacing Eq. 3.22 in the formula for Ridge Regression, Eq. 3.19.

$$\mathbf{w} = \left(U\mathrm{diag}\left(\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}}\right) U^* + \lambda I\right)^{-1} X^T\mathbf{y} \tag{A.9}$$

By simple algebra, and the unitarity of U, we have

$$\begin{aligned}\mathbf{w} &= \left(U\mathrm{diag}\left(\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}}\right) U^* + \lambda U I U^*\right)^{-1} X^T\mathbf{y} \\ &= \left(U\mathrm{diag}\left(\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}} + \lambda\right)^{-1} U^*\right) X^T\mathbf{y} \\ &= U\mathrm{diag}\left(\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}} + \lambda\right)^{-1} U^* U\mathrm{diag}\left(\hat{\mathbf{x}}^*\right) U^*\mathbf{y} \\ &= U\mathrm{diag}\left(\frac{\hat{\mathbf{x}}^*}{\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}} + \lambda}\right) U^*\mathbf{y}.\end{aligned}$$

Then, this is equivalent to

$$U^*\mathbf{w} = \mathrm{diag}\left(\frac{\hat{\mathbf{x}}^*}{\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}} + \lambda}\right) U^*\mathbf{y}. \tag{A.10}$$

Since for any vector $U\mathbf{z} = \frac{1}{\sqrt{s}}\hat{\mathbf{z}}$, and likewise $U^*\mathbf{z} = \frac{1}{\sqrt{s}}\hat{\mathbf{z}}^*$,

$$\hat{\mathbf{w}} = \mathrm{diag}\left(\frac{\hat{\mathbf{x}}^*}{\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}} + \lambda}\right) \hat{\mathbf{y}}. \tag{A.11}$$

We may go one step further, since the product of a diagonal matrix and a vector is just their element-wise product:

$$\hat{\mathbf{w}} = \frac{\hat{\mathbf{x}}^* \odot \hat{\mathbf{y}}}{\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}} + \lambda}. \tag{A.12}$$

A.1.6 *MOSSE filter*

The only difference between Eq. 3.24 and the MOSSE filter [11] is that the latter minimizes the error over (cyclic shifts of) multiple base samples $\mathbf{x}_i$, while Eq. 3.24 is defined for a single base sample $\mathbf{x}$. This was done for clarity of presentation, and the general case is easily derived. Note also that MOSSE does not support multiple channels, which we do through our dual formulation.[2]

The cyclic shifts of each base sample $\mathbf{x}_i$ can be expressed in a circulant matrix X_i. Then, replacing the data matrix $X' = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \end{bmatrix}$ in Eq. 2.4 results in

$$\mathbf{w} = \sum_j \left(\sum_i X_i^H X_i + \lambda I \right)^{-1} X_j^H \mathbf{y}, \tag{A.13}$$

by direct application of the rule for products of block matrices. Factoring the bracketed expression,

$$\mathbf{w} = \left(\sum_i X_i^H X_i + \lambda I \right)^{-1} \left(\sum_i X_i^H \right) \mathbf{y}. \tag{A.14}$$

Eq. A.14 looks exactly like Eq. 2.4, except for the sums. It is then trivial to follow the same steps as in Section A.1.5 to diagonalize it, and obtain the filter equation

$$\hat{\mathbf{w}} = \frac{\sum_i \hat{\mathbf{x}}_i^* \odot \hat{\mathbf{y}}}{\sum_i \hat{\mathbf{x}}_i^* \odot \hat{\mathbf{x}}_i + \lambda}. \tag{A.15}$$

2 Note that by convention, Bolme et al. [11] consider a template $\mathbf{w}^*$ that is the complex-conjugated version of ours.

A.2 COMMUTATIONS AND BLOCK-MATRICES

Manipulating block matrices can lead to burdensome notation, and theorems built for standard matrices do not always apply to them. In order to alleviate this issue, we introduce the notion of the *commutation matrix*. Although it was originally developed for problems involving Kronecker products [85], we will show that its usefulness extends to much more general problems involving block matrices.

Define $\mathrm{vec}\,(A)$ as the operator that takes the columns of a $m \times n$ matrix A, and stacks them into a $mn \times 1$ vector (*vectorization*). For example, if A is 2×3, we have

$$\mathrm{vec}\,(A) = \mathrm{vec}\left(\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}\right) = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{12} \\ a_{22} \\ a_{13} \\ a_{23} \end{bmatrix} \tag{A.16}$$

The commutation matrix R permutes the elements of a vectorized matrix A to obtain its transpose A^T [85]. Formally,

$$R\mathrm{vec}\,(A) = \mathrm{vec}\left(A^T\right). \tag{A.17}$$

Note that there is a *uniquely defined* commutation matrix R that does not depend on A, but only on the size of A [85]. It is the unique permutation matrix that obeys Eq. A.17.

As a concrete example, if A is 2×3,

$$\overbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}^{R} \overbrace{\begin{bmatrix} a_{11} \\ a_{21} \\ a_{12} \\ a_{22} \\ a_{13} \\ a_{23} \end{bmatrix}}^{\mathrm{vec}(A)} = \overbrace{\begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix}}^{\mathrm{vec}(A^T)}. \tag{A.18}$$

Notice how the elements were permuted in Eq. A.18. We can view $\mathrm{vec}\,(A)$ as a $mn \times 1$ block-matrix with n blocks, where each block is a column of A. The right-hand side of Eq. A.18 contains m blocks, where each block is a row of A instead. The commutation matrix stacks the first element of every block, then the second element of every block, and so on. This characterization is important, and will be very useful in the remainder of this section.

Because R is a permutation matrix, it is orthogonal, so its inverse can be obtained easily: $R^T = R^{-1}$. Applying a permutation matrix to an arbitrary matrix on the left will permute its rows, while doing so on the right permutes its columns. In this way we can use R to permute the blocks of a block-matrix, in the same order as the blocks of the vector in Eq. A.18. As a few examples, that can be verified quickly using Eq. A.17, we have the following.

A.2.1 *Block-diagonal matrices and matrices with diagonal blocks*

A $mn \times mn$ block-diagonal matrix (left of Eq. A.19) can be transformed into a $mn \times mn$ block matrix, where each block is diagonal (right of Eq. A.19), by application of the permutation R:

$$\begin{bmatrix} A(1) & & \\ & \ddots & \\ & & A(n) \end{bmatrix} = R \begin{bmatrix} D(1,1) & \cdots & D(1,m) \\ \vdots & \ddots & \vdots \\ D(m,1) & \cdots & D(m,m) \end{bmatrix} R^T, \tag{A.19}$$

where the $A(i)$ are arbitrary $m \times m$ matrices, and each $D(i,j)$ is a $n \times n$ diagonal matrix, where the kth diagonal element is $A_{ij}(k)$.

A.2.2 *Block-circulant matrices and matrices with circulant blocks*

Similarly to the above, a block-circulant matrix (left of Eq. A.20) can be transformed into a block matrix, where each block is circulant (right of Eq. A.20), by the same application of the permutation R:

$$\begin{bmatrix} A(1) & A(2) & A(3) & \cdots \\ A(n) & A(1) & A(2) & \cdots \\ A(n-1) & A(n) & A(1) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} = R \begin{bmatrix} C(1,1) & \cdots & C(1,m) \\ \vdots & \ddots & \vdots \\ C(m,1) & \cdots & C(m,m) \end{bmatrix} R^T, \tag{A.20}$$

where the $A(i)$ are arbitrary $m \times m$ matrices, and each $C(i,j)$ is a $n \times n$ circulant matrix, where the kth element of its first row is $A_{ij}(k)$.

Hopefully a pattern starts to emerge: because R only permutes the rows and columns of a matrix, in the manner described at the beggining of the section, matrix properties can be carried over from the block-level to the level of individual blocks. Specifically, properties that vary with whole blocks of matrices (such as being block-diagonal or block-circulant) now describe individual elements (diagonal or cir-

culant), because these elements are simply rearranged by R from a block-wise order to a block-element order.

A.2.3 *Commutation of Kronecker products*

Kronecker products create block matrices, and given the previous considerations it is not surprising that the commutation matrix has an interesting effect on Kronecker products.

The Kronecker product of two matrices is a block matrix, defined as [95]

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}. \tag{A.21}$$

The commutation matrix can then be used to exchange the block order, essentially commuting A and B [85]

$$B \otimes A = R\,(A \otimes B)\,R^T. \tag{A.22}$$

Eq. A.19-A.22 provide several relations between blocks and the commutation matrix. Together with its orthogonality ($R^T = R^{-1}$), we have a diverse set of tools to manipulate block matrices, which are very useful for Chapters 5 and 6.

A.3 CIRCULANT DECOMPOSITION OF LARGE-SCALE PROBLEMS

A.3.1 *Separability of learning problems*

For the exact case, we will assume D can be written in terms of dot-products, as mentioned. Keeping in mind that its arguments are scalar,

$$D\left(\alpha_i, y_i\right) = d_1 \left\|\alpha_i\right\|^2 + d_2 \left\|y_i\right\|^2 + d_3 \alpha_i^H y_i, \tag{A.23}$$

where we defined the new constants d_1, d_2 and d_3. Then, Eq. 5.12 becomes

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^H G \boldsymbol{\alpha} + \sum_{i=1}^{n} \left(d_1 \left\|\alpha_i\right\|^2 + d_2 \left\|y_i\right\|^2 + d_3 \alpha_i^H y_i\right) \\ =&\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^H G \boldsymbol{\alpha} + d_1 \left\|\boldsymbol{\alpha}\right\|^2 + d_2 \left\|\mathbf{y}\right\|^2 + d_3 \boldsymbol{\alpha}^H \mathbf{y}, \end{aligned} \tag{A.24}$$

where the Hermitian transpose $(\cdot)^H$ is used instead of $(\cdot)^T$. Notice that all the learning algorithms we consider use the Hermitian transpose when extended from reals to complex numbers, which simplifies some expressions, and has no effect if the quantities are indeed real.

Performing the substitutions $G = V\hat{G}V^*$, $\boldsymbol{\alpha} = V\hat{\boldsymbol{\alpha}}$ and $\mathbf{y} = V\hat{\mathbf{y}}$, by unitarity the V's cancel out, and we are left with

$$\min_{\hat{\boldsymbol{\alpha}}} \frac{1}{2}\hat{\boldsymbol{\alpha}}^H \hat{G} \hat{\boldsymbol{\alpha}} + d_1 \left\|\hat{\boldsymbol{\alpha}}\right\|^2 + d_2 \left\|\hat{\mathbf{y}}\right\|^2 + d_3 \hat{\boldsymbol{\alpha}}^H \hat{\mathbf{y}}. \tag{A.25}$$

This may seem like a trivial change, but $\hat{G}$ is block-diagonal, while G is not. Recall Eq. 5.9,

$$\hat{G} = \begin{bmatrix} \hat{G}(1) & & & \\ & \hat{G}(2) & & \\ & & \ddots & \\ & & & \hat{G}(s) \end{bmatrix}, \tag{A.26}$$

where each block is $n \times n$. To correspond to the same structure, split the vectors $\hat{\boldsymbol{\alpha}}$ and $\hat{\mathbf{y}}$ into s blocks of size $n \times 1$,

$$\hat{\boldsymbol{\alpha}} = \begin{bmatrix} \hat{\boldsymbol{\alpha}}_1 \\ \hat{\boldsymbol{\alpha}}_2 \\ \vdots \\ \hat{\boldsymbol{\alpha}}_s \end{bmatrix}, \quad \hat{\mathbf{y}} = \begin{bmatrix} \hat{\mathbf{y}}_1 \\ \hat{\mathbf{y}}_2 \\ \vdots \\ \hat{\mathbf{y}}_s \end{bmatrix}, \tag{A.27}$$

where each block $\hat{\boldsymbol{\alpha}}_f$ and $\hat{\mathbf{y}}_f$ has n elements $\hat{\alpha}_{fi}$ and $\hat{y}_{fi}$, respectively. Now, since the rules for matrix products are the same as for block-matrix products, direct computation yields

$$\begin{aligned}&\min_{\hat{\boldsymbol{\alpha}}} \sum_{f=1}^{s} \left(\frac{1}{2} \hat{\boldsymbol{\alpha}}_f^H \hat{G}_f \hat{\boldsymbol{\alpha}}_f + d_1 \left\| \hat{\boldsymbol{\alpha}}_f \right\|^2 + d_2 \left\| \hat{\mathbf{y}}_f \right\|^2 + d_3 \hat{\boldsymbol{\alpha}}_f^H \hat{\mathbf{y}}_f \right) \\ =&\min_{\hat{\boldsymbol{\alpha}}} \sum_{f=1}^{s} \left(\frac{1}{2} \hat{\boldsymbol{\alpha}}_f^H \hat{G}_f \hat{\boldsymbol{\alpha}}_f + \sum_{i=1}^{n} D\left(\hat{\alpha}_{fi}, \hat{y}_{fi}\right) \right). \end{aligned} \tag{A.28}$$

Notice that Eq. A.28 is a sum of objective functions over different (and non-interacting) optimization variables, $\hat{\boldsymbol{\alpha}}_f$. As such, they can be optimized independently, and Eq. A.28 is equivalent to the s sub-problems,

$$\min_{\hat{\boldsymbol{\alpha}}_f} \frac{1}{2} \hat{\boldsymbol{\alpha}}_f^H \hat{G}_f \hat{\boldsymbol{\alpha}}_f + \sum_{i=1}^{n} D\left(\hat{\alpha}_{fi}, \hat{y}_{fi}\right), \tag{A.29}$$

for $f = 1, \ldots, s$, as required.

A.3.2 *Transformation matrices that yield exact decompositions*

As an interesting aside, it is possible to characterize the class of matrices V that would yield an *exact* decomposition for most algorithms. Just as unitary matrices preserve the L^2-norm, it is known [79] that generalized permutation matrices (which extend permutation matrices by allowing the non-zero elements to take the values 1 and -1) preserve all L^p-norms, for $p \geq 1$. By restricting V to this class, the decomposition would be exact for algorithms such as the SVR. Even though this result may be useful for other block-diagonalizations, we cannot use it since it is too restrictive for the case of block-circulant matrices.

A.3.3 *Complex Support Vector Regression*

This section deals with the extension of a linear regression problem,

$$\min_{\mathbf{w}} \lambda \left\| \mathbf{w} \right\|^2 + \sum_{j=1}^{n} \left| \mathbf{w}^H \mathbf{x} - y_j \right|_\epsilon, \tag{A.30}$$

to the complex domain, through the extended loss function,

$$\left|\mathbf{w}^H\mathbf{x} - y\right|_\epsilon = \left|\mathrm{Re}\left(\mathbf{w}^H\mathbf{x} - y\right)\right|_\epsilon + \left|\mathrm{Im}\left(\mathbf{w}^H\mathbf{x} - y\right)\right|_\epsilon, \tag{A.31}$$

as mentioned in Section 5.5. Note the Hermitian transpose reduces to the transpose for real arguments. Also, the ϵ-insensitive loss in Eq. A.30 and A.31 can be easily squared, in the case of L^2-SVR, and our result still holds.

In this section we will use i to denote a pure imaginary unit, $i = \sqrt{-1}$. First, decompose all quantities into their real and imaginary components, as

$$\begin{aligned} \mathbf{w} &= \mathbf{w}^R + i\mathbf{w}^I \\ \mathbf{x}_j &= \mathbf{x}_j^R + i\mathbf{x}_j^I \\ y_j &= y_j^R + iy_j^I. \end{aligned} \tag{A.32}$$

Substituting into Eq. A.30, and applying the rules of the complex product,

$$\begin{aligned} \min_{\mathbf{w}} \lambda \left\|\mathbf{w}^R + i\mathbf{w}^I\right\|^2 + \sum_{j=1}^{n} \Big| &\left(\mathbf{w}^R\right)^T \mathbf{x}_j^R + \left(\mathbf{w}^I\right)^T \mathbf{x}_j^I - y_j^R \\ &+ i\left(\left(\mathbf{w}^R\right)^T \mathbf{x}_j^I - \left(\mathbf{w}^I\right)^T \mathbf{x}_j^R - y_j^I\right)\Big|_\epsilon. \end{aligned} \tag{A.33}$$

Expanding the first term, and applying Eq. A.31 to the second,

$$\begin{aligned} \min_{\mathbf{w}} \lambda \left\|\mathbf{w}^R\right\|^2 + \sum_{j=1}^{n} &\left|\left(\mathbf{w}^R\right)^T \mathbf{x}_j^R + \left(\mathbf{w}^I\right)^T \mathbf{x}_j^I - y_j^R\right|_\epsilon \\ +\lambda \left\|\mathbf{w}^I\right\|^2 + \sum_{j=1}^{n} &\left|\left(\mathbf{w}^R\right)^T \mathbf{x}_j^I - \left(\mathbf{w}^I\right)^T \mathbf{x}_j^R - y_j^I\right|_\epsilon. \end{aligned} \tag{A.34}$$

Eq. A.34 shows that the complex SVR is equivalent to an augmented real SVR, with:

1. Double the features, to account for real and imaginary parts of the inputs and weights.
2. Double the samples, to account for the loss function in the real axis and in the imaginary axis (from Eq. A.31).

The augmented real SVR can be written more compactly as

$$\min_{\mathbf{w}} \lambda \left\|\mathbf{w}'\right\|^2 + \sum_{j=1}^{n} \left|\mathbf{w}'^T \mathbf{x}'_j - y'_j\right|_\epsilon , \tag{A.35}$$

with $\mathbf{x}'_j$ the rows of X' and y'_j the elements of $\mathbf{y}'$, defined in terms of the analogous original complex quantities,

$$X' = \begin{bmatrix} \mathrm{Re}\,(X) & \mathrm{Im}\,(X) \\ \mathrm{Im}\,(X) & -\mathrm{Re}\,(X) \end{bmatrix} \tag{A.36}$$

$$\mathbf{y}' = \begin{bmatrix} \mathrm{Re}\,(\mathbf{y}) \\ \mathrm{Im}\,(\mathbf{y}) \end{bmatrix} \tag{A.37}$$

$$\mathbf{w}' = \begin{bmatrix} \mathrm{Re}\,(\mathbf{w}) \\ \mathrm{Im}\,(\mathbf{w}) \end{bmatrix} . \tag{A.38}$$

A.4 GENERAL GEOMETRIC TRANSFORMATIONS

A.4.1 *Proof of Theorem 7*

Though the main claim of the Theorem is the last one, there are also other claims which we will prove in turn.

- *The data matrix X and the uncentered covariance matrix X^TX are not circulant in general.*

This can be demonstrated with a simple counterexample. Consider the following transformation Q, which simply reverses the order of any 3×1 input vector:

$$Q = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{A.39}$$

It is orthogonal and cyclic with period $s = 2$ ($Q^2 = Q^0 = I$).
The corresponding data matrix X (using Eq. 6.1) is

$$X = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_3 & x_2 & x_1 \end{bmatrix}. \tag{A.40}$$

The matrix in Eq. A.40 is not square, and thus cannot be circulant [56].

Another necessary (but not sufficient) condition for a matrix to be circulant is that its diagonal elements are constant [56]. By direct computation, the uncentered covariance matrix X^TX fails this requirement; its diagonal elements are given by $[x_1^2 + x_3^2,\ 2x_2^2,\ x_1^2 + x_3^2]$.

- *The data matrix X and the uncentered covariance matrix X^TX are circulant for $Q = P$.*

This is an earlier result (Sections 3.2 and 3.7).

- *The Gram matrix $G = XX^T$ is circulant.*

We have

$$G_{pr} = (Q^p\mathbf{x})^T Q^r\mathbf{x} = \mathbf{x}^T (Q^p)^{-1} Q^r\mathbf{x} = \mathbf{x}^T Q^{r-p}\mathbf{x} = \mathbf{x}^T Q^{(r-p) \bmod s}\mathbf{x}, \tag{A.41}$$

where the second equivalence is due to orthogonality, and the last one is due to cyclicity. The strict dependence on $(r - p) \bmod s$ implies that G is circulant [56].

A.4.2 *Fast solution in the dual for training with multiple transformed images*

We are given n sample groups, each of them containing s transformed versions of an image $\mathbf{x}_i$. Let us organize the data into n blocks $X(i)$, one per sample group, each block with size $s \times m$:

$$X(i) = C_Q(\mathbf{x}_i), \qquad i = 1, \ldots, n \tag{A.42}$$

The full $ns \times m$ data matrix is obtained by vertical concatenation of all the $X(i)$. We can compute the corresponding Gram matrix easily since it is just the product of two block matrices. It is composed of n^2 blocks, each one of size $s \times s$, defined by

$$G(i,j) = X^T(i)\, X(j), \qquad i,j = 1, \ldots, n. \tag{A.43}$$

The Ridge Regression (RR) problem with an $ns \times ns$ Gram matrix composed of these blocks is given by

$$\begin{bmatrix} \boldsymbol{\alpha}(1) \\ \vdots \\ \boldsymbol{\alpha}(n) \end{bmatrix} = \left(\begin{bmatrix} G(1,1) & \cdots & G(1,n) \\ \vdots & \ddots & \vdots \\ G(n,1) & \cdots & G(n,n) \end{bmatrix} + \lambda I \right)^{-1} \begin{bmatrix} \mathbf{y}(1) \\ \vdots \\ \mathbf{y}(n) \end{bmatrix}, \tag{A.44}$$

where $\boldsymbol{\alpha}(i)$ are $s \times 1$ vectors of solution coefficients, and $\mathbf{y}(i)$ are $s \times 1$ vectors of target labels.

Each block $G(i,j)$ verifies Theorem 7, which means that they are circulant. As such, they are defined by their first row,

$$G(i,j) = C\left(\mathbf{x}_i\, C_Q^T(\mathbf{x}_j)\right). \tag{A.45}$$

We can diagonalize the blocks of the Gram matrix individually, by transforming the problem (block-wise) to the Fourier domain. Eq. A.44 is equivalent to

$$\begin{bmatrix} \hat{\boldsymbol{\alpha}}(1) \\ \vdots \\ \hat{\boldsymbol{\alpha}}(n) \end{bmatrix} = \left(\begin{bmatrix} \hat{G}(1,1) & \cdots & \hat{G}(1,n) \\ \vdots & \ddots & \vdots \\ \hat{G}(n,1) & \cdots & \hat{G}(n,n) \end{bmatrix} + \lambda I \right)^{-1} \begin{bmatrix} \hat{\mathbf{y}}(1) \\ \vdots \\ \hat{\mathbf{y}}(n) \end{bmatrix}, \tag{A.46}$$

with the Fourier-domain variables $\hat{\boldsymbol{\alpha}}(i) = U\boldsymbol{\alpha}(i)$, $\hat{\mathbf{y}}(i) = U\mathbf{y}(i)$, and

$$\hat{G}(i,j) = U^* G(i,j) U, \qquad i,j = 1,\ldots,n, \tag{A.47}$$

The identity I is unaffected by U because the later is unitary.

Since $G(i,j)$ is circulant, $\hat{G}(i,j)$ must be diagonal, i.e.,

$$\hat{G}_{pr}(i,j) = 0, \text{ if } p \neq r. \tag{A.48}$$

We can turn the Gram matrix with diagonal blocks into a block-diagonal matrix by a permutation of its rows and columns. Define s^2 blocks, each one $n \times n$, with elements obtained just by reordering the elements of $\hat{G}(i,j)$:

$$G'_{ij}(p,r) = \hat{G}_{pr}(i,j), \qquad i,j = 1,\ldots,n. \tag{A.49}$$

The two forms offer different views into the same data. $\hat{G}(i,j)$ describes the interactions through pose-space, after fixing two samples i and j. $G'(p,r)$ emphasizes the interactions between pairs of samples, for a given Fourier frequency.

Given Eq. A.48 and Eq. A.49, we know that the off-diagonal $G'(p,r)$ blocks must be zero, i.e.,

$$G'(p,r) = \mathbf{0}, \text{ if } p \neq r, \tag{A.50}$$

with $\mathbf{0}$ denoting an $n \times n$ matrix of zeros. The RR problem in the permuted domain is then

$$\begin{bmatrix} \boldsymbol{\alpha}'(1) \\ \boldsymbol{\alpha}'(2) \\ \vdots \\ \boldsymbol{\alpha}'(s) \end{bmatrix} = \left(\begin{bmatrix} G'(1,1) & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & G'(2,2) & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & G'(s,s) \end{bmatrix} + \lambda I \right)^{-1} \begin{bmatrix} \mathbf{y}'(1) \\ \mathbf{y}'(2) \\ \vdots \\ \mathbf{y}'(s) \end{bmatrix}, \tag{A.51}$$

where $\boldsymbol{\alpha}'_i(p) = \hat{\boldsymbol{\alpha}}_p(i)$ and $\mathbf{y}'_i(p) = \hat{\mathbf{y}}_p(i)$ are the remaining variables under the same permutation.

By direct computation with the rules of block matrices, we obtain

$$\begin{bmatrix} \boldsymbol{\alpha}'(1) \\ \boldsymbol{\alpha}'(2) \\ \vdots \\ \boldsymbol{\alpha}'(s) \end{bmatrix} = \begin{bmatrix} (G'(1,1) + \lambda I)^{-1} \mathbf{y}'(1) \\ (G'(2,2) + \lambda I)^{-1} \mathbf{y}'(2) \\ \vdots \\ (G'(s,s) + \lambda I)^{-1} \mathbf{y}'(s) \end{bmatrix}, \tag{A.52}$$

or more concisely,

$$\boldsymbol{\alpha}'(p) = \left(G'(p,p) + \lambda I\right)^{-1} \mathbf{y}'(p), \qquad p = 1, \ldots, s. \tag{A.53}$$

Note that Eq. A.53 hinges on the earlier definitions of $\boldsymbol{\alpha}'(p)$, $G'(p,p)$ and $\mathbf{y}'(p)$, which are Fourier-transformed and permuted versions of the original quantities.

A.4.3 *Formulation using multi-dimensional arrays*

To make Eq. A.53 more self-contained, we can express it using multi-dimensional arrays, by tracing back the elements of $\boldsymbol{\alpha}'(p)$, $G'(p,p)$ and $\mathbf{y}'(p)$.

Define the $n \times n \times s$ array of unique inner-products $\mathbf{g}$, with elements

$$\mathbf{g}_{ijp} = \mathbf{x}_i^T Q^{p-1} \mathbf{x}_j. \tag{A.54}$$

Also, define the $n \times s$ matrix Y, where the element Y_{ip} is the label of sample image i for pose p.

Then Eq. A.53 can be implemented by taking the DFT of Y along the second dimension and the DFT of $\mathbf{g}$ along the third dimension, i.e.,

$$\hat{Y} = \mathcal{F}_{(2)}\left(Y\right) \tag{A.55}$$

$$\hat{\mathbf{g}} = \mathcal{F}_{(3)}\left(\mathbf{g}\right), \tag{A.56}$$

and computing the $n \times s$ solution in the Fourier domain, $\hat{A}$, with

$$\hat{A}_{\bullet p} = \left(\hat{\mathbf{g}}_{\bullet\bullet p} + \lambda I\right)^{-1} \hat{Y}_{\bullet p}, \qquad p = 1, \ldots, s, \tag{A.57}$$

where $\hat{A}_{\bullet p}$ denotes the pth column from $\hat{A}$ (and similarly for $\hat{Y}$), while $\hat{\mathbf{g}}_{\bullet\bullet p}$ slices the pth subarray (of size $n \times n$) along the third dimension of $\hat{\mathbf{g}}$. For reference, the slicing operator $\bullet$ works the same way as the slicing operator : in Matlab or NumPy.

Note that Eq. A.57 and Eq. A.53 are exactly the same, except with different notation.

We can retrieve the solution from Fourier space by taking the IDFT of $\hat{A}$ along the second dimension,

$$A = \mathcal{F}_{(2)}^{-1}\left(\hat{A}\right). \tag{A.58}$$

The element A_{ip} is the dual coefficient of sample image i for pose p.

A.4.4 *Solution for a single classifer*

Using the data matrix in Eq. A.42 and the solution in the dual from Eq. A.57,

$$\mathbf{w} = \sum_{i=1}^{n} X^T(i)\, A_{i\bullet}\,. \tag{A.59}$$

A.4.5 *Solution for multiple pose classifiers*

For multiple pose classifiers, we have

$$\begin{aligned} W = \left[\ \mathbf{w}_0 \ \middle|\ \cdots \ \middle|\ \mathbf{w}_{s-1}\ \right] &= \sum_{i=1}^{n} X^T(i)\ \left[\ P^0 A_{i\bullet} \ \middle|\ \cdots \ \middle|\ P^{s-1} A_{i\bullet}\ \right] \\ &= \sum_{i=1}^{n} X^T(i) C^T\left(A_{i\bullet}\right), \end{aligned} \tag{A.60}$$

because permuting the rows of the labels Y results in the same permutation being applied to the rows of the solution A. Diagonalizing with U, we obtain

$$W^T = \mathcal{F}^{-1}\left(\sum_{i=1}^{n} \operatorname{diag}\left(\hat{A}^*_{i\bullet}\right) \mathcal{F}\left(X(i)\right)\right), \tag{A.61}$$

where * denotes complex-conjugation. Note that a product by a diagonal matrix on the left simply amounts to multiplying each row with one of the diagonal elements.

If $\hat{X}$ is the $m \times n \times s$ data matrix, Fourier-transformed in the third dimension, we can rewrite Eq. A.61 as

$$\begin{aligned} \hat{W}_{\bullet p} &= \hat{X}_{\bullet\bullet p} \hat{A}^*_{\bullet p} \\ &= \hat{X}_{\bullet\bullet p} \left(\hat{\mathbf{g}}_{\bullet\bullet p} + \lambda I\right)^{-1} \hat{Y}^*_{\bullet p} \end{aligned}$$

for $p = 1, \ldots, s$, and recover W by taking the IDFT over the second dimension.

A.4.6 *Complex-valued Support Vector Regression in the Dual*

We build on the primal solution for complex-valued Support Vector Regression (SVR) given in Appendix A.3.3 (complex-SVR for short). Note that the algorithm that we propose in Chapter 6 encodes each sub-problem as a complex-valued Gram matrix G, not a data matrix X. This requires us to express the complex-SVR in the dual variables instead, differently from Appendix A.3.3.

For conciseness, let the subscripts R and I denote real and imaginary parts, respectively (e.g., $X_R = \mathrm{Re}\,(X)$ and $X_I = \mathrm{Im}\,(X)$). By direct computation, the complex-valued Gram matrix G obtained from a complex-valued $X = X_R + i.X_I$ is

$$G = XX^H = X_R^2 + X_I^2 + i.\left(X_I X_R^T - X_R X_I^T\right) = G_R + i.G_I, \quad \text{(A.62)}$$

where we used i to denote a pure imaginary unity.

Again by direct computation, the real-valued Gram matrix G' obtained from the equivalent Eq. A.36 is

$$G' = X'X'^T = \begin{bmatrix} X_R^2 + X_I^2 & X_R X_I^T - X_I X_R^T \\ X_I X_R^T - X_R X_I^T & X_R^2 + X_I^2 \end{bmatrix} \quad \text{(A.63)}$$

Comparing Eq. A.62 to A.63, we see that

$$G' = \begin{bmatrix} G_R & G_I^T \\ G_I & G_R \end{bmatrix}, \quad \text{(A.64)}$$

and thus we can use Eq. A.64 to express the complex-valued G of a complex-valued SVR with an equivalent real-valued G. A simple implementation is shown in Algorithm 6.3.

B

ADDITIONAL FIGURES

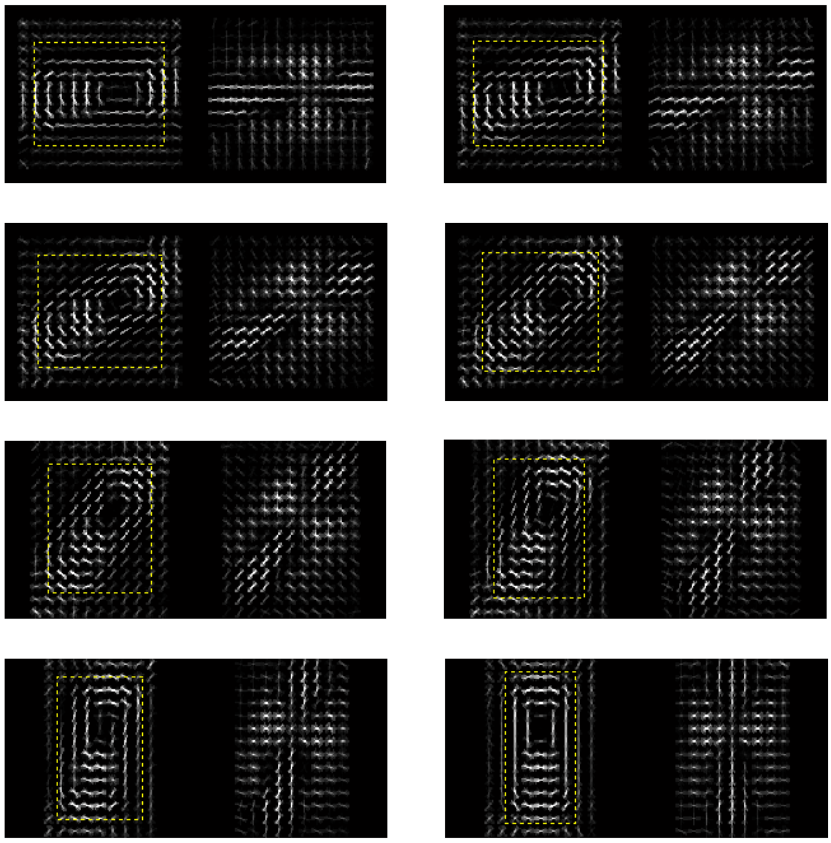


Figure B.1: Visualization of HOG templates, obtained with Fourier training and RR, for the Google Earth dataset. Each template represents a distinct pose across planar rotations. Only the templates from 0º to 90º are shown. Positive weights are shown on the left, negative weights on the right. The tight-fitting bounding box is also displayed as a yellow line. Note that rotated HOG templates cannot be obtained from a single template simply by applying a rotation to the cells – the gradient bins must also adjust their orientation correctly. For more complicated transformations and features, this cannot be done in closed form. The implicit transformation model allows us to bypass this issue, and obtain correct gradient orientations.

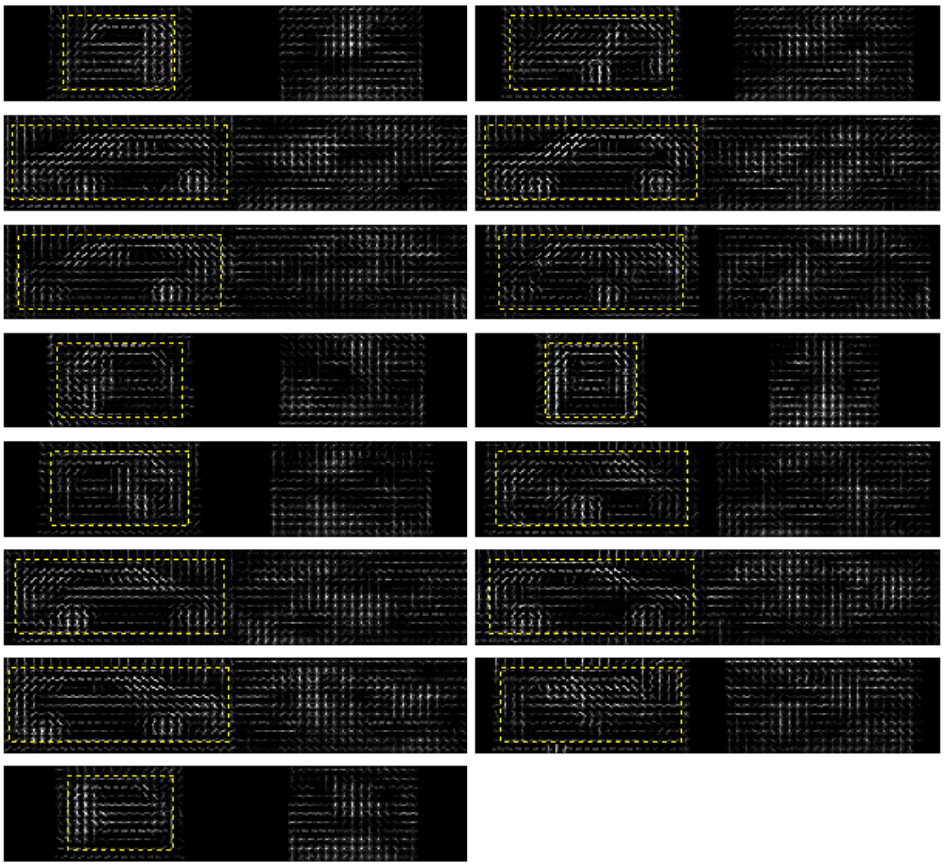


Figure B.2: Visualization of HOG templates, obtained with Fourier training and RR, for the KITTI dataset. Each template represents a distinct pose across out-of-plane rotations (along the vertical axis). Positive weights are shown on the left, negative weights on the right. The tight-fitting bounding box is also displayed as a yellow line.

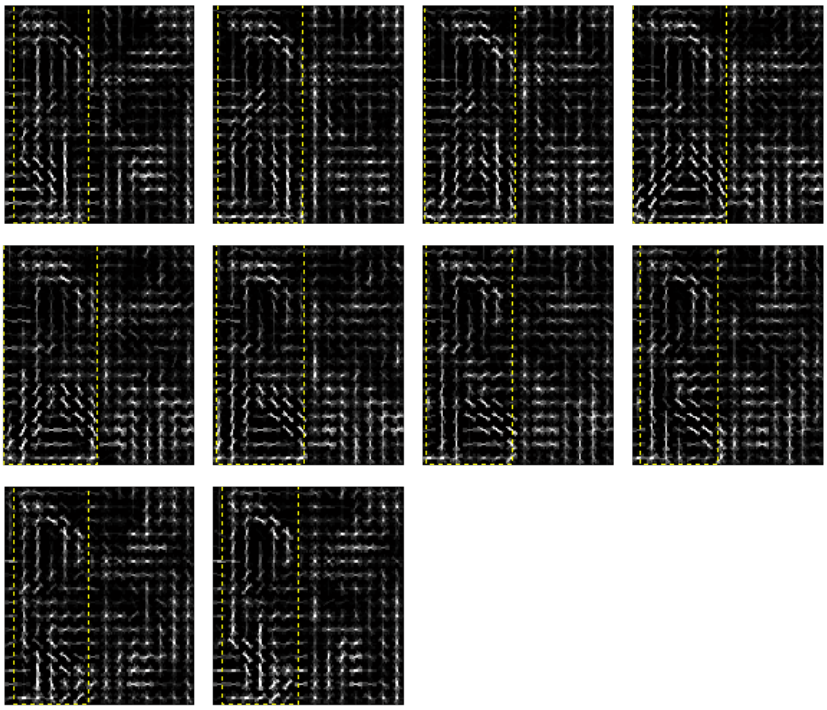


Figure B.3: Visualization of HOG templates, obtained with Fourier training and RR, for the TUD-Campus/TUD-Crossing dataset. Each template represents a distinct pose across a pedestrian's walk cycle. Positive weights are shown on the left, negative weights on the right. The tight-fitting bounding box is also displayed as a yellow line.

BIBLIOGRAPHY

[1] A. Alda. *Things I Overheard While Talking to Myself*. Random House, 2007. (Cited on page ix.)

[2] R. Aldrovandi. *Special Matrices of Mathematical Physics: Stochastic, Circulant, and Bell Matrices*. World Scientific, 2001. (Cited on page 31.)

[3] B. Alexe, V. Petrescu, and V. Ferrari. Exploiting spatial overlap to efficiently compute appearance distances between image windows. In *Advances in Neural Information Processing Systems*, 2011. (Cited on page 6.)

[4] M. Andriluka, S. Roth, and B. Schiele. People-tracking-by-detection and people-detection-by-tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008. (Cited on pages 5, 34, and 93.)

[5] S. Avidan. Support vector tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004. (Cited on page 46.)

[6] B. Babenko, M. Yang, and S. Belongie. Robust object tracking with online multiple instance learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011. (Cited on pages 6, 34, 45, 46, 57, and 58.)

[7] A. Barla, R. Odone, and A. Verr. Histogram intersection kernel for image classification. In *IEEE International Conference on Image Processing*, 2003. (Cited on page 29.)

[8] J. Biemond, R. L. Lagendijk, and R. M. Mersereau. Iterative methods for image deblurring. *Proceedings of the IEEE*, 78(5): 856–883, 1990. (Cited on page 31.)

[9] V. N. Boddeti. *Advances in correlation filters: vector features, structured prediction and shape alignment*. PhD thesis, Carnegie Mellon University, 2012. (Cited on page 47.)

[10] D. S. Bolme, B. A. Draper, and J. R. Beveridge. Average of synthetic exact filters. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009. (Cited on pages 7, 43, 45, and 63.)

[11] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. Visual object tracking using adaptive correlation filters. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2010. (Cited on pages 7, 41, 43, 45, 46, 47, 53, 56, 58, 59, 101, and 105.)

[12] D. S. Bolme, Y. M. Lui, B. A. Draper, and J. R. Beveridge. Simple real-time human detection using a single correlation filter. In *Performance Evaluation of Tracking and Surveillance (PETS-Winter), 2009 Twelfth IEEE International Workshop on*, 2010. (Cited on page 43.)

[13] L. Bourdev, S. Maji, T. Brox, and J. Malik. Detecting people using mutually consistent poselet activations. In *European Conference on Computer Vision*, 2010. (Cited on pages 4, 5, 63, and 73.)

[14] S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004. (Cited on pages 15, 18, 20, 22, 23, and 63.)

[15] B. L. Buzbee, G. H. Golub, and C. W. Nielson. On direct methods for solving poisson's equations. *SIAM Journal on Numerical analysis*, 7(4):627–656, 1970. (Cited on page 31.)

[16] D. Casasent and R. Patnaik. Analysis of kernel distortion-invariant filters. In *Proceedings of SPIE*, volume 6764, page 1, 2007. (Cited on pages 43 and 47.)

[17] R. Caseiro, J. F. Henriques, P. Martins, and J. Batista. A nonparametric riemannian framework on tensor field with application to foreground segmentation. In *IEEE International Conference on Computer Vision*, 2011.

[18] R. Caseiro, J. F. Henriques, P. Martins, and J. Batista. Semi-intrinsic mean shift on riemannian manifolds. In *European Conference on Computer Vision*, 2012.

[19] R. Caseiro, P. Martins, J. F. Henriques, and J. Batista. A nonparametric riemannian framework on tensor field with application to foreground segmentation. *Pattern Recognition*, 45(11), 2012.

[20] R. Caseiro, P. Martins, J. F. Henriques, J. Carreira, and J. Batista. Rolling riemannian manifolds to solve the multi-class classification problem. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[21] R. Caseiro, P. Martins, J. F. Henriques, and J. Batista. Beyond the shortest path: Unsupervised domain adaptation by sampling subspaces along the spline flow. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[22] C.-Y. Chang, A. A. Maciejewski, and V. Balakrishnan. Fast eigenspace decomposition of correlated images. *IEEE Transactions on Image Processing*, 9(11):1937–1949, 2000. (Cited on page 83.)

[23] C.-C. Chang and C.-J. Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011. (Cited on pages 18, 19, 81, and 96.)

[24] O. Chapelle and B. Scholkopf. Incorporating invariances in non-linear support vector machines. In *Advances in Neural Information Processing Systems*, 2002. (Cited on pages 82 and 84.)

[25] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang. Fast neural networks with circulant projections. *arXiv preprint arXiv:1502.03436*, 2015. (Cited on page 99.)

[26] T. S. Cohen and M. Welling. Transformation properties of learned visual representations. In *International Conference on Learning Representations*, 2014. (Cited on page 99.)

[27] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. (Cited on page 17.)

[28] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press, 2000. (Cited on pages 13, 15, 17, 18, and 23.)

[29] J. Daemen and V. Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2002. (Cited on page 31.)

[30] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2005. (Cited on pages 2, 4, 5, 6, 34, 63, 64, 74, 75, 81, and 92.)

[31] C. Davies. Facebook moments app auto-curates event albums. *Slashgear*, June 2015. `http://www.slashgear.com/facebook-moments-app-auto-curates-event-albums-15388473/`. (Cited on page 1.)

[32] P. J. Davis. *Circulant matrices*. American Mathematical Society, 1994. (Cited on pages 31, 33, and 38.)

[33] D. Decoste and B. Schölkopf. Training invariant support vector machines. *Machine learning*, 46(1-3):161–190, 2002. (Cited on pages 2 and 98.)

[34] P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012. (Cited on pages 74 and 75.)

[35] P. Dollár. Piotr's Computer Vision Matlab Toolbox (PMT). `http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html`. (Cited on page 56.)

[36] P. Dollár, R. Appel, S. Belongie, and P. Perona. Fast feature pyramids for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014. (Cited on pages 7 and 56.)

[37] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Unsupervised feature learning by augmenting single images. In *International Conference on Learning Representations*, 2014. (Cited on pages 2 and 81.)

[38] C. Dubout and F. Fleuret. Exact acceleration of linear object detectors. In *European Conference on Computer Vision*, 2012. (Cited on pages 7 and 64.)

[39] C. Dubout and F. Fleuret. Accelerated training of linear object detectors. In *CVPR Workshop on Structured Prediction*, 2013. (Cited on pages 7 and 64.)

[40] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. John Wiley & Sons, 2000. (Cited on pages 13 and 15.)

[41] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The PASCAL visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010. (Cited on page 73.)

[42] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 2008. (Cited on pages 10, 18, 19, 64, 66, 70, and 71.)

[43] L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, 2006. (Cited on page 99.)

[44] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010. (Cited on pages 2, 4, 5, 6, 7, 52, 53, 56, 59, 63, 73, 81, 82, and 92.)

[45] V. Ferrari, L. Fevrier, F. Jurie, and C. Schmid. Groups of adjacent contour segments for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008. (Cited on page 78.)

[46] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997. (Cited on page 15.)

[47] A. Geiger, C. Wojek, and R. Urtasun. Joint 3D estimation of objects and scene layout. In *Advances in Neural Information Processing Systems*, 2011. (Cited on pages 82, 86, and 92.)

[48] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? The KITTI Vision Benchmark Suite. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012. (Cited on pages 93 and 94.)

[49] M. Gharbi, T. Malisiewicz, S. Paris, and F. Durand. A gaussian approximation of feature space for fast image similarity. In *MIT CSAIL Technical Report*, 2012. (Cited on page 64.)

[50] R. Girshick, J. Shotton, P. Kohli, A. Criminisi, and A. Fitzgibbon. Efficient regression of general-activity human poses from depth images. In *IEEE International Conference on Computer Vision*, 2011. (Cited on page 1.)

[51] T. Goldstein and S. Osher. The split Bregman method for L1-regularized problems. *SIAM Journal on Imaging Sciences*, 2(2): 323–343, 2009. (Cited on page 14.)

[52] G. H. Golub and C. F. Van Loan. *Matrix computations*, volume 3. JHU Press, 2012. (Cited on page 67.)

[53] M. Gorman. Google patents new facial recognition technology to let users unlock phones with a wink and a smile.

Engadget, June 2013. http://www.engadget.com/2013/06/07/google-face-unlock-facial-gesture-patent/. (Cited on page 1.)

[54] H. Grabner, C. Leistner, and H. Bischof. Semi-supervised online boosting for robust tracking. In *European Conference on Computer Vision*, 2008. (Cited on page 46.)

[55] K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2005. (Cited on page 4.)

[56] R. M. Gray. *Toeplitz and Circulant Matrices: A Review*. Now Publishers, 2006. (Cited on pages 31, 32, 38, 85, 98, 102, and 113.)

[57] S. Hare, A. Saffari, and P. Torr. Struck: Structured output tracking with kernels. In *IEEE International Conference on Computer Vision*, 2011. (Cited on pages 6, 34, 43, 45, 46, and 58.)

[58] B. Hariharan, J. Malik, and D. Ramanan. Discriminative decorrelation for clustering and classification. In *European Conference on Computer Vision*, 2012. (Cited on page 64.)

[59] H. Harzallah, F. Jurie, and C. Schmid. Combining efficient object localization and image classification. In *IEEE International Conference on Computer Vision*, 2009. (Cited on page 6.)

[60] G. Heitz and D. Koller. Learning spatial context: Using stuff to find things. In *European Conference on Computer Vision*, 2008. (Cited on page 93.)

[61] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *European Conference on Computer Vision*, 2012. (Cited on page 57.)

[62] J. F. Henriques, J. Carreira, R. Caseiro, and J. Batista. Beyond hard negative mining: Efficient detector learning via block-circulant decomposition. In *IEEE International Conference on Computer Vision*, 2013.

[63] J. F. Henriques, P. Martins, R. Caseiro, and J. Batista. Fast training of pose detectors in the fourier domain. In *Advances in Neural Information Processing Systems*, 2014.

[64] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015.

[65] J. Henriques, R. Caseiro, and J. Batista. Globally optimal solution to multi-object tracking with merged measurements. In *IEEE International Conference on Computer Vision*, 2011. (Cited on page 45.)

[66] C.-H. Ho and C.-J. Lin. Large-scale linear support vector regression. *Journal of Machine Learning Research*, 2012. (Cited on page 23.)

[67] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 2012. (Cited on page 22.)

[68] M. Isard and A. Blake. Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998. (Cited on pages 45 and 46.)

[69] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2010. (Cited on page 4.)

[70] K.-H. Jeong, P. P. Pokharel, J.-W. Xu, S. Han, and J. Principe. Kernel based synthetic discriminant function for object recognition. In *ICASSP*, 2006. (Cited on pages 43 and 47.)

[71] J. Johnson. Digital camera face recognition: How it works. *Popular Mechanics*, September 2009. `http://www.popularmechanics.com/technology/gadgets/how-to/a1857/4218937/`. (Cited on page 1.)

[72] N. Jojic, P. Simard, B. J. Frey, and D. Heckerman. Separating appearance from deformation. In *IEEE International Conference on Computer Vision*, 2001. (Cited on page 82.)

[73] Z. Kalal, K. Mikolajczyk, and J. Matas. Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012. (Cited on pages 6, 45, 46, and 58.)

[74] D. Kalman and J. E. White. Polynomial equations and circulant matrices. *American Mathematical Monthly*, pages 821–840, 2001. (Cited on page 31.)

[75] I. Kra and S. R. Simanca. On circulant matrices. *Notices of the American Mathematical Society*, 59(3):368–377, 2012. (Cited on page 31.)

[76] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012. (Cited on pages 2, 15, 34, 64, 81, and 99.)

[77] C. Lampert, M. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008. (Cited on page 6.)

[78] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2006. (Cited on page 4.)

[79] C.-K. Li and W. So. Isometries of Lp-norm. *The American Mathematical Monthly*, 101(5):452, May 1994. (Cited on page 110.)

[80] C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region Newton method for logistic regression. *Journal of Machine Learning Research*, 9:627–650, 2008. (Cited on page 66.)

[81] K. Liu, H. Skibbe, T. Schmidt, T. Blein, K. Palme, T. Brox, and O. Ronneberger. Rotation-invariant HOG descriptors using fourier analysis in polar and spherical coordinates. *International Journal of Computer Vision*, 106(3):342–364, February 2014. (Cited on pages 83 and 93.)

[82] D. G. Lowe. Object recognition from local scale-invariant features. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1999. (Cited on pages 2 and 4.)

[83] R. G. Lyons. *Understanding digital signal processing*. Pearson Education, 2010. (Cited on pages 7, 36, 37, 40, 41, 52, and 101.)

[84] F. J. MacWilliams and N. J. A. Sloane. *The theory of error correcting codes*, volume 16. Elsevier, 1977. (Cited on page 31.)

[85] J. R. Magnus and H. Neudecker. The commutation matrix: some properties and applications. *The Annals of Statistics*, pages 381–394, 1979. (Cited on pages 68, 106, and 108.)

[86] A. Mahalanobis, B. Kumar, and D. Casasent. Minimum average correlation energy filters. *Applied Optics*, 1987. (Cited on pages 7, 41, 43, 47, and 63.)

[87] S. Maji and A. C. Berg. Max-margin additive classifiers for detection. In *IEEE International Conference on Computer Vision*, 2009. (Cited on pages 28 and 29.)

[88] T. Malisiewicz, A. Gupta, and A. A. Efros. Ensemble of Exemplar-SVMs for object detection and beyond. In *IEEE International Conference on Computer Vision*, 2011. (Cited on pages 5, 64, 82, and 92.)

[89] P. Martins, R. Caseiro, J. F. Henriques, and J. Batista. Let the shape speak - face alignment using conjugate priors. In *British Machine Vision Conference*, 2012.

[90] P. Martins, R. Caseiro, J. F. Henriques, and J. Batista. Discriminative bayesian active shape models. In *European Conference on Computer Vision*, 2012.

[91] P. Martins, R. Caseiro, J. F. Henriques, and J. Batista. Likelihood-enhanced bayesian constrained local models. In *IEEE International Conference on Image Processing*, 2014.

[92] P. Martins, J. F. Henriques, R. Caseiro, and J. Batista. Bayesian constrained local models revisited. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015 (to appear).

[93] R. Memisevic and G. E. Hinton. Learning to represent spatial transformations with factored higher-order boltzmann machines. *Neural Computation*, 22(6):1473–1492, 2010. (Cited on page 83.)

[94] X. Miao and R. Rao. Learning the lie groups of visual invariance. *Neural computation*, 19(10):2665–2693, 2007. (Cited on pages 2, 83, and 99.)

[95] T. P. Minka. Old and new matrix algebra useful for statistics. 2000. (Cited on pages 38 and 108.)

[96] T. Moynihan. Google photos is your new essential picture app. *Wired*, June 2015. `http://www.wired.com/2015/05/google-photos-new-essential-picture-app/`. (Cited on page 1.)

[97] J. L. Mundy. Object recognition in the geometric era: A retrospective. *Lecture Notes in Computer Science*, pages 3–28, 2006. (Cited on page 82.)

[98] P. Niyogi, F. Girosi, and T. Poggio. Incorporating prior information in machine learning by creating virtual examples. *Proceedings of the IEEE*, 86(11):2196–2209, 1998. (Cited on pages 2 and 98.)

[99] T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987, 2002. (Cited on page 4.)

[100] J. P. Oliveira, J. M. Bioucas-Dias, and M. A. Figueiredo. Adaptive total variation image deblurring: a majorization–minimization approach. *Signal Processing*, 89(9):1683–1693, 2009. (Cited on pages 14 and 31.)

[101] S. Oron, A. Bar-Hillel, D. Levi, and S. Avidan. Locally orderless tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012. (Cited on page 45.)

[102] M. Osadchy, D. Keren, and B. Fadida-Specktor. Hybrid classifiers for object classification with a rich background. In *European Conference on Computer Vision*, 2012. (Cited on page 64.)

[103] R. Patnaik and D. Casasent. Fast FFT-based distortion-invariant kernel filters for general object recognition. In *Proceedings of SPIE*, volume 7252, 2009. (Cited on pages 43 and 47.)

[104] M. Paulin, J. Revaud, Z. Harchaoui, F. Perronnin, and C. Schmid. Transformation pursuit for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014. (Cited on page 81.)

[105] F. Perronnin, J. Sánchez, and T. Mensink. Improving the Fisher kernel for large-scale image classification. In *Computer Vision–ECCV 2010*, pages 143–156. Springer, 2010. (Cited on pages 4 and 43.)

[106] H. Pirsiavash and D. Ramanan. Steerable part models. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012. (Cited on page 5.)

[107] R. Plomin. *Genetics and experience: The interplay between nature and nurture.* Sage Publications, 1994. (Cited on page 1.)

[108] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2007. (Cited on page 61.)

[109] H. Rauhut. Compressive sensing and structured random matrices. *Theoretical foundations and numerical methods for sparse recovery*, 9:1–92, 2010. (Cited on page 31.)

[110] M. Reisert. *Group integration techniques in pattern analysis: a kernel view*. PhD thesis, University of Freiburg, 2008. (Cited on page 2.)

[111] J. Revaud, M. Douze, S. Cordelia, and H. Jégou. Event retrieval in large video collections with circulant temporal encoding. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013. (Cited on page 99.)

[112] R. Rifkin, G. Yeo, and T. Poggio. Regularized least-squares classification. *Nato Science Series Sub Series III: Computer and Systems Sciences*, 190:131–154, 2003. (Cited on pages 16, 45, and 70.)

[113] E. T. Rolls and G. Deco. *Computational neuroscience of vision*. Oxford University Press, 2002. (Cited on page 1.)

[114] D. A. Ross, J. Lim, R. Lin, and M. Yang. Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 77(1-3):125–141, August 2007. (Cited on page 45.)

[115] D. L. Ruderman and W. Bialek. Statistics of natural images: Scaling in the woods. In *Advances in Neural Information Processing Systems*, 1993. (Cited on page 64.)

[116] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014. `http://arxiv.org/abs/1409.0575`. (Cited on page 4.)

[117] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof. On-line random forests. In *3rd IEEE ICCV Workshop on On-line Computer Vision*, 2009. (Cited on pages 6, 45, and 46.)

[118] T. Santhanam and A. Tekumalla. Quantum mechanics in finite dimensions. *Foundations of Physics*, 6(5):583–587, 1976. (Cited on page 31.)

[119] U. Schmidt and S. Roth. Learning rotation-aware features: From invariant priors to equivariant descriptors. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012. (Cited on pages 83 and 93.)

[120] B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In *Computational learning theory*, pages 416–426. Springer, 2001. (Cited on pages 20, 21, and 22.)

[121] B. Schölkopf and A. Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. The MIT Press, 2002. (Cited on pages 7, 13, 14, 15, 17, 18, 19, 22, 23, 24, 25, 29, 43, 45, 46, and 61.)

[122] L. Sevilla-Lara and E. Learned-Miller. Distribution fields for tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012. (Cited on page 46.)

[123] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated sub-gradient solver for SVM. *Mathematical programming*, 127(1):3–30, 2011. (Cited on page 15.)

[124] P. Simard, Y. LeCun, J. Denker, and B. Victorri. Transformation invariance in pattern recognition – tangent distance and tangent propagation. In *Lecture Notes in Computer Science*. Springer, 1998. (Cited on page 82.)

[125] S. Singh. *Fermat's last theorem*. Fourth Estate, 1997. (Cited on page 31.)

[126] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2003. (Cited on page 4.)

[127] A. Smeulders, D. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah. Visual tracking: an experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013. (Cited on pages 5, 44, and 45.)

[128] I. Steinwart, D. Hush, and C. Scovel. An explicit description of the reproducing kernel hilbert spaces of gaussian RBF kernels. *IEEE Transactions on Information Theory*, 52(10):4635–4643, 2006. (Cited on pages 28 and 29.)

[129] I. Steinwart, D. Hush, and C. Scovel. Training SVMs without offset. *Journal of Machine Learning Research*, 12:141–202, 2011. (Cited on page 23.)

[130] S. M. Stigler. Gauss and the invention of least squares. *The Annals of Statistics*, pages 465–474, 1981. (Cited on page 16.)

[131] B. Tamaki, T.and Yuan, K. Harada, B. Raytchev, and K. Kaneda. Linear discriminative image processing operator analysis. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012. (Cited on pages 2, 83, and 84.)

[132] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B (Methodological)*, pages 267–288, 1996. (Cited on page 15.)

[133] A. N. Tikhonov and V. Y. Arsenin. Solutions of ill-posed problems. 1977. (Cited on page 14.)

[134] M. Uenohara and T. Kanade. Optimal approximation of uniformly rotated images. *IEEE Transactions on Image Processing*, 7 (1):116–119, 1998. (Cited on page 83.)

[135] S. Ullman, M. Vidal-Naquet, and E. Sali. Visual features of intermediate complexity and their use in classification. *Nature neuroscience*, 2002. (Cited on pages 4, 5, 63, and 73.)

[136] J. Valmadre, S. Sridharan, and S. Lucey. Learning detectors quickly with stationary statistics. In *Asian Conference on Computer Vision*. 2014. (Cited on page 98.)

[137] V. Vapnik. *The nature of statistical learning theory*. Springer Science & Business Media, 2000. (Cited on pages 13, 14, and 70.)

[138] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011. (Cited on pages 28, 29, 43, 49, and 61.)

[139] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman. Multiple kernels for object detection. In *IEEE International Conference on Computer Vision*, 2009. (Cited on page 6.)

[140] A. Vedaldi, M. Blaschko, and A. Zisserman. Learning equivariant structured output SVM regressors. 2011. (Cited on page 5.)

[141] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2001. (Cited on page 5.)

[142] M. Welling. Lecture notes on kernel ridge regression. 2013. `http://www.ics.uci.edu/~welling/classnotes/papers_class/Kernel-Ridge.pdf`. (Cited on page 22.)

[143] C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, 2001. (Cited on page 61.)

[144] Y. Wu, B. Shen, and H. Ling. Online robust image alignment via iterative convex optimization. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012. (Cited on pages 46 and 58.)

[145] A. Wyn-Jones. *Circulants*. 2008. (Cited on page 31.)

[146] C. Xie, M. Savvides, and B. Vijaya-Kumar. Kernel correlation filter based redundant class-dependence feature analysis (KCFA) on FRGC2.0 data. In *Analysis and Modelling of Faces and Gestures*. 2005. (Cited on pages 43, 47, and 63.)

[147] Y. Wu, J. Lim, and M. H. Yang. Online object tracking: A benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013. (Cited on pages 46, 55, 56, 57, 58, and 59.)

[148] H. Yang, L. Shao, F. Zheng, L. Wang, and Z. Song. Recent advances and trends in visual tracking: A review. *Neurocomputing*, 74(18):3823–3831, November 2011. (Cited on pages 5, 44, and 45.)

[149] B. Yao, A. Khosla, and L. Fei-Fei. Combining randomization and discrimination for fine-grained image categorization. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2011. (Cited on page 99.)

[150] A. Zamir, A. Dehghan, and M. Shah. GMCP-Tracker: global multi-object tracking using generalized minimum clique graphs. In *European Conference on Computer Vision*, 2012. (Cited on page 45.)

[151] K. Zhang, L. Zhang, and M.-H. Yang. Real-time compressive tracking. In *European Conference on Computer Vision*, 2012. (Cited on pages 6, 45, 46, and 58.)