



UNIVERSITÀ DI PARMA

UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

32 Ciclo

**COMPUTER VISION AND
MACHINE LEARNING
FOR THE
CREATIVE INDUSTRY**

Coordinatore:

Chiar.mo Prof. Mario Locatelli

Tutor:

Chiar.mo Prof. Andrea Prati

Dottorando: *Luca Donati*

Anni 2016/2019

To Moira and Lorenzo

Contents

1	Introduction	1
1.1	Adidas collaborations	2
1.2	Sketch vectorization	4
1.3	Feature extraction	6
2	Related Works	9
2.1	Papers related to sketch vectorization	9
2.2	Papers related to feature extraction	15
3	A complete Hand-drawn Sketch Vectorization Framework	19
3.1	Framework Overview	19
3.1.1	Line extraction	21
3.1.2	Thinning	29
3.1.3	Creating and improving paths	35
3.1.4	Vectorization process	41
3.2	Experiments	43
3.2.1	Line extraction	43
3.2.2	Unbiased thinning	47
3.2.3	Vectorization algorithm	49
4	Fashion Product Classification through Deep Learning and Computer Vision	57
4.1	System Overview	57

4.2	Logo Detection	63
4.2.1	Logo detection through deep learning techniques	64
4.2.2	Logo detection through standard computer vision approaches	66
4.3	Other Features	72
4.3.1	Stripes recognition	72
4.3.2	Colors segmentation	74
4.3.3	Deep learning based feature extraction	75
4.4	Experimental Results	76
4.4.1	DL modules	76
4.4.2	CV modules	80
5	Conclusions	83
5.1	Future works	85
5.1.1	Shoe colorization	85
5.1.2	Attribute modification	86
	Bibliography	91
	Acknowledgments	101

List of Figures

2.1	Some vectorization works, such as [1], limit themselves to process only clean CAD inputs.	11
2.2	Simo-Serra [2], and other neural network approaches, perform very good looking beautifications. Still, their algorithm exhibits poor Recall, e.g. it tends to lose detail (see Figure 2.3).	12
2.3	Detail lost by beautification network [2].	12
2.4	Some papers about vectorization, such as [3], assume real pencil and paper sketches as inputs, but limit their discussion to quite clean ones.	14
3.1	Examples of the kind of sketches we treat (a) (monochrome, lines only), and another type that is not subject of this article (b) (color, large dark blobs).	20
3.2	An overview of the system: vectorization of a portion of shoe sketch.	22
3.3	Perpendicular sections of a pencil line can be well approximated by a bell-like function (e.g. Gaussian or arc of circumference). The trait intensity has a strong correlation with the pencil tip that traced it.	23
3.4	Detecting two parallel lines could be just a matter of stroke hardness and surrounding context.	23

3.5	Two examples of <i>PCC</i> images obtained with different kernels. These pictures show that using a line-shaped kernel (<i>KLine</i>) can be detrimental for retrieval quality: (b), (e); crossing lines are truncated or detected as thinner than they should be. Using <i>KDot</i> can alleviate the problem: (c), (f); this kernel detects more accurately ambiguous junctions.	26
3.6	Part of a shoe sketch and its extracted <i>LinesRegion</i> (after postprocessing).	29
3.7	Example of the biasing effect while thinning a capital letter <i>N</i> (on the left). On the right, the ideal representation of the shape to be obtained.	30
3.8	Applying an equal erosion to all the points of a concave shape implies eroding at a faster speed alongside steep angles. A speed of $s = 1/\sin(\alpha)$ must be applied.	31
3.9	(a) Curvature color map for the contours. Concave angles (navigating the contour clockwise) are presented in green, convex angles in red. Blue points are zero-curvature contours. (b) A portion of the contours curvature map, and a real example of the erosion-thinning process steps (c), (d), (e). The green part of (b) is the neighborhood N_{P_E} that will be eroded at the same time along d_E direction.	33
3.10	Four rotations of these masks of a hit-miss operator (1 = hit, 0 = miss, empty = ignored) are used to detect pixels necessary to preserve path connectivity.	35
3.11	Four rotations of this simple mask of <i>in-place</i> hit-miss morph-operator (1 = hit, 0 = miss, empty = ignored) are used to transform a thinned image to a “strictly 8-connected” one.	36
3.12	Four rotations of these masks of a hit-miss operator (1 = hit, 0 = miss, empty = ignored) are used to detect all the junctions in a “strictly 8-connected skeleton”.	37
3.13	Examples of adjacent junctions (highlighted). Each of these junctions can not be deleted without changing the underlying topology, but can be treated as one.	37

3.14	Four rotations of these masks of a hit-miss operator (1 = hit, 0 = miss, empty = ignored) are used to detect all the endpoints in a “strictly 8-connected skeleton”.	38
3.15	Examples of the four types of paths: $e \leftrightarrow e$ (a), $e \leftrightarrow j$ (b), $j \leftrightarrow j$ (c) and closed (d). Junctions and endpoints are highlighted.	39
3.16	Examples of semantic-aware connectivity (a), where only four paths are detected, opposed to “tic-tac-toe” connectivity (b), where twelve small paths are detected. In this second case, overlapping paths are treated separately as different paths and no semantic knowledge of the context is retained.	39
3.17	An example of <i>pruning</i> applied to an input image (a). Small branches are deleted from the resulting image (b).	40
3.18	An example of <i>merging</i> applied to an input image (a). Parallel paths are combined together in the resulting image (b).	41
3.19	An example of endpoints <i>linking</i> applied to an input image (a). Paths with adjacent endpoints have been connected in the output (b). . . .	42
3.20	Comparisons for the line extraction step: an image from our inverse dataset - first column; a random sketch from the internet, author: Michael Bencik - second column; a real Adidas AG TM hand drawn design - third column. Results are obtained from: a commercial tool in second row, two state of the art algorithms ([2] in third row and [4] in fourth row), and our method in the bottom row.	51
3.21	Examples of the “inverse dataset” sketches created from SHREC13 [5].	52
3.22	Examples of thinning results. (a) The input images. (b) Thinning results with the “standard” Zhang-Suen algorithm’. (c) Results for K3M method (state of the art). (d) Our algorithm results. Fonts used (from the top): 200pts Arial; 180pts Dijkstra; 150pts Times New Roman (30 degrees rotated), 120pts TwCen.	53

3.23	Four example portions of vectorization. (a) and (c) use Schneider's stock algorithm, (b) and (d) use our improved version. Our algorithm considerably reduces the number of control points for both cases ($\text{err} = 3$ and $\text{err} = 6$), without decreasing result quality. For all the examples the maximum number of iterations has been dynamically set for each path to its own length in pixels (1 iteration per pixel).	54
3.24	Examples of end-to-end vectorizations performed by our system. An Adidas AG TM shoe sketch (a), (b).	55
3.25	Other examples of end-to-end vectorizations performed by our system. Another difficult Adidas AG TM shoe sketch (a), (b), and a dirty, low resolution, preparatory fashion sketch (c), (d).	56
4.1	Types of Adidas logos.	58
4.2	Prints and clothing patterns examples.	59
4.3	Types of neck shapes.	59
4.4	In "raglan" shirts, the junction between the body and the sleeves starts from the armpits and goes to the neck; the term "set-in", instead, refers to those shirts that have the sleeves sewn with the body of the garment with a vertical closure.	60
4.5	Detailed organization of system modules. Uppercase statements highlight inputs and outputs, while boxed ones emphasize the actual modules of the system. Solid arrows denote conceptual connections among modules, while dashed arrows denote inputs or outputs.	61
4.6	Heatmap of a nearly invisible medium logo.	66
4.7	Examples of occlusion (a), rotation (b), deformation (c), three typical issues that have to be taken into account for correct logo detection. The ground truth dataset comprises only some examples of these difficult cases. Still, more examples can be generated via data augmentation.	67
4.8	An example of logo detection using template matching.	71

5.1	A shoe from the Adidas AG TM high-resolution shoe dataset (1200x600 pixels, 20.000 images).	86
5.2	A shoe from the Adidas AG TM dataset and its correspondent sketch, generated with edge-detection. Also note the color hints in the right, bottom parts of the sketch. Those will guide the network in reconstructing the color of each part.	87
5.3	Some input output examples of transformations performed by our U-net. The input has not been previously seen by the network (e.g. it is a validation sample).	88
5.4	Facial attributes generation by our proposed few shot image to image translation method.	90

List of Tables

3.1	Accuracy of the three implementations over the <i>inverse dataset</i> generated from SHREC13 [5] (2700 images).	47
3.2	Running time and number of control points generated by the two versions of the vectorization algorithm. Desired error $err = 6$. Less is better.	49
3.3	Running time and number of control points generated by the two versions of the vectorization algorithm. Desired error $err = 3$. Less is better.	49
4.1	Training of a simple CNN from scratch.	76
4.2	Some tests on fine-tuning with VGG16 topology and relative accuracy values.	77
4.3	Some tests on fine-tuning with VGG19 topology and relative accuracy values.	78
4.4	Learning rate parameter variations.	78
4.5	Results on large logo detection on an unseen validation dataset, performed by Adidas team.	79
4.6	Results on prints classification on an unseen validation dataset, performed by Adidas team.	79
4.7	Results on medium logo detection, using different architectures. . .	80
4.8	Results on logo size detection on an unseen validation dataset, performed by Adidas team. Overall accuracy is 96.6% on 290 images. .	80

4.9	Results on small logo detection on an unseen validation dataset, performed by Adidas team.	81
4.10	Results on not-printed color combination on an unseen validation dataset, performed by Adidas team.	81

Chapter 1

Introduction

*In the beginning the Universe was created.
This has made a lot of people very angry
and been widely regarded as a bad move.*

– Douglas Adams

Computer Vision, a staple technique for the automation industry, is becoming a prominent tool also in the creative industry. Fashion is less subject than automation to the strict timings and accuracies required by a production line, for which computer vision has proven to be such a formidable tool; however, many areas of fashion business can clearly benefit from automation techniques and the processing power of modern processors.

For long, computer graphics have been the major IT field of interest for the apparel business. The need of creating 2D and 3D renderings of clothes, furniture, shoes has been expressed and satisfied by several tools. Advertising is also a major composing part of fashion, where computer graphics is a fundamental tool. Nowadays, tasks such as picture processing, 3D modeling, photoediting are ubiquitous and relative tools are used by the large majority of fashion designers.

Somewhat tangentially, computer vision started to gain traction in this crowded field. At first, as an aid for computer graphics. Many of the common filters and algo-

rithms found in rendering and graphics software were born and developed as Computer Vision techniques. Modern renderings, filters, stunning editings rely on computer vision as a powerful backbone to operate.

In a second wave, computer vision arose as a tool for measurement and analysis for fashion. The world vehiculates products and trends mainly with images (or sequences of them). Analyzing those images is often mandatory for understanding what is being produced and advertised by both a company itself and its main competitors.

Finally, Machine Learning, Convolutional Neural Networks and Generative Networks provided the last push for the definitive explosion of Computer Vision for fashion. Neural networks (and Machine learning in general) can process and cluster massive amounts of data, providing directions to the development teams; such directions range from which colors are better for a new season of apparel, to what kind of garments sell most in a specific region, or which direction is a competitor heading to. Generative Networks have started providing the last bit of utility that was missing from machines: proposing new products ideas, suggesting colors and shapes, providing creativity and synthesizing new textures and material patterns.

As a proof of this, from as soon as 2017, the major Computer Vision conferences and journals started hosting workshops and special issues dedicated solely to this topic.

1.1 Adidas collaborations

In this work of thesis we will explore three years of collaboration of the author with Adidas AGTM, one of the leader companies in the sport fashion market, and the many systems and ideas developed as consequence. These systems are very related to real, specific needs of a creative business company as big as Adidas, which have partnerships with many relevant actors in the Computer Vision field, such as Adobe. Nevertheless they relate to the aforementioned hot topics in Computer Vision for Fashion, and provided valuable research material.

We will now briefly explain three of the pathways of development undertook

by the author, which touch all the aforementioned drives in the computer vision for fashion community.

- Computer Vision as a Graphics tool. Every creative company has designers, each one uses its own tools, may they be manual, like pencil and paper, or technological, like drawing tablets. Both of these working styles usually produce proof of concepts and designs that need to be converted in final products and renderings. This process involves what is called *sketch vectorization*, i.e. transforming a raw paper sketch in a full vectorized collection of B-splines. Computer vision can perform this task automatically, saving much of designers' time. Part of this Introduction and the whole Chapter 3 will discuss this topic.
- Computer Vision and Machine Learning as tools for analysis. Medium to Large business companies will sooner or later find the need of analyzing their production during time and look at competitors. Adidas is no exception, since it has development teams across the world and a huge amount of new products per year. Moreover, fashion has seasons and trends that rapidly change, and each business company has to monitor closely what is "cool" and emerges as a tendency, and what is "old", and needs to be ceased. One of the main ways of doing this is analyzing thousands of products pictures and relating them to the sales; another is looking at social media, and finding what is a hot trend that bounces around the web and what not. To do this, we need flexible tools of image analysis, in the form of Computer Vision (color palette estimation, template matching for detection), and Machine Learning (classification, clustering). This Introduction and Chapter 4 will highlight and find answers for this topic.
- Machine Learning as a tool for creation, creativity and suggestions. Every designer is in need of ideas. Machines have not been known in the past for creativity, but this is changing with the advent of the Deep Learning paradigm. Modern networks have been able to suggest new colors for upcoming seasons, color black and white photographs, transform an image into another due to some constraints, and even produce new image of objects from scratch. Fash-

ion can greatly benefit from all of these possibilities. Suggesting colors to both designers and users is paramount in any business that has seasonal colorways, of for those cases in which products differ from each other only by color (e.g. sport shoes). Coloring a simple grayscale sketch is a great source of inspiration for both a designer perspective and a website in the manner of “Design Your Shoe”. Image to image transform allows for example to apply a new style to an existing garment. Neural Networks Image Generation allows to create a new product from scratch (only with some source examples), the ultimate source of inspiration for creatives. In Conclusions and Future Works chapter we will briefly propose and discuss some applications on the topic.

1.2 Sketch vectorization

Raw paper sketches are usually the starting point of many creative and fashion workflows. For many artists the choice of drawing with pencil and paper (or pen) grants them the most expressiveness and creative freedom possible. By using these simple tools they can convey ideas in a very fast and natural way. That allows them to propose powerful and innovative designs. Later, the prototypal idea from the hand-drawn sketch must be converted to a real world product.

The *de-facto* standard for distributing fashion and mechanical designs is the vectorized set of lines composing the raw sketches: formats like SVG, CAD, Adobe Illustrator files are manually created by the designers, and delivered and used by a plethora of departments for many applications, e.g. marketing, production line, end-of-season analyses, etc..

Unfortunately, the vectorization process is still a manual task, which is both tedious and time-consuming. Designers have to click over each line point by point and gain a certain degree of experience with the used tools to create a good representation of the original sketch model.

Therefore, the need for an automated tool arises. A great amount of designers’ time can be saved and re-routed in the creative part of their job. Some tools with

this specific purpose are commercially available, such as Adobe Illustrator^{TM1} Live Trace, Wintopo² and Potrace³. Anyway, to our knowledge and experience with Adidas designers, none of these tools does this job in a proper or satisfying way.

Vectorization of hand-drawn sketches is a well-researched area, with robust algorithms, such as SPV [6] and OOPSV [7]. However, these methods, as well as others in the literature, fail to work with real scribbles composed of multiple strokes, since they tend to vectorize each single line, while not getting the right semantics of the drawing [8]. Problems to be faced in real cases are mainly the followings: bad/imprecise line position extractions; lines merged together when they should not or split when they were a single one in the sketch; lines extracted as “large” blobs (shapes with their relative widths instead of zero-width line segments); unreliability of detection with varying stroke hardness (dark lines are overly detected, faint lines are not detected at all); resulting “heavy” b-splines (the vectorized shapes are composed of too many control points, making subsequent handling hard).

Some works for a complete vectorization workflow are also present in the literature, even if mainly addressing vectorization of really clean sketches, or obtaining decent artistic results from highly noisy and “sloppy” paintings. None of them is designed to work with “hard” real data trying to retrieve the most precise information of exact lines and produce high quality results.

This work addresses exactly this issue and provides a complete workflow for automated vectorization of rough and noisy hand-drawn sketches. Two new methods (contributions) are presented:

- A reliable line extraction algorithm. Being based on Pearson’s Correlation Coefficient, it is able to work under severe variations and inconsistencies of the input sketch.
- A fast unbiased thinning. The algorithm solves a well-known biasing problem that afflicts most thinning algorithms (where steep angles are not well preserved).

¹<https://www.adobe.com/products/illustrator.html>

²<http://wintopo.com/>

³<http://potrace.sourceforge.net/>

Moreover, many different existing techniques are discussed, improved and evaluated: paths extraction, pruning, edge linking, and Bézier curve approximation.

All of these techniques are used together in a cohesive system and presented as a modular framework that produces accurate vectorizations of difficult, noisy sketches. Valuable properties of the system are: the high quality of its output, in both representation error and simplification strength; the low number of needed parameters; the fast real-time performance; the complete modularity of the system; the full control and maintainability of a classic Computer Vision system (as opposed as a Deep Learning system).

The efficacy of the proposal has been demonstrated on both hand-drawn sketches and images with added artificial noise, showing in both cases excellent performance w.r.t. the state of the art. Numerical benchmarks such as Precision, Recall and 'mean Centerline Distance' are evaluated. Furthermore, a full-fledged Adobe Illustrator plugin has been developed, and feedback from Adidas designers testify the quality of the system and prove that it greatly outperforms existing solutions, in terms of both efficiency and accuracy.

1.3 Feature extraction

In fashion industries, obtaining a visual analysis of the overall production is a key aspect, both in developing marketing strategies and for helping fashion designers in the creative workflow of new products. As a first important step in order to proceed with a visual analysis, the various outcomes of the designers' work must be collected and categorized. This applies especially if there are many different designer teams which are employed with outsourcing contracts, and located all around the world, as it usually happens with large companies. In particular, in Adidas AGTM, the designers' production comprises a consistent number of images which represent a very large variety of products, from clothing to footwear. Moreover, these works are usually independent from each other. Then, articles are of different types and come from different sources, but their characteristics must be analyzed and classified as a whole by data experts and analysts. Hence, a significant step in visual analysis is recognizing,

classifying, and extracting features directly from final images or 3D-renderings of products, collected before the actual fabrication of the clothing. Categorizing apparel products is also useful for e-commerce purposes, as well as for avoiding duplicates, ranking product types, or doing statistical analysis [9].

The categorization of apparel products was, until now, manually performed by Adidas teams, since it requires both domain expertise and a comprehensive knowledge over the range of products. Such a manual classification is an error-prone task that may cause incorrect results, misleading the subsequent visual analysis, and which also requires too much time. As a matter of fact, over the years, the number of produced articles has grown significantly, and, currently, Adidas teams invent $\sim 20k$ different articles per season (twice a year). For all these products, about two dozens of main attributes (or categories) were identified by Adidas data experts, such as the presence and position of logos or the three-stripes, which is the primary color, the presence of prints or clothing patterns, and so on. Each of these categories takes values in a different domain, and ranges across a set of possible classes. As expected, manually performing the classification of attributes for each article has become an unfeasible task. Due to the large amount of data, and the diversity of the source of images, the task of classifying and recognizing features of clothing images is not only hard to perform manually, but at the same time it is also difficult to automate properly. A subset of seven key attributes were chosen, which we will call *features* in the remainder of this thesis, namely, (i) logo type and size, (ii) three-stripes presence and colors, (iii) three main colors palette, (iv) prints and patterns, (v) neck shape, (vi) sleeves shape, and (vii) material of the clothing. Those features are the most important both for business reasons (e.g., detecting logos in clothing is fundamental information for a brand), and for their significance in identifying a particular garment, so as to avoid or substantially reduce duplicates or similar products. The characteristics of each feature and the reasons for their importance are detailed later in the thesis.

The core idea is to automate the classification process of these seven features by using machine learning techniques and computer vision, but many aspects in such a task raise concrete research problems. In fact, each clothing feature could eventually require a different method of classification, spacing from segmentation [10, 11], im-

age retrieval [12], to machine learning techniques, such as deep learning [13, 14, 15, 16]. Those algorithms must be applied to the under-investigated area of fashion, and must be refined for the specific domain. In addition, the automatization must have an accuracy comparable to manual methods in order to be useful for business purposes. Thus, the problem of automatic classification of apparel features turned out to be a challenging application of computer vision and machine learning techniques. The purpose of this thesis is to describe a work that was developed by the author to offer an effective solution to the aforementioned problem, by means of a computer vision software system, and to discuss challenges that arise in the development of that system, feature by feature.

The remainder of this thesis is structured as follows. The next Chapter reports the related works in both Sketch Vectorization and Feature Extraction. Chapter 3 describes the different steps of the vectorization framework (namely, line extraction, thinning, path creation and vectorization). Experimental results on these steps are reported in Section 3.2. Chapter 4 shows an overview of the feature extraction system developed by the author. In sections 4.2 and 4.3, feature extraction is described in detail, with an emphasis on logo detection. Experimental results are provided in Section 4.4. Finally, Chapter 5 presents a discussion of main achievements and future works.

Chapter 2

Related Works

*To go wrong in one's own way is better
than to go right in someone else's.*

– Fyodor Dostoevsky

2.1 Papers related to sketch vectorization

This section reports the most relevant previous works on sketch vectorization. The work in [8] proposed a line enhancement method, based on Gabor and Kalman filters. It can be used to enhance lines for subsequent vectorization. However, this approach fails to correctly extract all the drawing components when the image is noisy or presents parallel strokes, resulting, for instance, in gaps in the final vectorized result or strokes incorrectly merged. Moreover, experiments are conducted with quite simple images.

The paper in [17] provided the first treatment of ridge extraction at multiple resolutions. Their approach works by applying derivatives of Gaussians to an input image and looking for a very specific response configuration to that derivatives. In particular, they look for what the derivative of an ideal ridge should look like. Their experiments, while being good enough, are limited and obtained using only low resolution images.

See the related work in [18] for more details and examples. While this is an interesting approach, we claim that derivatives are better suited for edge detection than line detection. Just looking for derivatives when modeling a line essentially implies that:

- line surface is uniform (or symmetric along middle bisector);
- left and right line limits have the same slope to white background.

Moreover, we claim that looking at the edges of a line to locate it, is an unnatural approach, and does not take into account of the information spread over the whole line surface. Instead, this thesis proposes to match the image with an opportune kernel, taking into account all the information residing over the whole line area. Also, while they address the scale variation problem, they do not address (or just partially) the contrast variation problem, and the line width problem. Their work also needs more parameters to be used than ours.

The work in [1] reported a first proposal of a framework transforming raw images to full vectorized representations. However, the “binarization” step is not considered at all, by presenting directly the skeleton processing and vectorization steps. In addition to this limitation, this paper also bases the vectorization to the simple fitting of straight lines and circular arcs (instead of using Bézier interpolation), which represents a too simplified and limited representation of the resulting path. Figure 2.1 shows the simple inputs their proposal is based on.

The paper in [19] provided a more complete study of the whole vectorization process. This is another proposal that works with derivatives. In fact, they provide a neat derivation-based algorithm to estimate accurate centerlines for sketches. They also provide a good insight of the problem of correct junction selection. Unfortunately, they work under the assumption of somewhat “clean” lines, that does not hold in many real case scenarios, such as those we are aiming at.

Simo-Serra *et al.* [2] trained a Convolutional Neural Network to automatically learn how to simplify a raw sketch. No preprocessing or postprocessing is necessary and the network does the whole process of conversion from the original sketch image to a highly-simplified version. Figure 2.2 shows an example beautification performed

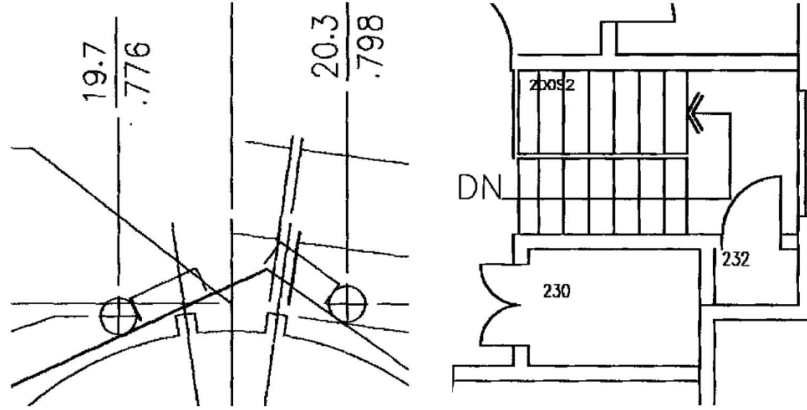


Figure 2.1: Some vectorization works, such as [1], limit themselves to process only clean CAD inputs.

by the network. This task is related to our needs, since it can be viewed as a full preliminary step, that just needs the vectorization step to provide the final output form. In Section 3.2.1 we will compare with this work and show that our proposal achieves better results for retrieval (in particular in terms of recall) and is more reliable when working with different datasets and input types. The same author still improved their results in a recent work [4], with which we will compare too.

The paper in [3] provided an overview of the whole process. Anyway, they gave just brief notions of line extraction and thinning, while they concentrated more on the final vectorization part, in which they proposed an interesting global fitting algorithm. Indeed, this paper is the only paper providing guidelines to obtain a full Bézier-curves representation as final output. Representing images with Bézier curves is of paramount importance in our application domain (fashion design), and is moreover important to obtain “lightweight” vectorial representations of the underlying shapes (composed of as few control points as possible). The vectorization task is casted as a graph optimization problem. However, they treated just partially the noisy image problem, focusing on working with somehow clean paintings. Figure 2.4 shows an

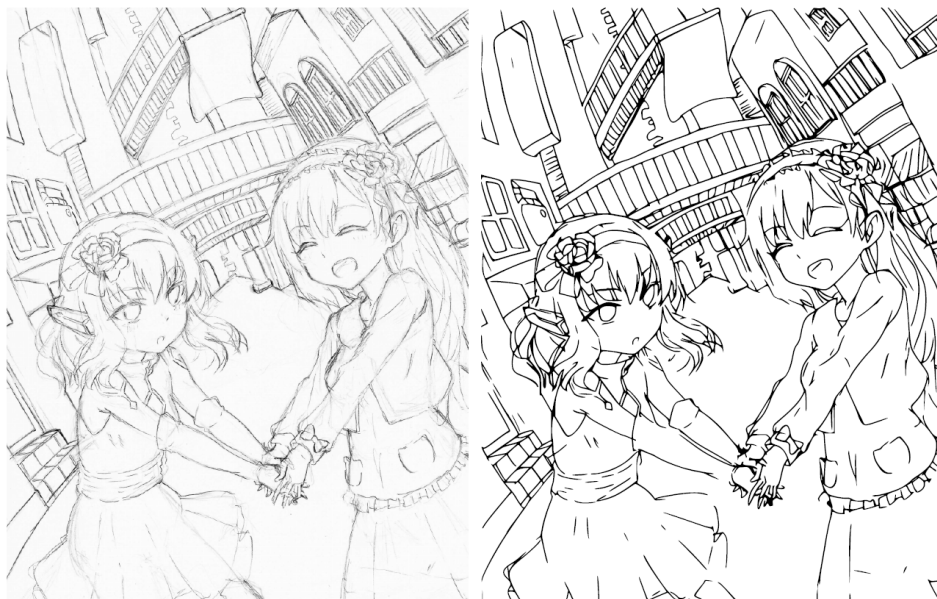


Figure 2.2: Simo-Serra [2], and other neural network approaches, perform very good looking beautifications. Still, their algorithm exhibits poor Recall, e.g. it tends to lose detail (see Figure 2.3).

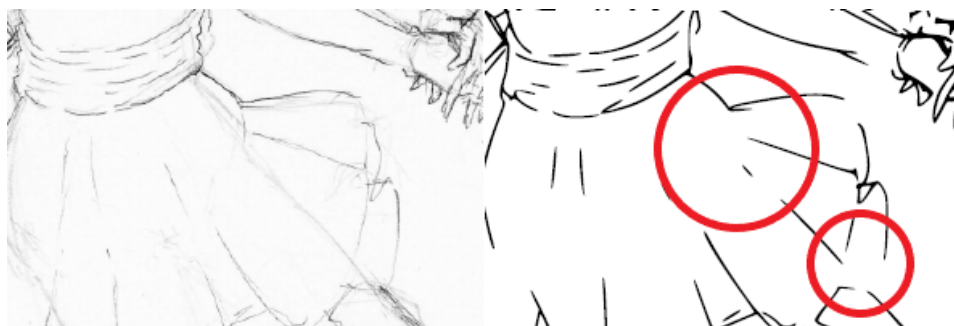


Figure 2.3: Detail lost by beautification network [2].

example.

The work in [20] proposes interesting ways for simplifying sketches, giving much focus on multiple strokes merging and paths linking. It also works in detecting and taking into account different, adjacent regions. Still, to our purpose, their solution only works on very clean images, where the main difficulty lies the presence of multiple strokes. It does not take into account varying stroke pressure and width, nor it accounts pencil roughness and paper porosity or non-uniform background.

Another recent work [21] provided a good proposal for a vectorization system. Regarding line extraction, they rely on Vector Fields, which give high quality results with clean images, but fail in presence of noise and fuzzy lines. Still, they dedicated a lot of attention to correctly disambiguate junctions and parallel strokes.

An interesting work regarding vectorization is reported in [22]. Great care is taken into producing faithful and “beautiful” vectorial representations of drawings. In this case, the input is directly a shape produced by a mouse or a drawing tablet, still their process is relevant to obtain good artistic results from inexperienced users inputs.

When working with colored sketches (or cartoons), the paper in [23] proposed a robust method for vectorization. This paper also introduces an algorithm called Trapped Ball Segmentation, that can take into account additional details about the context surrounding each line, and could be very beneficial in applications that expect colored input sketches.

Another interesting variation of the vectorization problem consists in taking advantage of on-line information about the sketch, while it is drawn. This is typically done using a PC and a drawing tablet. For example, the PC can measure the order in which the traits are drawn, their direction, the speed and the pressure of the pen, etc.. All of these information greatly help the vectorization process: e.g. crossing lines are automatically disambiguated, multiple strokes are easily merged. Examples of works in this specific area are: [24] and [25].

The sketch vectorization field also partially overlaps with the so-called “Coherence Enhancing” field. For instance, the work in [26] estimated Tangent Vector Fields from images, and used them in order to clean or simplify the input. They do that by averaging a pixel value with its corresponding neighbors along the Vector Fields. This

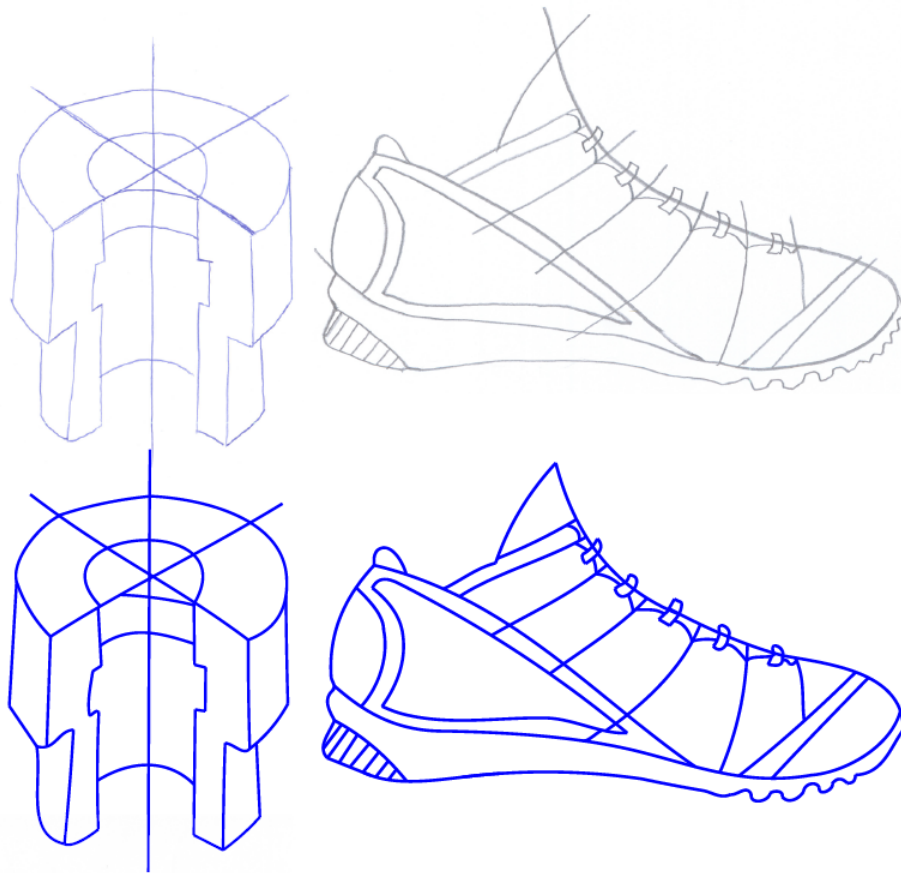


Figure 2.4: Some papers about vectorization, such as [3], assume real pencil and paper sketches as inputs, but limit their discussion to quite clean ones.

could be integrated as a useful preprocessing step in our system, or could be used as a standalone tool if the objective is just to obtain a simplified representation of the input image.

The same research topic has been explored in the work from [27], that showed remarkable results in the task of disambiguating parallel, almost touching, strokes (a very useful property in the sketch analysis domain). Again this could be used as a preprocessing step of our system o to perform picture beautification.

2.2 Papers related to feature extraction

As briefly explained in the introduction, feature extraction from final images of products requires different methods and techniques, depending on which feature to extract. Two main fields are usually investigated, namely, the image classification (or recognition), and the detection of objects in the image. In the following, a brief outline of the major contributions in image classification, objects localization and detection is provided. An outline of this related works section could be found in author's previous paper [28], which presents also a preliminary discussion on this project. However, the study of the state-of-the-art, as well as the development of the described system and the experimental results are substantially enhanced and expanded in this thesis.

Many features, such as recognizing logos and locating them, or detecting prints and clothing patterns, are sub-problems of the object detection task. Object detection is a well-known computer vision task [29] which is employed in many application contexts, such as face detection [30], visual search engines [9], image analysis [31], self-driving cars [32] and so on. A very popular competition which poses the object detection as a central challenge is the ImageNet¹ competition [33]. A large dataset is provided, and the purpose of the competition is to find multiple objects for each image and classify them properly. In particular, the candidate system must retrieve the five best-matching categories of objects in the input image, putting bounding boxes around recognized objects.

Early object detection algorithms were based on integral images [30, 34], and

¹<http://www.image-net.org/>

feature descriptors [35, 36]. In detail, the work of Viola and Jones [30] is a fast face detection algorithm based on integral images and that uses a learning algorithm. Such an approach proved successful, and showed the potentialities of computer vision in real world applications. Nevertheless, face detection is only a narrowed type of object detection, and an improvement is necessary in order to apply it in a more generic context. Then, Dalal and Triggs [35] enhanced previous methods by proposing feature descriptors called Histograms of Oriented Gradients (HOG) for pedestrian detection. The problem with such a method is the same: generalization to other domains, such as fashion, is hardly obtainable. Other descriptors are shown in the work of Lowe [36], that presents a method for image feature generation called Scale Invariant Feature Transform (SIFT), and Bay et al. [37], that propose SURF (Speeded-Up Robust Features), a detector and descriptor which is both scale and rotation invariant, and that speeds up detection using integral images. Nowadays, such approaches are useful in very specialized contexts, and in general—in the task of object detection—they are outperformed by more accurate learning algorithms.

As a matter of fact, most successful approaches are based on deep learning techniques, such as Convolutional Neural Networks (CNNs or ConvNets) [14, 38, 39, 13, 40, 41, 42, 43, 44], that achieve good results in detecting multiple objects, even of various classes, in a given image. Overfeat [43] is an early work that uses multi-scale sliding windows and deep learning (CNNs) for object detection. An enhancement of the CNN approach is a method developed by Girshick et al. [38], called Regions with CNN features (R-CNN), which uses a three-stage algorithm that is not entirely based on deep learning. As a matter of fact, the first step is a region proposal, while the second uses a CNN for feature extraction. For the final classification, a SVM classifier is used. R-CNN significantly enhanced object detection w.r.t. training a CNN approach from scratch, but it had still many problems during the training phase. Girshick [39] then additionally proposed an improvement of R-CNN that deals with the speed and training problems of the previous model. Such an improvement was called Fast R-CNN, and conversely to R-CNN, it uses a pure deep learning paradigm: it employs CNNs also for classification, and uses the Region of Interest (RoI) Pooling. As a further enhancement of the R-CNN approach, a Region Proposal Network (RPN) was

added to the Fast R-CNN method. This development is called Faster R-CNN [41]. The Region-based Fully Convolutional Networks (R-FCN) [14] is somehow similar to Faster R-CNN, since it uses the same architecture, but relying also on CNNs. A work that substantially helps at developing our feature extractor system is the VGG19 network [16], the 2014 winner of the ImageNet challenge in the localization task. VGG19 has a simple architecture which consists of a plain chain of layers, and it owes its good performances—comparable to other more complicated networks—to the use of much more memory and to a slower evaluation time. Therefore, its simple structure makes it valuable for our work. Recently, a novel and deep learning based approach was proposed by Redmon et al. [13]. The project is called YOLO (You Only Look Once), and it is under active development². Another recent approach, named SSD (Single Shot Multibox Detector) [44], is based on a single deep neural network, as for YOLO, and it achieves better results and speed than it. The second enhancement of YOLO is YOLO9000 [40], and a new version was also released in 2018 [42].

As an important drawback, deep learning requires huge amount of data to train a neural network with many layers and to prevent the overfitting of such a network. Moreover, using a supervised approach also requires data to be labeled. Unfortunately, there is a scarcity of labeled dataset from apparel industries, and all the previously detailed deep learning techniques suffer from this disadvantages [45]. Public datasets of colored images at an acceptable resolution for deep learning, such as the ImageNet one, have a high number of classes which have nothing to do with the fashion industry, and the Fashion-MNIST³ dataset, which could appear more appropriate for the application domain, consists of very small grayscale images (28×28) divided in 10 classes, which are not sufficient for our purposes.

In Chapter 4, we approach the problem of lack of data with two different methods. One is the use of Pearson Correlation Coefficient (PCC) [46] to perform a masked template matching. PCC is by definition color invariant, automatically granting generalization capabilities in this sense. This method shows, in specific cases (e.g., recog-

²<https://pjreddie.com/darknet/yolo/>

³<https://github.com/zalandoresearch/fashion-mnist>

dition of small logos), more accurate results than a CNN approach. The other one is the fine-tuning of a pre-trained CNN, i.e. a method to specialize a neural model on personalized data by training only the last layers of the network. In this last case, a pre-processing step of data-augmentation [45] is also needed.

Other features require a more classic computer vision approach, since they are either based on the geometric properties of the garment contours (e.g., stripes recognition), or they involve the calculation of distances in color spaces. First, a border-following algorithm to retrieve the contour of the shape of the garment is needed. A great method for such a task is described in [47]. Then, the natural choice for reducing line shapes to a compact form is to apply a thinning algorithm [48]. Thinning variants are well described in the review [49]. Nevertheless, thinning algorithms suffer from a problem called skeletonization bias [49]. The unwanted bias effect arises when thinning is applied to strongly-concave angles. As described by Chen [50], the bias effect appears when the shape to be thinned has a contour angle steeper than 90 degrees. The original proposal in [50] is based, first, on the detection of steep angles and, then, on the application of a custom local erosion specifically designed to eliminate the so-called “hidden deletable pixels”. In the work [51] an overview of line extraction and vectorization of sketches is shown, and in particular a solution to the skeletonization bias problem is proposed. The same research topic has been explored in the work by Chen et al. [52], that showed remarkable results in the task of disambiguating parallel, almost touching, strokes, a useful property in this domain. A great line extraction and thinning is of fundamental importance for reasoning on geometric characteristics of clothing, in terms of parallel or perpendicular strokes. A line extraction and thinning approach was used in this work for the recognition of the Adidas three-stripes.

Chapter 3

A complete Hand-drawn Sketch Vectorization Framework

*If you think it's simple,
then you have misunderstood the problem*

– Bjarne Stroustrup

3.1 Framework Overview

We propose a modular workflow for sketch vectorization, differently from some papers (e.g., [2]) that are based on monolithic approaches. Following a monolithic approach usually gives very good results for the specific-task dataset, but grants far less flexibility and adaptability, failing to generalize for new datasets/scenarios. A modular workflow is better from many perspectives: it is easy to add parts to the system, change them or adapt them to new techniques and it is much easier to maintain their implementations, and to parametrize or add options to them (expected error, simplification strength, etc.).

The approach proposed in this thesis starts from the assumption that the sketch is



Figure 3.1: Examples of the kind of sketches we treat (a) (monochrome, lines only), and another type that is not subject of this article (b) (color, large dark blobs).

a monochromatic, lines-only image. That is, we assume that no “dark” large blob is present in the area, just traits and lines to be vectorized. Fig. 3.1a shows an example of the sketches we aim to vectorize, while Fig. 3.1b reports another exemplar sketch that, given the previous assumption, will not be considered in this work. Moreover, we will also assume a maximum allowed line width to be present in the input image. These assumptions are valid in the vast majority of sketches that would be useful to vectorize: fashion designs, mechanical parts designs, cartoon pencil sketches and more.

Our final objective is to obtain a vectorized representation composed by zero-width curvilinear segments (Bézier splines or b-splines), with as few control points as possible.

This is an overview of the modules composing the workflow:

- First, line presence and locations from noisy data, such as sketch paintings, need to be extracted. Each pixel of the input image will be labeled as either part of a line or background.
- Second, these line shapes are transformed into 2d paths (each path being an array of points). This can be done via a thinning and some subsequent post-processing steps.
- Third, these paths are used as input data to obtain the final vectorized b-splines.

Each of these modules will be extensively described in the next subsections. A visual overview of the system can be seen in Fig. 3.2

3.1.1 Line extraction

Extracting precise line locations is the mandatory starting point for the whole vectorization process. When working with hand-drawn sketches, we usually deal with pencil lines traced over rough paper. Other options are pens, marker pens, ink or PC drawing tables.

The most difficult of these tool traits to be robustly recognized is, by far, pencil. Unlike ink, pens and PC drawing, it presents a great “hardness” (color) variability. Also, it is noisy and not constant along its perpendicular direction (bell shaped). Fig. 3.3 shows a simple demonstration of the reason of this. Moreover, artists may intentionally change the pressure while drawing to express artistic intentions.

In addition, it is common for a “wide” line to be composed by multiple superimposed thinner traits. At the same time, parallel lines that should be kept separated may converge and almost touch in a given portion of the paint, having as the sole delimiter the brightness of the trait (Fig. 3.4).

With these premises, precise line extraction in this situation represents a great challenge. Our proposed approach is a custom line extraction mechanism that tries to be invariant to the majority of the foretold caveats, and aims to: be invariant to stroke hardness and stroke width; detect bell “shaped” lines, transforming them into classical “uniform” lines; merge multiple superimposed lines, while keeping parallel neighbor lines separated.

Cross Correlation as a similarity measure

The proposed line extraction algorithm is based on Pearson’s Correlation Coefficient (PCC, hereinafter). This correlation coefficient exhibits the properties to identify the parts of the image which resemble a “line”, no matter the line width or strength. This section will briefly introduce the background about PCC.

The “standard” Cross Correlation (or Convolution) operator is known for ex-

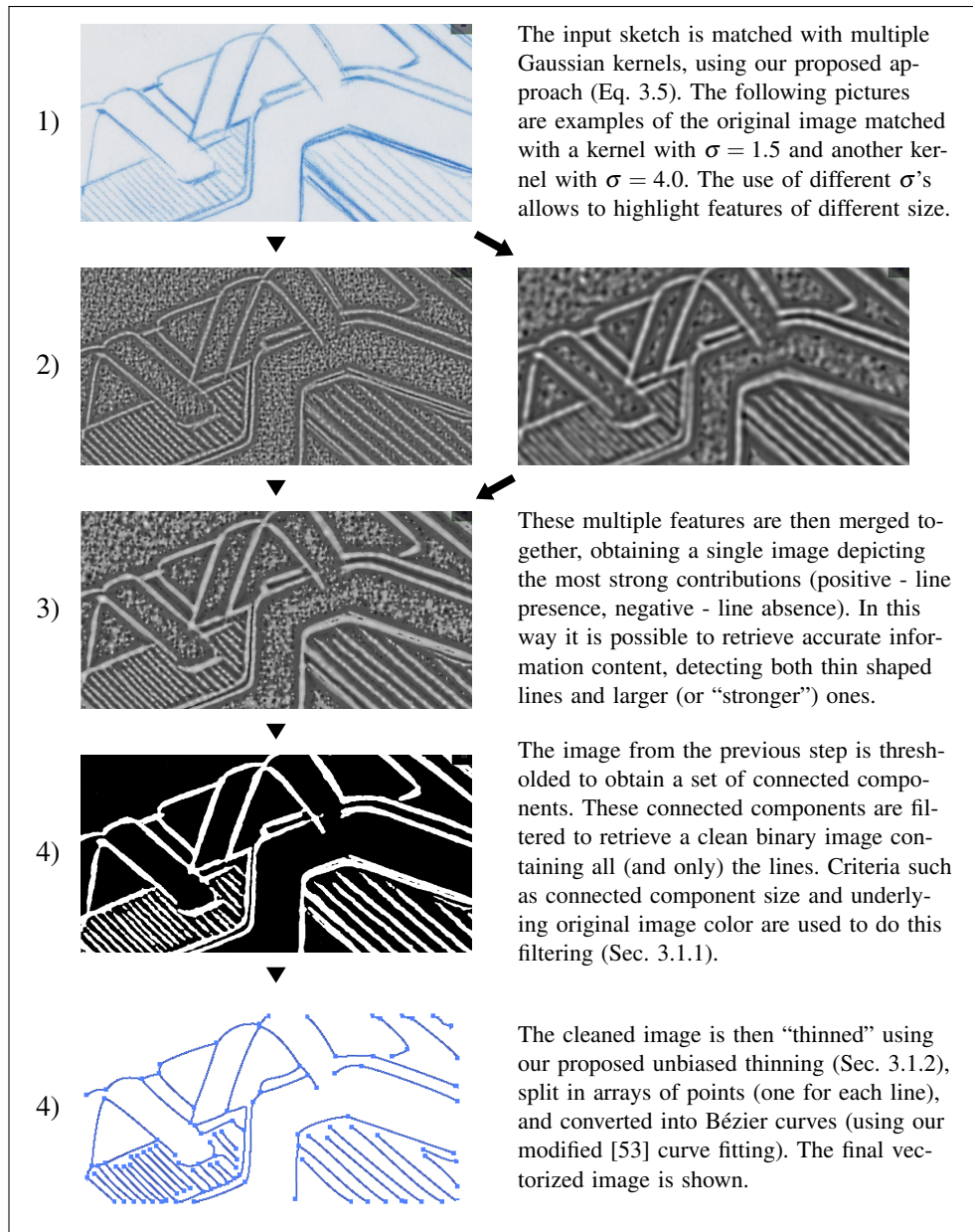


Figure 3.2: An overview of the system: vectorization of a portion of shoe sketch.



Figure 3.3: Perpendicular sections of a pencil line can be well approximated by a bell-like function (e.g. Gaussian or arc of circumference). The trait intensity has a strong correlation with the pencil tip that traced it.

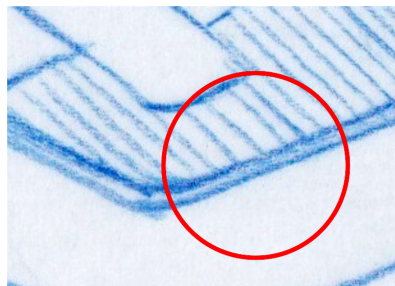


Figure 3.4: Detecting two parallel lines could be just a matter of stroke hardness and surrounding context.

pressing the *similarity* between two signals (or *images* in the discrete 2D space), but it presents several limitations, i.e. dependency on the sample average, the scale and the vector's sizes. To address these problems, pcc between two samples a and b can be defined as:

$$pcc(a, b) = \frac{cov(a, b)}{\sigma_a \sigma_b} \quad (3.1)$$

where $cov(a, b)$ is the covariance between a and b , and σ_a and σ_b are their standard deviations. From the definitions of covariance and standard deviation, eq. 3.1 can be written as follows:

$$pcc(a, b) = pcc(m_0 a + q_0, m_1 b + q_1) \quad (3.2)$$

$\forall q_{0,1} \wedge \forall m_{0,1} : m_0 m_1 > 0$. Eq. 3.2 implies invariance to affine transformations. Another strong point in favor of PCC is that its output value is of immediate interpretation. In fact, it holds that $-1 \leq pcc(a, b) \leq 1$. $pcc \approx 1$ means that a and b are very correlated, whereas $pcc \approx 0$ means that they are not correlated at all. On the other hand, $pcc \approx -1$ means that a and b are strongly inversely correlated (i.e., raising a will decrease b accordingly).

PCC has been used in the image processing literature and in some commercial machine vision applications, but mainly as an algorithm for object detection and tracking ([54], [55]). Its robustness derives from the properties of illumination and reflectance, that apply to many real-case scenarios involving cameras. Since the main lighting contribution from objects is linear ([56]), pcc will give very consistent results for varying light conditions, because of its affine transformations invariance (eq. 3.2), showing independence from several real-world lighting issues.

PCC grants much-needed robustness in detecting lines under major changes in illumination conditions, for instance when images can potentially be taken from different devices, such as a smartphone, a satellite, a scanner, an x-ray machine, etc.. In our application domain, at the best of our knowledge, this is the first work proposing to use PCC for accurate line extraction from hand-drawn sketches. Besides robustness to noise, this also allows to work with different “sources” of the lines: from hand-drawn sketches, to fingerprints, to paintings, to corrupted textbook characters,

etc.. In other words, the use of PCC makes our algorithm generalized and applicable to many different scenarios.

Template matching with Pearson's Correlation Coefficient

In order to obtain the punctual PCC between an image I and a smaller template T , for a given point $p = (x, y)$, the following equation can be used:

$$pcc(I, T, x, y) = \frac{\sum_{j,k} (I_{xy}(j, k) - u_{I_{xy}}) (T(j, k) - u_T)}{\sqrt{\sum_{j,k} (I_{xy}(j, k) - u_{I_{xy}})^2 \sum_{j,k} (T(j, k) - u_T)^2}} \quad (3.3)$$

$\forall j \in [-T_w/2; T_w/2]$ and $\forall k \in [-T_h/2; T_h/2]$, and where T_w and T_h are the width and the height of the template T , respectively. I_{xy} is a portion of the image I with the same size of T and centered around $p = (x, y)$. $u_{I_{xy}}$ and u_T are the average values of I_{xy} and T , respectively. $T(j, k)$ (and, therefore, $I_{xy}(j, k)$) is the pixel value of that image at the coordinates j, k computed from the center of that image.

The punctual PCC from eq. 4.1 can be computed for all the pixels of the input image I (except for border pixels). This process will produce a new image depicting how well each pixel of image I resembles the template T . In the remainder of the chapter, we will call it *PCC*. In Fig. 3.5 you see *PCCs* obtained with different templates. It is worth remembering that $PCC(x, y) \in [-1, 1], \forall x, y$.

Choosing a Kernel for line extraction

To achieve line extraction, we will use *PCC* with a suitable template, or *kernel*. Intuitively, the best kernel to find lines would be a sample approximating a “generic” line. A good generalization of a line might be a 1D Gaussian kernel replicated over the y coordinate, i.e.:

$$KLine(x, y, \sigma) = gauss(x, \sigma)$$

Since this is a vertical Gaussian kernel, we would also need all of its rotations, in order to match different orientations of lines. This kernel achieves good detection results for simple lines, which are composed of clear (i.e., well separable from the

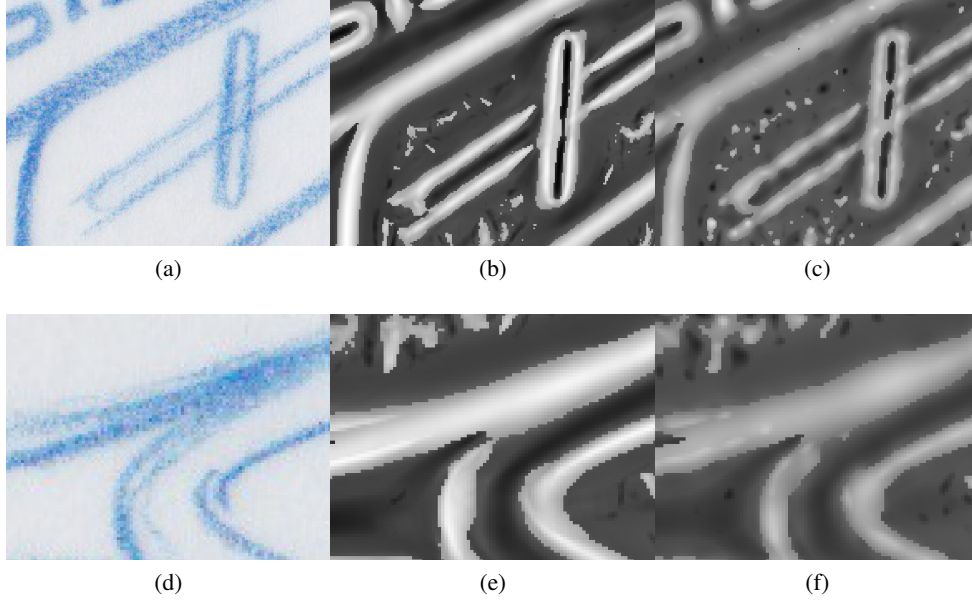


Figure 3.5: Two examples of *PCC* images obtained with different kernels. These pictures show that using a line-shaped kernel (*KLine*) can be detrimental for retrieval quality: (b), (e); crossing lines are truncated or detected as thinner than they should be. Using *KDot* can alleviate the problem: (c), (f); this kernel detects more accurately ambiguous junctions.

background) and separated (from other lines) points. Unfortunately, this approach can give poor results in the case of multiple overlapping or perpendicularly-crossing lines. In particular, when lines are crossing, just the “stronger” would be detected around the intersection point. If both lines have about the same intensity, both lines would be detected, but with an incorrect width (extracted thinner than they should be). An example is shown in the middle column of Fig. 3.5.

Considering these limits, a full symmetric 2D Gaussian kernel might be more appropriate:

$$KDot(x, y, \sigma) = gauss(x, \sigma) \cdot gauss(y, \sigma)$$

This kernel would also have the plus of being fully isotropic. Experimental tests have also proven that it solves the concerns raised with *KLine*, as shown in the rightmost column of Fig. 3.5. In fact, this kernel resembles a dot, and considering a line as a continuous stroke of dots, it will approximate our problem just as well as the previous kernel. Moreover, it behaves better in line intersections, where intersecting lines become (locally) *T-like* or *plus-like* junctions, rather than simple straight lines. Unfortunately, this kernel will also be more sensitive to noise.

Multi-scale size invariance

One of the main objectives of this proposal is to detect lines without requiring many parameters or custom, domain-specific techniques. We also aim to detect both small and large lines that might be mixed together, as it happens in many real drawings. In order to achieve invariance to line widths, we will use kernels of different sizes.

We will generate N Gaussian kernels, each with its σ_i . In order to find lines of width w a sigma of $\sigma_i = w/3$ would work, since a Gaussian kernel gives a contribution of about 84% of samples at $3 \cdot \sigma$.

We use a multiple-scale approach similar to the pyramid used by the SIFT detector [57]. Given w_{min} and w_{max} as, respectively, the minimum and maximum line width to be detected, we can set $\sigma_0 = w_{min}/3$ and $\sigma_i = C \cdot \sigma_{i-1} = C^i \cdot \sigma_0$, $\forall i \in [1, N-1]$, where $N = \log_C(w_{max}/w_{min})$, and C is a constant factor or base (e.g., $C = 2$). Choosing a different base C (smaller than 2) for the exponential and the logarithm will give a finer granularity.

The numerical formulation for the kernel will then be:

$$KDot_i(x, y) = gauss(x - S_i/2, \sigma_i) \cdot gauss(y - S_i/2, \sigma_i) \quad (3.4)$$

where S_i is the kernel size and can be set as $S_i = next_odd(7 \cdot \sigma_i)$, since the Gaussian can be well reconstructed in $7 \cdot \sigma$ samples.

This generates a set of kernels that we will call *KDots*. We can compute the correlation image *PCC* for each of these kernels, obtaining a set of images *PCCdots*, where $PCCdots_i = pcc(Image, KDots_i)$ with *pcc* computed using eq. 4.1.

Merging results

Once the set of images $PCCdots$ is obtained, we will merge them into a single image that will uniquely express the probability of line presence for a given pixel of the input image. This merging is obtained as follows:

$$MPCC(x,y) = \begin{cases} \max PCC_{xy}, & \text{if } |\max PCC_{xy}| > |\min PCC_{xy}| \\ \min PCC_{xy}, & \text{otherwise} \end{cases} \quad (3.5)$$

where

$$\begin{aligned} \min PCC_{xy} &= \min_{\forall i \in [0, N-1]} PCCdots_i(x,y), \\ \max PCC_{xy} &= \max_{\forall i \in [0, N-1]} PCCdots_i(x,y). \end{aligned}$$

Given that $-1 \leq pcc \leq 1$ for each pixel, where ≈ 1 means strong correlation and ≈ -1 means strong inverse correlation, eq. 3.5 tries to retain only confident decisions: “it is definitely a line” or “it is definitely NOT a line”.

By thresholding $MPCC$ of eq. 3.5, we obtain the binary image *LinesRegion*. The threshold has been set to 0.1 in our experiments and resulted to be stable in different scenarios.

Post-processing Filtering

The binary image *LinesRegion* will unfortunately still contain incorrect lines due to the random image noise. Some post-processing filtering techniques can be used, for instance, to remove too small connected components, or to delete those components for which the input image is too “white” (no strokes present, just background noise).

For post-processing hand-drawn sketches, we first apply a high-pass filter to the original image, computing the median filter with window size $s > 2 \cdot w_{max}$ and subtracting the result from the original image value. Then, by using the well-known [58] method, the threshold that minimizes black-white intraclass variance can be estimated and then used to keep only the connected components for which the corresponding gray values are lower (darker stroke color) than this threshold. A typical output example can be seen in Fig. 3.6.

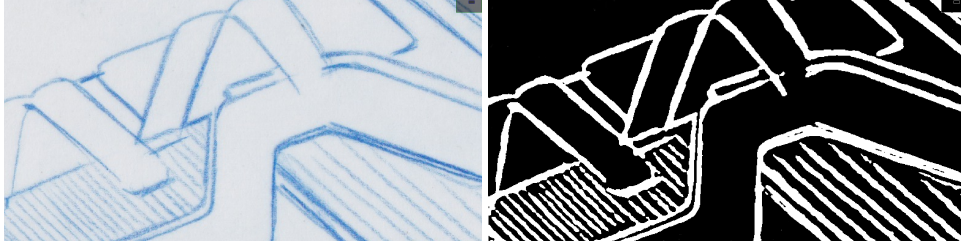


Figure 3.6: Part of a shoe sketch and its extracted *LinesRegion* (after postprocessing).

3.1.2 Thinning

An extracted line shape is a “clean” binary image. After post-processing (holes filling, cleaning) it is quite polished. Still, each line has a varying, noisy width, and if we want to proceed towards vectorization we need a clean, compact representation. The natural choice for reducing line shapes to a compact form is to apply a thinning algorithm [48].

Several thinning variants are well described in the review [49]. In general terms, thinning algorithms can be classified in one-pass or multiple-passes approaches. The different approaches are mainly compared in terms of processing time, rarely evaluating the accuracy of the respective results. Since it is well-known, simple and extensively tested, we chose [59]’s algorithm as baseline. However, any iterative, single-pixel erosion-based algorithm will work well for simple skeletonizations.

Unfortunately, Zhang and Suen’s algorithm presents an unwanted effect known as “skeletonization bias” [49] (indeed, most of the iterative thinning algorithms produce biased skeletons). In particular, along steep angles the resulting skeleton may be wrongly shifted, as shown in Fig. 3.7. The skeleton is usually underestimating the underlying curve structure, “cutting” curves too short. This is due to the simple local nature of most iterative, erosion-based algorithms. These algorithms usually work by eroding every contour pixel at each iteration, with the added constraint of preserving full skeleton connectivity (not breaking paths and connected components). They do that just looking at a local 8-neighborhood of pixels and applying masks. This works quite well in practice, and is well suited for our application, where the shapes

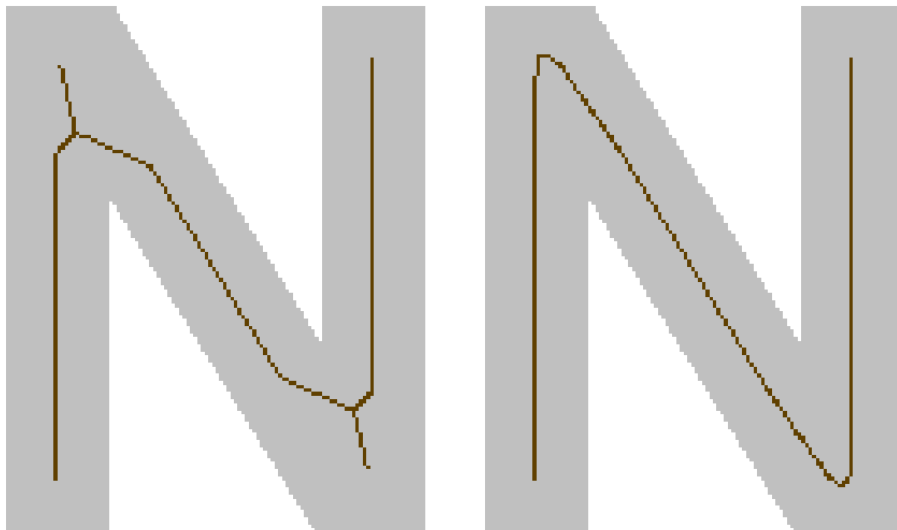


Figure 3.7: Example of the biasing effect while thinning a capital letter *N* (on the left). On the right, the ideal representation of the shape to be obtained.

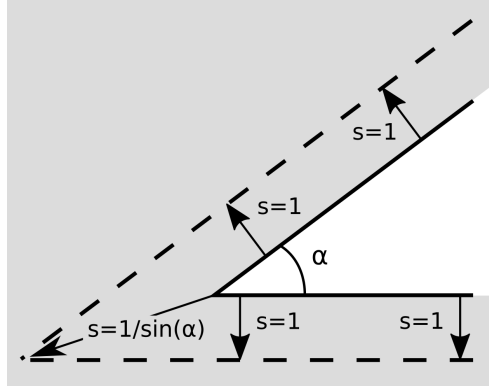


Figure 3.8: Applying an equal erosion to all the points of a concave shape implies eroding at a faster speed alongside steep angles. A speed of $s = 1/\sin(\alpha)$ must be applied.

to be thinned are lines (shapes already very similar to a typical thinning result). The unwanted bias effect arises when thinning is applied to strongly-concave angles. As described by the work of Chen [60], the bias effect appears when the shape to be thinned has a contour angle steeper (lower) than 90 degrees.

To eliminate this problem, we developed our custom unbiased thinning algorithm. The original proposal in [60] is based, first, on the detection of the steep angles and, then, on the application of a “custom” local erosion specifically designed to eliminate the so-called “hidden deletable pixels”. We propose a more rigorous method that generalizes better with larger shapes (where a 8-neighbors approach fails).

Our algorithm is based on this premise: a standard erosion thinning works equally in each direction, eroding one pixel from each contour at every iteration. However, if that speed of erosion (1 pixel per iteration) is used to erode regular portions of the shape, a faster speed should be applied at steep angle locations, if we want to maintain a well proportioned erosion for the whole object, therefore extracting a more correct representation of the shape.

As shown in Fig. 3.8, an erosion speed of:

$$s = 1/\sin(\alpha)$$

should be applied at each angle point that needs to be eroded, where α is the angle size. Moreover, the erosion direction should be opposite to the angle bisector. In this way, even strongly concave shapes will be eroded uniformly over their whole contours.

The steps of the algorithm are the following:

- First, we extract the contour of the shape to be thinned (by using the border-following algorithm described by [47]). This *contour* is simply an array of 2d integer coordinates describing the shape outlines. Then, we estimate the curvature (angle) for each pixel in this contour. We implemented the technique proposed in [61], based on cord distance accumulation. Their method estimates the curvature for each pixel of a contour and grants good generalization capabilities. Knowing each contour pixel supposed curvature, only pixels whose angle is steeper than 90 degrees are considered. To find the approximate angle around a point of a contour the following formula is used:

$$\alpha \approx 6I_L/L^2$$

where I_L is the distance accumulated over the contours while traveling along a cord of length L . Fig. 3.9a shows an example where concave angles are represented in green, convex in red, and straight contours in blue. Han *et al.*'s method gives a rough estimate of angle intensity, but does not provide its direction. To retrieve it, we first detect the point of local maximum curvature, called P_E . Starting from it, the contours are navigated in the left direction, checking curvature at each pixel, until we reach the end of a straight portion of the shape (zero curvature - blue in Fig. 3.9a), which presumably concludes the angular structure (see Fig. 3.9). This reached point is called P_L , the left limit of the angle. We do the same traveling right along the contour, reaching the point that we call P_R . These two points act as the angle surrounding limits.

- We then estimate the precise 2D direction of erosion and speed at which the angle point should be eroded. Both values can be computed by calculating the angle between segment $\overline{P_L P_E}$ and segment $\overline{P_E P_R}$, that we call α_E . As already

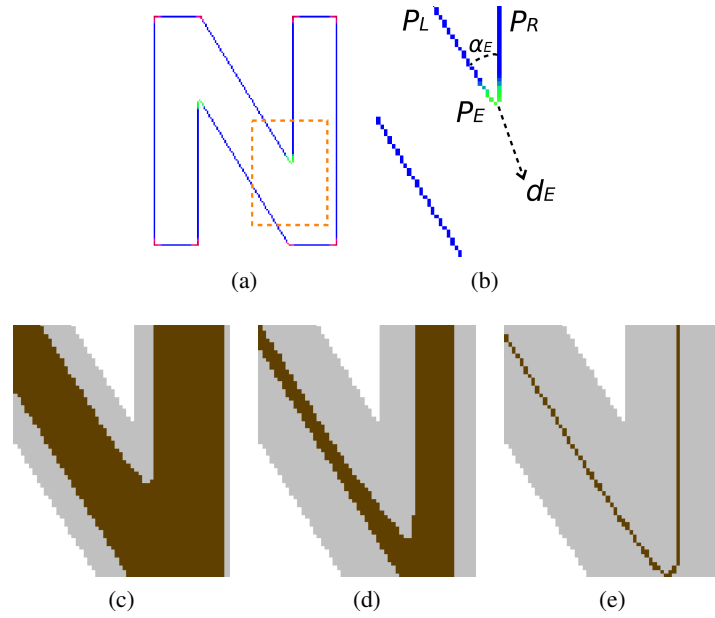


Figure 3.9: (a) Curvature color map for the contours. Concave angles (navigating the contour clockwise) are presented in green, convex angles in red. Blue points are zero-curvature contours. (b) A portion of the contours curvature map, and a real example of the erosion-thinning process steps (c), (d), (e). The green part of (b) is the neighborhood N_{P_E} that will be eroded at the same time along d_E direction.

said, the direction of erosion d_E is the opposite of α_E bisector, while the speed is $s_E = 1/\sin(\alpha_E)$.

- After these initial computations, the actual thinning can start. Both the modified faster erosion of P_E and the classical iterative thinning by [59] are run in parallel. At every classical iteration of thinning (at speed $s = 1$), the point P_E is moved along its direction d_E at speed s_E , eroding each pixel it encounters on the path. The fact that P_E is moved at a higher speed compensates for the concaveness of the shape, therefore performing a better erosion of it. Figs. 3.9c, 3.9d and 3.9e show successive steps of this erosion process.
- Additional attention should be posed to not destroy the skeleton topology; as a consequence, the moving P_E erodes the underlying pixel only if it does not break surrounding paths connectivity. Path connectivity is checked by applying four rotated masks of the hit-miss morphological operator, as shown in Fig. 3.10. If the modified erosion encounters a pixel which is necessary to preserve path connectivity, the iterations for that particular P_E stop for the remainder of the thinning.

To achieve better qualitative results, the faster erosion is performed not only on the single P_E point, but also on some of its neighbor points (those who share similar curvature). We call this neighborhood set of points N_{P_E} and are highlighted in green in Fig. 3.9b. Each of these neighbor points P_i should be moved at the same time with appropriate direction, determined by the angle α_i enclosed by segments $\overline{P_L P_i}$ and $\overline{P_i P_R}$. In this case, it is important to erode not only P_i , but also all the pixels that connect it (in straight line) with the next P_{i+1} eroding point. This is particularly important because neighbor eroding pixels will be moving at different speeds and directions, and could diverge during time.

As usual for thinning algorithms, thinning is stopped when, after any iteration, the underlying skeleton has not changed (reached convergence).

1				1				1		
	1	0			1	0		0	1	
	0	1			0	1		1	0	
	1							1		
0	1	0		0	1	0		1	1	1
	1			1	0	1			1	

Figure 3.10: Four rotations of these masks of a hit-miss operator (1 = hit, 0 = miss, empty = ignored) are used to detect pixels necessary to preserve path connectivity.

3.1.3 Creating and improving paths

Path creation

The third major step towards vectorization is transforming the thinned image (a binary image representing very thin lines) in a more condensed and hierarchical data format. A good, simple representation of a thinned image is the set of paths contained in the image (also called “contours”). A *path* is defined as an array of consecutive 2D integer points representing a single “line” in the input thinned image:

$$path = [(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})]$$

In this work, sub-pixel accuracy is not considered. A simple way to obtain sub-pixel accuracy could be zooming the input image before thinning and extracting real valued coordinates.

The successive steps of skeleton analysis will rely on 8-connectivity, so we need to convert the output of our thinning algorithm to that format. Our algorithm, being based on Zhang-Suen’s thinning, produces 4-connected skeletons (with lines wide 1-2 pixels). A 4-connected skeleton can be transformed into a 8-connected one by applying the four rotations of the mask shown in Fig. 3.11. Pixels that match that mask must be deleted in the original skeleton (*in-place*). Applying an *in-place* hit-

	1	
	1	1
0		

Figure 3.11: Four rotations of this simple mask of *in-place* hit-miss morph-operator (1 = hit, 0 = miss, empty = ignored) are used to transform a thinned image to a “strictly 8-connected” one.

miss operator implies reading and writing consecutively on the same input skeleton.

The resulting thinned image is called a “strictly 8-connected skeleton”: this definition does not imply that a pixel can not be connected with its 4-neighbors, but that each 4-neighbor pixel has been erased if not needed for topology preservation.

Path classification

Once the “strictly 8-connected skeleton” is obtained, the path classification step is issued. In order to better understand how this step works, let us define some basic concepts. We first define a *junction* between paths. A junction is a pixel that connects three or more paths. By deleting a junction pixel, paths that were connected are then split. We can detect all the junctions in the strictly 8-connected skeleton by applying the masks in Fig. 3.12 (and their rotations) in a hit-miss operator. In this case, the algorithm must be performed with an input image (read only) and an output image (write only) and the order of execution does not matter (i.e., it can be executed in parallel).

Sometimes, for peculiar input conformations, a junction pixel could be neighbor of another junction (see Fig. 3.13). These specific junctions are merged together for successive analysis. Each path connecting to one of these junctions is treated as connecting to all of them. For simplicity, we choose one of them as representative.

Similarly, an *endpoint* of a path can be defined as a pixel connected with one and only one other pixel, which needs to be part of the same path. Basically, an endpoint corresponds to either the starting or the ending point of a path. Endpoints can be

Figure 3.12: Four rotations of these masks of a hit-miss operator (1 = hit, 0 = miss, empty = ignored) are used to detect all the junctions in a “strictly 8-connected skeleton”.

Figure 3.13: Examples of adjacent junctions (highlighted). Each of these junctions can not be deleted without changing the underlying topology, but can be treated as one.

	1		1	0	0
0	1	0	0	1	0
0	0	0	0	0	0

Figure 3.14: Four rotations of these masks of a hit-miss operator (1 = hit, 0 = miss, empty = ignored) are used to detect all the endpoints in a “strictly 8-connected skeleton”.

straightforwardly found by applying the hit-miss morphological operator with the rotated versions of the masks reported in Fig. 3.14.

Given these definitions and masks, we can detect all the endpoints and the junctions of the strictly 8-connected skeleton. Then, starting from each of these points and navigating the skeletons, we will at the end reach other endpoints or junctions. By keeping track of the traversed pixels we can compose paths, grouped in one of these categories:

- paths from an endpoint to another endpoint ($e \leftrightarrow e$);
- paths from a junction to endpoint ($j \leftrightarrow e$);
- paths from a junction to a junction ($j \leftrightarrow j$).

There exists one last type of path that can be found, i.e. the closed path. Obviously, a closed path has no junctions or endpoints. In order to detect closed paths, all the paths that cannot be assigned to one of the three above groups are assigned to this category.

Examples of all types of paths are reported in Fig. 3.15. Fig. 3.16 also shows a notable configuration, called “tic-tac-toe” connectivity. In this case, an assumption is made, i.e. paths cannot overlap or cross each other. If this happens, paths are treated as distinct paths. In order to recognize that two crossing paths actually belong to the same “conceptual” path, a strong semantic knowledge of the underlying structures is necessary, but this is beyond the scope of this work. A treatment of the problem can be found in [62], where they correctly group consecutive intersecting line segments.

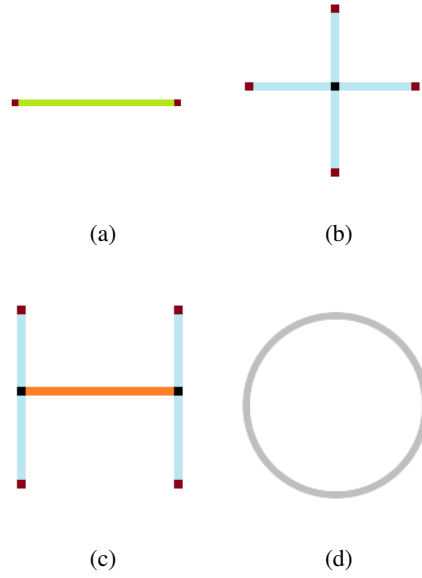


Figure 3.15: Examples of the four types of paths: $e \leftrightarrow e$ (a), $e \leftrightarrow j$ (b), $j \leftrightarrow j$ (c) and closed (d). Junctions and endpoints are highlighted.

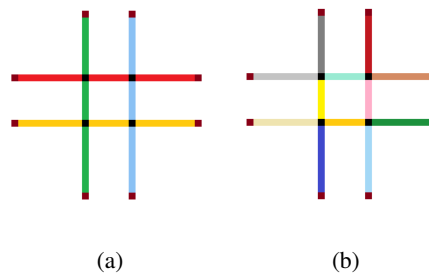


Figure 3.16: Examples of semantic-aware connectivity (a), where only four paths are detected, opposed to “tic-tac-toe” connectivity (b), where twelve small paths are detected. In this second case, overlapping paths are treated separately as different paths and no semantic knowledge of the context is retained.

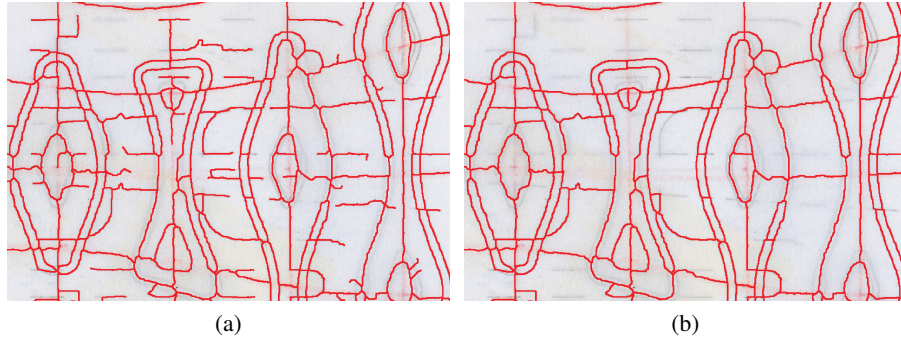


Figure 3.17: An example of *pruning* applied to an input image (a). Small branches are deleted from the resulting image (b).

Path post-processing

The resulting paths are easy to handle as they contain condensed information about the underlying shapes. However, they can often be further improved using some prior knowledge and/or visual clues from the original sketch image. Examples of these improvements are *pruning*, *merging* and *linking*.

Pruning consists in deleting small, unwanted “branches” from the thinning results. The skeleton is usually composed by many long paths that contain most of the information needed. However, also smaller, unwanted “branches” are often present. They may be artifacts of thinning, or resulting from noise in the original image. By applying the pruning, branches shorter than a given length l can be deleted. Branches are paths of the first or second type: $e \leftrightarrow e$ or $j \leftrightarrow e$. They can not belong to the third type, $j \leftrightarrow j$, because we do not want to alter skeleton connectivity by deleting them. An example of pruning is reported in Fig. 3.17. Pruning can be performed with different strategies, by deciding to keep more or less details. One simple idea is to iteratively prune all the image with increasing branch length threshold l .

Merging is the process of grouping together junctions or paths. Junctions that are close each other could be grouped together in a single junction, simplifying the overall topology. After doing that, the same can be done for paths. Parallel and near paths that start and end in the same junctions are good candidates for merging (see

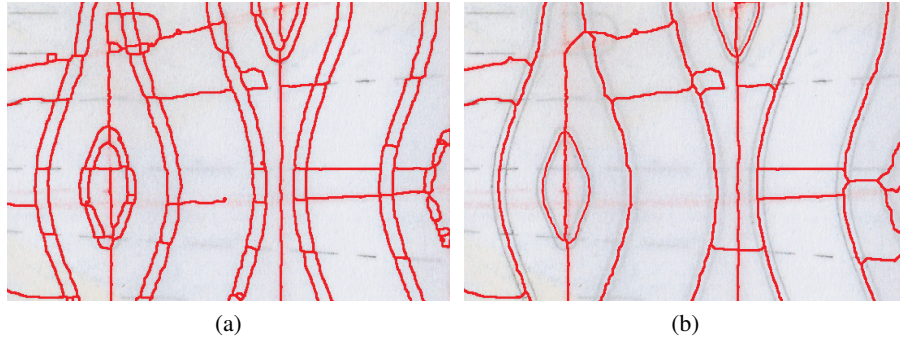


Figure 3.18: An example of *merging* applied to an input image (a). Parallel paths are combined together in the resulting image (b).

an example in Fig. 3.18).

Linking (or endpoint linking, or edge linking) is the technique of connecting two paths whose endpoints are close in order to create a single path. Besides the endpoints distance, a good criteria for linking could be the path directions at their endpoints. Incident and near paths are suitable to be linked into a single, more representative, path (an example is reported in Fig. 3.19).

In order to improve their accuracy, all these post-processing techniques might benefit from the data (pixel values) from the original color image.

3.1.4 Vectorization process

Once cleaned paths are obtained, they need to be converted in a vectorized version with the minimum number of points. The basic vectorization algorithm used was firstly introduced in [53]. Schneider's algorithm tries to solve a linear system (with least squares method), fitting a Bézier curve for each of the obtained paths. In detail, it tries iteratively to find the best set of cubic Bézier parameters that minimize the error, defined as the maximum distance of each path point from the Bézier curve. At each iteration, it performs a Newton-Raphson re-parametrization, to adapt the underlying path representation to the Bézier curve, in order to create a more suitable linear system to be optimized.

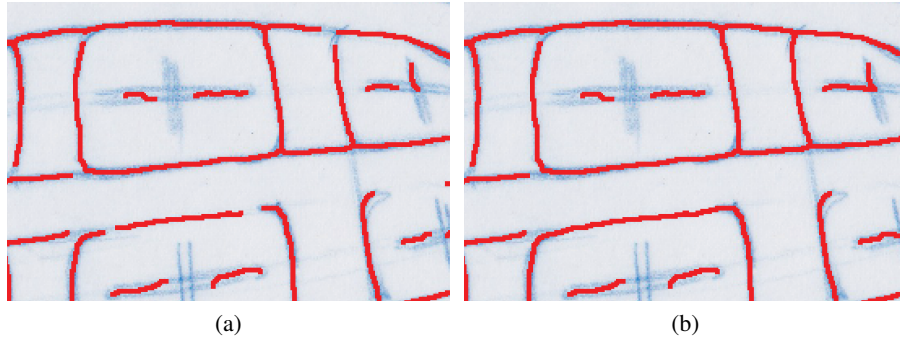


Figure 3.19: An example of endpoints *linking* applied to an input image (a). Paths with adjacent endpoints have been connected in the output (b).

The algorithm is parametrized with a desired error to be reached, and a maximum number of allowed iterations. Whenever it converges, the algorithm returns the four points representing the best-fitting Bézier curve. If the convergence is not reached within the maximum number of iterations, the algorithm splits the path in two and recursively looks for the best fitting for each of them separately. The path is split around the point which resulted to have the maximum error w.r.t. the previous fitting curve. An additional constraint is related to C^1 continuity of the two resulting curves on the splitting point, in order to be able to connect them smoothly.

In order to be faster, the original algorithm skips automatically all the curves which do not correspond to a minimum error (called “iteration-error” Ψ), and proceeds to the splitting phase without trying the Newton-Raphson re-parametrization. However, this simplification also affects accuracy of the vectorization, by generating a more complex representation due to the many split points created. Therefore, since the computational complexity is not prohibitive (worst case is $O(n \cdot \log n)$, with n being the path length), we modified the original algorithm by removing this simplification.

Conversely, another early-stop condition has been introduced in our variant. Whenever the optimization reaches an estimation error lower than a threshold after having run at least a certain number of iterations, the algorithm stops and the estimated

Bézier curve is returned. This can be summarized by the following condition:

```
if current_err < desired_err * f and current_iter > tot_iter * f:
    return current_bezier
```

where “f” is an arbitrary fraction (set to 0.1 in our experiments). This condition speeds up the algorithm if the paths are easily simplified (which is common in our case), while the full optimization process is run for “hard” portions that need more time to get good representations.

We also extended Schneider’s algorithm to work for closed paths. First, C1 continuity is imposed for an arbitrary point of the path, selected as the first (as well as the last) point of the closed path. A first fit is done using the arbitrary point. Then, the resulting point with the maximum error w.r.t. the fitted curve is selected as a new first/last point of the path and the fitting algorithm is run a second time. In this way, if the closed path has to be fitted to two or more Bézier curves, the split point will be the one of highest discontinuity, not a randomly chosen one.

3.2 Experiments

3.2.1 Line extraction

Assessing the accuracy of a line extraction method is not an easy task. Results can be subjectively considered as good or bad, depending on the final objective of the overall system. Having in mind our final objective (sketch vectorization of hand-drawn samples for fashion industry), we started by using a large dataset of hand-drawn shoe sketches (courtesy of Adidas AGTM). These sketches have been drawn from expert designers using different pens/pencils/tools, different styles and different backgrounds (thin paper, rough paper, poster board, etc.). Each image has its peculiar size and resolution, and has been taken from scanners or phone cameras.

However, this dataset does not come with a manual ground truth, preventing us to conduct a quantitative (together with a qualitative or visual) evaluation of the results. Manually segmenting complex hand-drawn images such as that reported in Fig. 3.20, last row, to obtain the ground truth to compare with, is not only tedious, but also very

prone to subjectivity. Therefore, we searched for large and public datasets (possibly with an available ground truth) to be used in this evaluation. One possible solution is the use of the SHREC13 - “Testing Sketches” dataset [5], whereas alternatives are Google Quick Draw and Sketchy dataset [63]. SHREC13 contains very clean, single-stroke “hand-drawn” sketches (created using a touch pad or mouse), such as those reported in Figs. 3.21a and 3.21c. It is a big and representative dataset: it contains 2700 sketches divided in 90 classes and drawn by several different authors. Unfortunately, these images are too clean to really challenge our algorithm, resulting in almost perfect results. The same can be said for Quick Draw and Sketchy datasets.

To fairly evaluate the ability of our and other algorithms to extract lines in more realistic situations, we have created a “simulated” dataset, called *inverse dataset*. The original SHREC13 images are used as ground truth and processed with a specifically-created algorithm with the aim of “corrupting” them and recreating as closely as possible different drawing styles, pencils and paper sheets. More specifically, this algorithm randomly selects portions of each ground truth image, and moves/alters them to generate simulated strokes of different strength, width, length, orientation, as well as multiple superimposed strokes, crossing and broken lines, background and stroke noise. The algorithm for *inverse dataset* creation works as follows:

- Select any of the SHREC13 images, resize it to 2000x2000 and convert the image to one-pixel thinned paths. Paths are white, while image background is black.
- Select a random circle from that image. The circle center is randomly chosen from the whole image coordinates, and circle diameter is selected as *random_uniform*(135,195). This should simulate a typical pencil trait length.
- Choose a random brush type, brush width and pressure. Brush type can be ball shaped (pen) or bell shaped (pencil). Brush width is selected as *random_uniform*(4.0,20.0). Pressure is selected as *random_uniform*(0.3,1.7)
- Paint the selected circle to the output image. This is done by blurring the thinned input ground truth with the ball or bell shaped tool (convolution fil-

ter). This blurred portion circle of the input image is copied over the output painting, multiplying it with the value of *pressure*. A lower pressure is applied to the start and end portion of each trait (first-last 10%); usually this is the case with real traits.

- To better simulate hand strokes uncertainty and multiple strokes, a random shift and rotation is applied before drawing the trait to the output (an x, y translation of $random_uniform(-15, +15)$ pixels, a rotation of $random_uniform(-5, +5)$ degrees). The random shifted trait is added over the output image.
- This process is repeated until the whole image has been copied (choosing many random circles). An additional constraint is applied: allowing for a maximum of multiple strokes drawn over the same portion of output image. For example, we choose a maximum of 3-5 overlapping strokes (three as a general rule, five allowed if needed to complete copying the last missing bits of the input image).
- After the image has been completely copied, a final global phase is performed to apply multiple kinds of global noise. This is done to simulate: pencil tip roughness, paper porosity, paper non-uniform background. We define a Gaussian noise image $GNI(\sigma)$ as follows: create a uniform noise image assigning a $random_uniform(0.0, 1.0)$ to each pixel. Smooth that image with a Gaussian filter of given σ , and normalize the resulting image to have mean and deviation of 0 and 1. It is worth noting that this noise should challenge our Gaussian correlation matching. Then, we define a exponential Gaussian noise image $EGNI(\sigma, s)$ pixel per pixel as following:

$$EGNI(\sigma, s)_{x,y} = s^{GNI(\sigma)_{x,y}} \quad (3.6)$$

$\forall x, y$ composing GNI . This form of exponential noise is useful when multiplying different noises together, as s models the noise strength. As a consequence:

- Pencil roughness is simulated by multiplying the traits with a $EGNI(\sigma = 0.5, s = 1.5)$. Only traits are multiplied, by applying a threshold in order to detect traits.

- Paper porosity is simulated by multiplying the image with a $EGNI(\sigma = 1.0, s = 1.2)$.
- Paper non-uniform background is simulated by multiplying the image with a $EGNI(\sigma = 101.0, s = 2.0)$. In this case, before applying $EGNI$ to the image, $EGNI$ is previously eroded with a morphological operator (gray-levels erosion) of diameter 71.0 to achieve more uniform, smoothed results.
- At this point the simulated pencil sketch has been obtained and it just needs to be scaled in the 0-255 domain (128 mean) and inverted (usually we want white background and dark strokes).

The resulting database has the same size as the original SHREC13 (2700 images), and each picture has been resized to 4 MegaPixels to better simulate real world pencil sketches. Example results (used as inputs for our experiments) are reported in Figs. 3.21b and 3.21d.

We performed visual/qualitative comparisons of our system with the state-of-the-art algorithms reported in [2] and in [4], and with the Adobe IllustratorTM's tool "Live Trace" using different input images. Regarding the algorithm in [2], we used their nice online tool where users can upload their own sketches and get the resulting simplified image back for comparison purposes. For [4] we compared using their *github* code. Results are reported in Fig. 3.20, where it is rather evident that our method performs better than the compared methods in complex and corrupted areas.

To obtain quantitative evaluations we used the "inverse dataset". Precision and recall in line extraction and the *mean Centerline Distance*, similar to the notion of Centerline Error proposed in [19], are used as performance metrics. The results are shown in Tab. 3.1. Our method outperforms Live Trace and strongly beats [2] in terms of recall, while nearly matching its precision. This can be explained by the focus we put in designing a true image color/contrast-invariant algorithm, also designed to work at multiple resolutions and stroke widths. Simo-Serra *et al.* low recall performance is probably influenced by the selection of training data (somewhat specific) and the data augmentation they performed with Adobe IllustratorTM (less general-

ized than ours). They changed this method in their second work [4], obtaining better performance mostly by improving recall. Still, their method exhibits a considerably lower recall than our proposed method. The slightly higher precision shown by [2] and [4] compared to our method is probably due to the fact that those methods work at one extreme of the ROC curve (the higher sensitivity zone). The strenght of our proposal is furtherly highlighted by a global F-measure of **97.8%** for our method w.r.t. **87.0%** of [2] and **90.8%** of [4].

Finally, we compared the running times of these algorithms. We run our algorithm and “Live Trace” using an Intel Core i7-6700 @ 3.40Ghz, while the performance of [2] are extracted from the paper. The proposed algorithm is much faster than Simo-Serra *et al.* method (0.64 sec per image on single thread instead of 19.46 sec on Intel Core i7-5960X @ 3.00Ghz using 8 cores), and offers performance within the same order of magnitude of Adobe IllustratorTM Live Trace (which on average took 0.5 sec). The theoretical complexity of our algorithm is $O(w_{max}^2 * \log(w_{max}) * n)$, where w_{max} is the maximum line width to be detected, and n is the total number of pixels of the image. It is also worth noting that [4] work is quite heavy on RAM usage, in our experience the simplification code needed more than 15 GB of dynamic memory, while our proposed method is very memory efficient (< 100 MB).

3.2.2 Unbiased thinning

In order to assess the quality of the proposed unbiased thinning algorithm, a qualitative comparison with other two thinning algorithms is shown in Fig. 3.22. The

	Precision	Recall	Center. Dist.
Our method	97.3%	98.4%	3.58 px
[2]	98.6%	77.9%	4.23 px
[4]	99.7%	83.3%	7.42 px
Live Trace	85.0%	83.8%	3.73 px

Table 3.1: Accuracy of the three implementations over the *inverse dataset* generated from SHREC13 [5] (2700 images).

other two algorithms are: [59], as a standard, robust, iterative algorithm, and K3M [49], which represents the state of the art for unbiased thinning. In the literature, the standard testing scenario for thinning algorithms is typewriting fonts.

It is quite evident that the proposed algorithm is able to correctly thin the most difficult parts of the characters, in particular along “N” steep angles, “X” crossing, and “Y” intersection, where it reconstructs the original structure of the letters with much more precision than the other algorithms.

Moreover, in all of the examples our algorithm associates the robustness of a standard algorithm (like Zhang-Suen), with the complete removal of biases, and has the additional benefit of working with shapes of arbitrary dimension. In fact, the test images range from the dimensions of 120 to 200 font points. K3M, on the other hand, has been designed to reduce bias but does not show any appreciable result with these shapes; this is because it has been designed to work with very small fonts (< 20 pts), and provides limited benefits for larger shapes.

For shapes that are already almost thin, with no steep angles (like Dijkstra cursive font - second row), the thinning results are similar for all the algorithms.

The unbiasing correction works better when the shapes to be thinned have a constant width (as in Arial and TwCen fonts). If the width of the line changes strongly within the shape, as in Times New Roman (and in general Serif fonts, that are more elaborated), the unbiasing correction is much more uncertain to perform, and our results are similar to the standard thinning by Zhang-Suen.

We can also note that K3M has some stability problems if the shape is big and has aliased contours, such as the rotated Times New Roman. In that scenario it creates multiple small fake branches.

For every example, our algorithm used a cord length accumulation with size of 15 points.

Finally, thinning speed is very similar to the one from [59], on which our algorithm relies, with just a 1-2% variation.

		Time (ms)	# Points
Schneider	10 iter/pix	2860	573
Schneider	1 iter/pix	380	590
Schneider	0.1 iter/pix	85	617
Our approach	1 iter/pix	14020	532
Our approach	0.1 iter/pix	1420	555

Table 3.2: Running time and number of control points generated by the two versions of the vectorization algorithm. Desired error $err = 6$. Less is better.

		Time (ms)	# Points
Schneider	10 iter/pix	4110	904
Schneider	1 iter/pix	550	939
Schneider	0.1 iter/pix	115	983
Our approach	1 iter/pix	14735	663
Our approach	0.1 iter/pix	1483	744

Table 3.3: Running time and number of control points generated by the two versions of the vectorization algorithm. Desired error $err = 3$. Less is better.

3.2.3 Vectorization algorithm

We improved Schneider’s algorithm mainly regarding the quality of the vectorized result. In particular, our objective was to reduce the total number of control points in the vectorized representation, thus creating shapes that are much easier to modify and interact with. This is sometimes called lines “weight”.

Fig. 3.23 shows that we successfully reduced the number of control points w.r.t. Schneider’s original algorithm with a percentage that ranges from 10% to 30% less control points, depending on the maximum desired error. This nice property of fewer control points comes at the cost of increased computational time. In fact, the total run time has increased, but is still acceptable for our purposes: standard Schneider’s algorithms takes about 1 - 4 seconds to vectorize a 5 MegaPixels sketch on a Intel Core i7-6700 @ 3.40Ghz, whereas the improved version takes 2 - 15 seconds. More-

over, Schneider's algorithm total running time is heavily dependent on the desired error (quality), while our version depends mainly on the specified maximum number of iterations. For our experiments we set the number of iterations dynamically for each path to its own path length, in pixels (short paths are optimized faster than long paths).

Tabs. 3.2 and 3.3 show some quantitative analyses about execution times and number of control points obtained. They also show that we could stick with 0.1 iterations per pixel and achieve execution times very similar to the original Schneider's version, while still reducing the number of control points considerably.

Finally, Fig. 3.24 and 3.25 report some challenging cases in which the final vectorization result of our system is reported.



Figure 3.20: Comparisons for the line extraction step: an image from our inverse dataset - first column; a random sketch from the internet, author: Michael Bencik - second column; a real Adidas AGTM hand drawn design - third column. Results are obtained from: a commercial tool in second row, two state of the art algorithms ([2] in third row and [4] in fourth row), and our method in the bottom row.

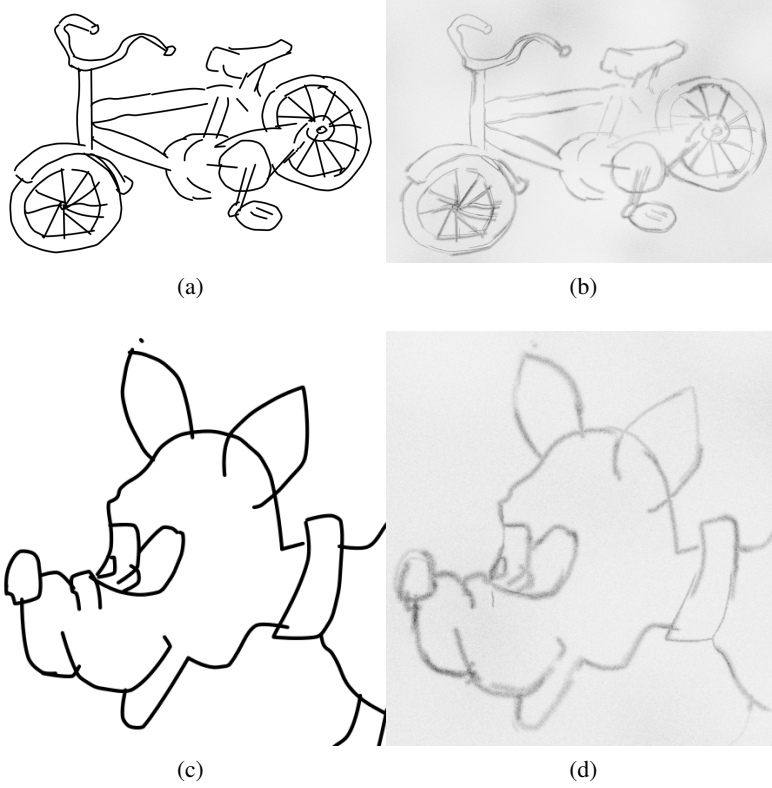


Figure 3.21: Examples of the “inverse dataset” sketches created from SHREC13 [5].

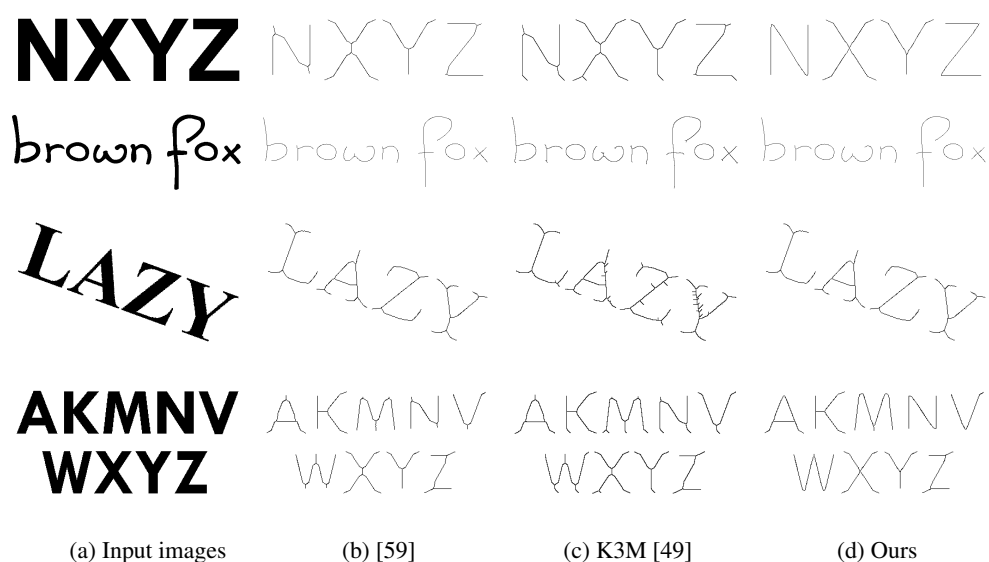


Figure 3.22: Examples of thinning results. (a) The input images. (b) Thinning results with the “standard” Zhang-Suen algorithm’. (c) Results for K3M method (state of the art). (d) Our algorithm results. Fonts used (from the top): 200pts Arial; 180pts Dijkstra; 150pts Times New Roman (30 degrees rotated), 120pts TwCen.

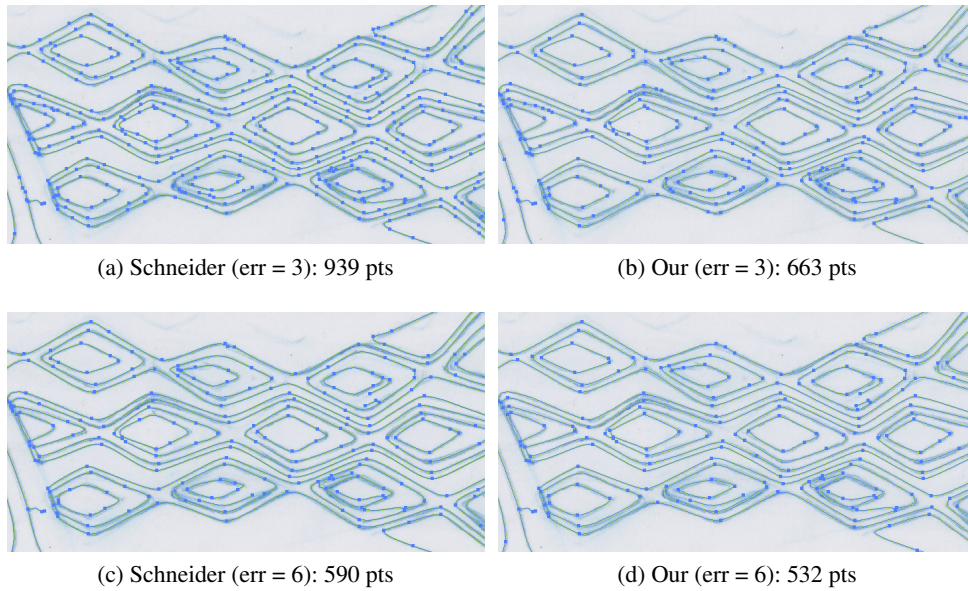
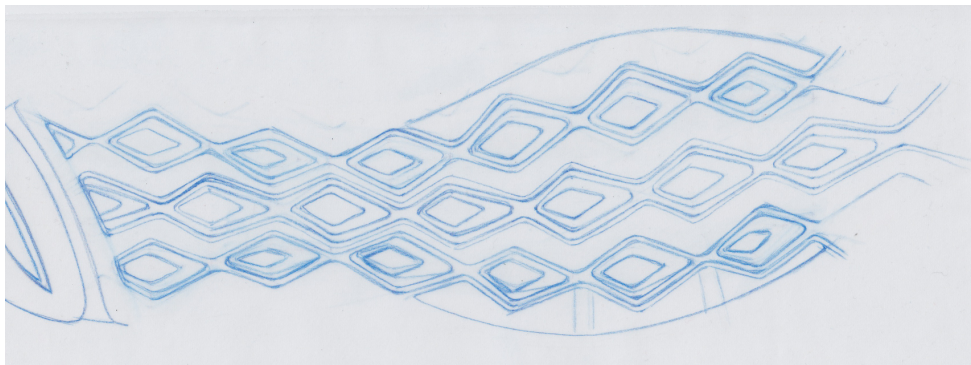
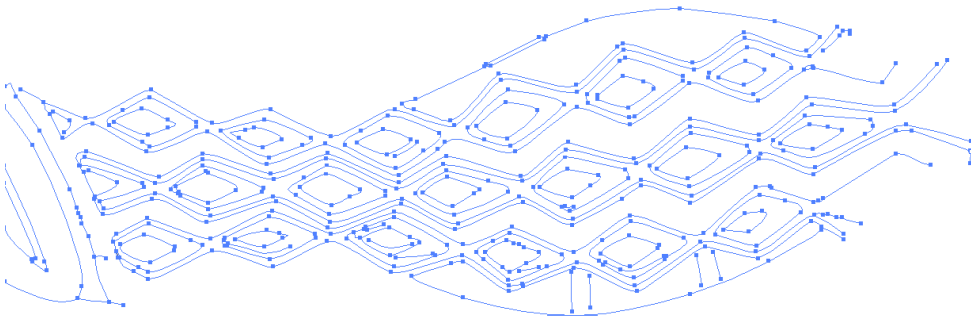


Figure 3.23: Four example portions of vectorization. (a) and (c) use Schneider's stock algorithm, (b) and (d) use our improved version. Our algorithm considerably reduces the number of control points for both cases (err = 3 and err = 6), without decreasing result quality. For all the examples the maximum number of iterations has been dynamically set for each path to its own length in pixels (1 iteration per pixel).

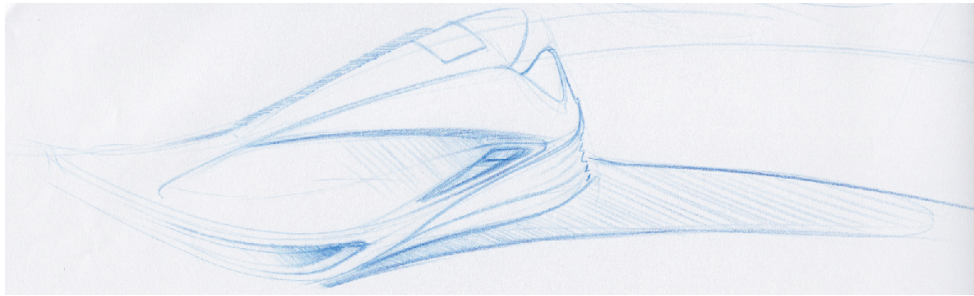


(a)

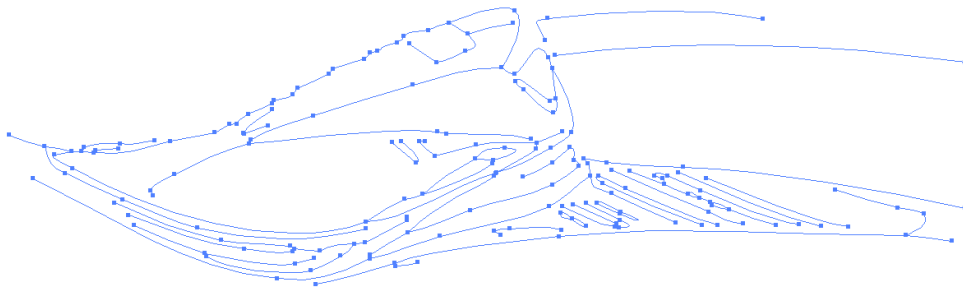


(b)

Figure 3.24: Examples of end-to-end vectorizations performed by our system. An Adidas AGTM shoe sketch (a), (b).



(a)



(b)



(c)



(d)

Figure 3.25: Other examples of end-to-end vectorizations performed by our system. Another difficult Adidas AGTM shoe sketch (a), (b), and a dirty, low resolution, preparatory fashion sketch (c), (d).

Chapter 4

Fashion Product Classification through Deep Learning and Computer Vision

*There are people who believe that you could make programming safe,
by wrapping things into something, or providing smart iterators.
I believe that programming is fundamentally an unsafe activity,
that is, it requires that you think.*

– Alexander Stepanov

4.1 System Overview

The proposed system aims at automatizing the classification of seven features, as previously mentioned. In the following there is a more detailed description of these features:

1. *Logo size and type.* There are three main Adidas logos, namely, the original “Badge Of Sports” (BOS), the trefoil, and the linear one (as shown in Fig-

ure 4.1); each of them could appear as a small, medium or large logo;



Figure 4.1: Types of Adidas logos.

2. *Three-stripes presence and colors.* A mark that is characteristic of Adidas consists of three equidistant bands of the same color, typically on the sleeves of shirts or on the side of pants. The three-stripes can be contrasted or tonal: this refers to the difference between their colors and the background;
3. *Three main colors palette.* For each article, a maximum of three visible colors should be extracted, ordered by their importance, and the percentage of the area covered by such colors should be provided;
4. *Prints and clothing patterns.* Besides colors, there could be prints over the clothing, textured fabric, or geometric patterns: this is called the main-execution of the article (examples are shown in Figure 4.2);



Figure 4.2: Prints and clothing patterns examples.

5. *Neck shape.* The type of the collar of a jacket, a sweatshirt or a shirt can be classified in major categories, like U-neck, V-neck, hooded, bomber, and so on (illustrated in Figure 4.3);



Figure 4.3: Types of neck shapes.

6. *Sleeves shape.* Adidas teams need to differentiate between “raglan” and “set-in” sleeves, i.e., the shape of the junction between sleeves and body of the garment, as in Figure 4.4;



Figure 4.4: In “raglan” shirts, the junction between the body and the sleeves starts from the armpits and goes to the neck; the term “set-in”, instead, refers to those shirts that have the sleeves sewn with the body of the garment with a vertical closure.

7. *Material of the clothing.* Clothes have a lot of fabric types, but at this stage (recognition from final renderings), only cotton, (generic) synthetic, and viscose fabrics have to be distinguished.

Some preliminary considerations help at understanding the main problems that have to be faced in the development of the system. Thus, when choosing the right method for each feature, the following points have been carefully taken in account:

- The complete dataset that Adidas teams provide us with consists of about 11k images, where articles could be duplicated (e.g., for many of them, clothes back-view and front-view are depicted separately): of these, only 2616 images were already classified and could be used as ground truth;
- Classifications are sometimes difficult to perform even by a human observer, since they depend on very small details, or not well-defined classes: e.g., recognizing the material (synthetic, cotton, or viscose) of a garment only from its rendering image is very hard;
- For some features, classes range in a high variety of values, as for color names, and examples for each class are very few; moreover, having many classes could mean that inter-class differences are small, and this may lead to inaccurate results.

For these reasons, a different module was developed for each feature. Features (1)-(3) have been approached with classic image processing methods, while the others have been approached with deep learning. As an exception, the logo search was also reinforced by using a deep network for certain types of logos. In Figure 4.5, a detailed schema illustrates the structure of the system. Note that the list of output classes in Figure 4.5 is not an exhaustive one, since in all cases the search could either fail, returning a NONE output (e.g., no logos have been found in the image), or may not make sense, returning a NOT APPLICABLE output (e.g., no neck shape has to be found in case the image depicts pants). Also, implementation details are outlined in the following, but they are not shown in the diagram, for the sake of clarity.

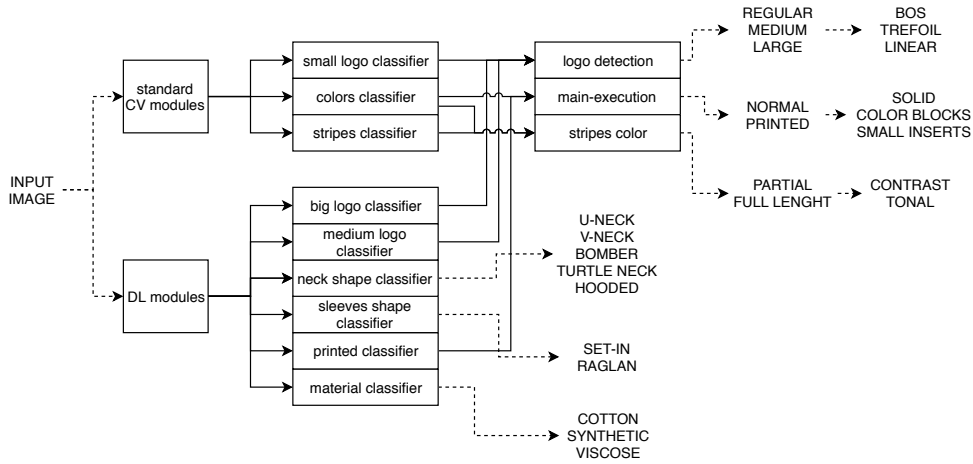


Figure 4.5: Detailed organization of system modules. Uppercase statements highlight inputs and outputs, while boxed ones emphasize the actual modules of the system. Solid arrows denote conceptual connections among modules, while dashed arrows denote inputs or outputs.

Input images are colored 3D-renderings or pictures of the final products. The provided dataset consists of about 3000 images depicting T-shirts, sweatshirts, short and long pants, and jackets. The diversity of such images is very high, meaning that the search for a feature may or may not make sense, depending on the input image given.

Nevertheless, images are comparable in size, although they do not always have the same dimension (small products are stretched and big ones are shrunk). As another common point, the front view or back view of the clothing to be analyzed is the only picture in the image, with no noise in the background. Colors are well visible (brightness is uniform) and as realistic as a photo. An initial step of preprocessing is needed, for each image. Such a little prior step is necessary for each feature.

The preprocessing phase consists of two main operations: the extraction of the shape of the garment in the image, separating it from the background, and the size normalization of the article. Only when needed, the clothing depicted in the image must be preliminary classified as a shirt or a sweatshirt, a jacket, short or long pants, bra or tank tops, and so on. Well-known image processing algorithms are used in this preprocessing phase, as well as some low-level heuristics. The shape of the article is treated as a binary image to be passed as additional informative content in the following steps of the elaboration. Its extraction is an easy task, given the absence of noisy backgrounds. Size normalization is based on the waist size of the product. This normalization aims, in fact, at producing a homogeneous dataset for future computation. The objective is achieved by detecting the waist in various types of clothing—usually placed in the middle of the image—and rescaling the image according to a given waist size to be matched. The article types classification is made by training a deep neural network, using a supervised approach, where images are labeled as shirts, pants, tops, and sweatshirts. Then, the input image enters into an “executor” module, which only purpose is the launch of one, some, or all feature extractors, as requested by the user. Feature extractors are divided into standard Computer Vision (CV) modules, and Deep Learning (DL) modules.

Small, medium, and big logo classifiers contribute to the logo detection task: they output the type of the—small, medium, or large—logo, or `NONE` if the logo is not found. Types are `BOS` for the Adidas original logos, `TREFOIL` for the trefoil logo, and `LINEAR` if there is only the word “adidas” in place of the logo. The logo detection has a top-down logic: once the big logo classifier ends its execution, if there are no large logos, the medium logo classifier runs, and if there are no middle-sized logos, the final step is the small logo search. This is because in case of multiple logos,

the priority should be given to the largest one, for business logics. Also, this allows better modularity, a key aspect in any real world application. Moreover, experiments of training a single network on multiple tasks have shown similar qualitative results. Printed and colors classifiers are put together to evaluate the main-execution of the clothing. The printed one is a binary classifier that detects prints anywhere in the image, while the color classifier detects the three main colors, obtaining a percentage of the area covered by each color. This fact allow us to divide the main-execution in three sub-classes, namely, `SOLID`, if the primary color covers equals or more than the 90% of the area of the clothing, `COLOR BLOCKS` if the primary ranges from 75% to 90% of the area, and `SMALL INSERTS` if the primary is less than 75% of the area. Three-stripes are detected using the stripes classifier module, which outputs the presence or not of stripes in the garment, as well as the length of them. Also the stripes classifier takes advantage of the use of the color classifier, but in terms of color distance: it detects if stripes are contrasted or tonal w.r.t. the background. Neck and sleeves classifiers are only used for non-pants articles, and they output the classes shown in the diagram, or the `NOT APPLICABLE` one. Finally, the material classifier detects the fabric type, as shown in the diagram.

4.2 Logo Detection

Logo detection is the most important task for Adidas data experts, because the logo is the most distinguishing mark on branded clothing, and, in fact, the great majority of articles produced have a well-visible logo. Recognizing its presence and, especially, its type and size, is a key problem, also for developing marketing campaigns and business logics. Logos are very diverse in size, shape, and position: they vary from the original Adidas logo, to the trefoil or the linear one, and also some related marks, like “Terran” or “Tango” logos can be present; logos could be very small, in a corner of the sweatshirt/shirt/pants, or they could cover the whole garment in form of a large print; moreover, medium sized logos have to be recognized as different from the ones mentioned above, keeping in mind that the real dimension of a medium logo stays in a square box of side $\sim 10cm$, while a small one in $\sim 6cm$. As briefly explained

in the previous section, three logo classifiers contribute to the task of logo detection. Methodologies and techniques are detailed below, describing the computer vision approach and the deep learning one separately.

4.2.1 Logo detection through deep learning techniques

The search for a light and general solution to the logo detection led us to the use of deep learning techniques, such as deep neural networks. In particular, CNNs are well-known for their properties that make them especially appropriate for object detection tasks. As a matter of fact, the characteristic convolutional layer of CNNs uses the convolution operation between the input and a kernel, resulting in sliding such kernels along the entire input and performing the dot product between each input units and a small set of weights. That is, the CNN learns which kernel to use to match the input. Moreover, CNNs reason in terms of matrices and volumes, which made them suitable for input like images, or input with grid-like topologies. Nevertheless, there is an important drawback: training a CNN, as for most of the deep neural networks, requires a very high number of samples, and high computational resources. For this reason, the *fine-tuning* technique confers many benefits to this approach.

As far as medium and large sized logos detection concerns, a deep learning approach with fine-tuning was used. The main idea of fine-tuning is to train a deep neural network that has already been trained on a certain dataset, which could be or not could be relevant in the application domain, e.g., the dataset for pre-training could contain images that are very distant from the fashion context. In this work, the CNN used is the *VGG19* one, and some experiments were made also with the *VGG16*, as detailed in Section 4.4. The first step of the fine-tuning process on VGG19 consists in the training of the network on the ImageNet competition dataset, though its evaluation classes have nothing in common with apparel, at a first look. Nevertheless, this first training provides the neural network with the ability to discriminate some basic features of images, that do not belongs to any specific context, namely, lines, simple geometric shapes, and color stains. In this context, the term “features” refers to the mid-outputs of a CNN, obtained after the passage of the image in a single convolutional layer, and not to the list of seven features we described previously. Since in this

section we approach only the logo detection, the term “feature” is not ambiguous. The second step of fine-tuning consists in freezing the weights of a fixed number of initial layers of the network. The reason behind this choice is that simple features, such as edges, lines, and so on, are generally attributed to the very first layers of a neural network and they are useful in almost all image classification and object detection tasks. Hence a single training phase on the ImageNet dataset is sufficient to obtain acceptable weights for some of the first layers. The third phase, after having fixed the parameters of the first layers, is a second training on a barely different network topology. In this second training cycle, the last layer of the pre-trained VGG19 is removed and some *fully-connected* layers, loaded with random weights, are added in its place. This last layer is the one responsible for the classification itself, i.e. it outputs the ImageNet classes, which is certainly inappropriate for our scope. These added layers, together with the last ones of the base model, were the ones responsible to recognize more specific and elaborated features inside the input image, namely, logos themselves or part of them. The fact that layers increased the complexity of recognized features as the depth of the network increases was helpful in the development phase. As a matter of fact, this step allows us to extract some evaluation parameters for each layer, and to represent them by means of *heatmaps*, i.e. images that show which parts of the input image have been triggered at each layer. Figure 4.6 shows the heatmap obtained by the neuron activation values of the Global Average Pooling (GAP) layer, which is placed after the last convolutional layer of the network. The figure shows the original image, the overlap between the image and the heatmap, and the extracted heatmap.



Figure 4.6: Heatmap of a nearly invisible medium logo.

Another technique that allows to workaround the lack of images in a dataset, and that brings significant improvements in system accuracy at the same time, is the *data-augmentation*. Data-augmentation consists in applying some transformations to the training set images, before inputting them into the net. These transformations vary from rotations, stretches, zooms, splits, to channel shifts (in order to reach color variations), and noise addition. The benefit of using such a technique is that it provides the net with much more training information. Although it is a sort of dummy information, it achieves the intended purpose. One last addition which leads to a slight increase in the overall accuracy percentage is the parameter called *class-weight*. The purpose of such a parameter is to apply an imbalance in the classification propensity of the net. Class-weight is applied because, by careful observation of the composition of Adidas AGTM datasets, it was discovered that clothes with the logo vastly outnumbered the ones without it. As a result, class-weight application has made the net more incline at attributing a piece of apparel to the logo class (MEDIUM or LARGE) than to the no-logo one (the class NONE).

4.2.2 Logo detection through standard computer vision approaches

The method used for detecting small logos is a template matching method. This method was previously described by the author in [28], but it is reported here for clarity. The PCC [46] is used, with personalized kernels. Such personalized kernels are actually small images called templates. The approach of PCC has to be improved to search for such templates into the input images, instead of low-level kernels. The



Figure 4.7: Examples of occlusion (a), rotation (b), deformation (c), three typical issues that have to be taken into account for correct logo detection. The ground truth dataset comprises only some examples of these difficult cases. Still, more examples can be generated via data augmentation.

template of a logo is a grayscale image which represent the logo itself. One of the main problems is that logos in images are not always completely visible, or clear. Logos could be deformed by folds, or printed in a small corner of the garment. For these reasons, a plain image representing a logo is not sufficient for matching most of the logo occurrence in input images. Hence, templates are subject to rescaling, angular sliding, and rotations, prior to the PCC computation. In case of small logos, which view can be easily occluded by other details in the apparel image, the left-hand side (LHS) and the right-hand side (RHS) of the logo are treated as two different templates. Each transformation of the template has to be matched with the input image. Such an approach has well-known limitations, which are outlined as follows:

- In input images, there is a wide variety of occlusions or deformations of the logo, e.g., rotations, angular sliding, and so on (see Figure 4.7): this means a large number of required transformations of the template;
- PCC is computed on each pixel of the input image, by sliding each template transformation on the whole image;
- false positive object detections are easy to obtain, since logos are composed mostly of geometric patterns, used also for other prints.

To approach the first two points, a downsampling is performed on the input image. Then, the search area for computing the PCC is reduced to regions where logos are more likely to be found. There are also some drawbacks, e.g., if the downsampling reduces too much the image, some important information about the logo could be lost, and this could result in a decreasing of the method accuracy. Moreover, regions where a logo is expected to be found have to be chosen correctly to achieve good results using this algorithm. On the other hand, appropriate choices of the available regions and the sampling rate drastically reduce the computational time, leading to good results on the majority of the input images. As stated in the third point, false positives are easily obtainable, and they cannot be avoided using only the previously described downsampling and reduction. In our approach, we use a mask, which we apply to the given template. Such a mask puts weights, as a degree of importance,

on each pixel of the template. In this specific application, a three-valued scale for masking a template is used, to differentiate three zones of the logo image: highly-relevant zones (which characterize well the logo—e.g. edges), common zones (which appear in each logo but also in other parts of the image—e.g. geometric patterns), and irrelevant zones (such as the background).

Precisely, the template is a grayscale image of the logo, which is usually much smaller than the input image. A mask is a three-valued grayscale image of the same size of the template. Let I be the two-dimensional matrix representing a grayscale version of the input image, and T the same for the template, and let n be the size of the template. M is a matrix of order n , representing a grayscale image used as a mask, which values lie between 0 and 1. The punctual equation of PCC, applied to images, is as follows:

$$pcc(I, T, x, y) = \frac{\sum_{j,k} (I_{xy}(j, k) - \mu_{I_{xy}}) (T(j, k) - \mu_T)}{\sqrt{\sum_{j,k} (I_{xy}(j, k) - \mu_{I_{xy}})^2 \sum_{j,k} (T(j, k) - \mu_T)^2}} \quad (4.1)$$

where j spans from $-T_w/2$ to $T_w/2$, and k from $-T_h/2$ to $T_h/2$, given T_w and T_h as the width and the height of the template T , respectively. I_{xy} is a portion of the image I with the same size of T and centered around $p = (x, y)$. Scalars $\mu_{I_{xy}}$ and μ_T are the average values of I_{xy} and T , respectively. $T(j, k)$ (and, therefore, $I_{xy}(j, k)$) is the pixel value of that image at the coordinates j, k computed *from the center* of that image. It is worth noting that the numerator is the covariance between I and T , and the denominator is the product of their standard deviations. Hence, the equation is invariant to affine transformations. This means that PCC is a robust method in case of varying light conditions, showing independence from several real-world lighting issues, since the main lighting contribution from objects is linear. Rearranging the Eq. (4.1) allows us to gain a simplified equation that does not depend directly on the averages of the image and the template. This is useful to apply a mask M to the

template, as follows:

$$pcc(I, T, M, x, y) = \frac{n \sum_{j,k} I_{xy} MT - \sum_{j,k} I_{xy} \sum_{j,k} MT}{\sqrt{n \sum_{j,k} I_{xy} - (\sum_{j,k} I_{xy})^2} \sqrt{n \sum_{j,k} MT - (\sum_{j,k} MT)^2}} \quad (4.2)$$

where indexes j, k are omitted for the sake of brevity. The value of PCC obtained in this way is still invariant to affine transformations.

The relevant region of the input image consists initially in the entire visible area of the garment in the image, obtained in the preprocessing phase: in fact, the binary image that represent the shape of the apparel product has value 0 on the background, and 1 otherwise. The initial template matching is performed on a downsampled grayscale version of the input image. The Eq. (4.2) is computed for each pixel of coordinates (x, y) in the search region, for each transformation of the template. It is worth noting that $-1 \leq PCC \leq 1$. Such a score has an immediate interpretation: a value close to 1 means a strong correlation between the portion of the input image and the template, whilst a value close to -1 an inverse correlation. Since logos could have different backgrounds and colors, both correlation and inverse correlation denote a good match. In our approach, the PCC value is then transformed into a probability, between 0 and 1. A threshold is applied to find the most probable matching. Applying a threshold returns a set of regions, and these regions are used in the following step of the algorithm to perform the successive matching. Simultaneously, the downsampling on the input image is reduced. Such an iterative process is repeated no more than three times in our system, to avoid high computational times. At the end of the iterative process, the remaining search regions become the regions where the logo was found, and a circle is drawn over it to indicate its position. Figure 4.8 shows the procedure for finding a small logo using PCC. Only the matching with the right-hand side of the logo is detailed, but the last image shows the results of both searches (LHS and RHS). In this case, the logo is clearly visible, but it could be confused with the background pattern: in fact, the first iteration of PCC in Figure 4.8c retrieves several false positives.

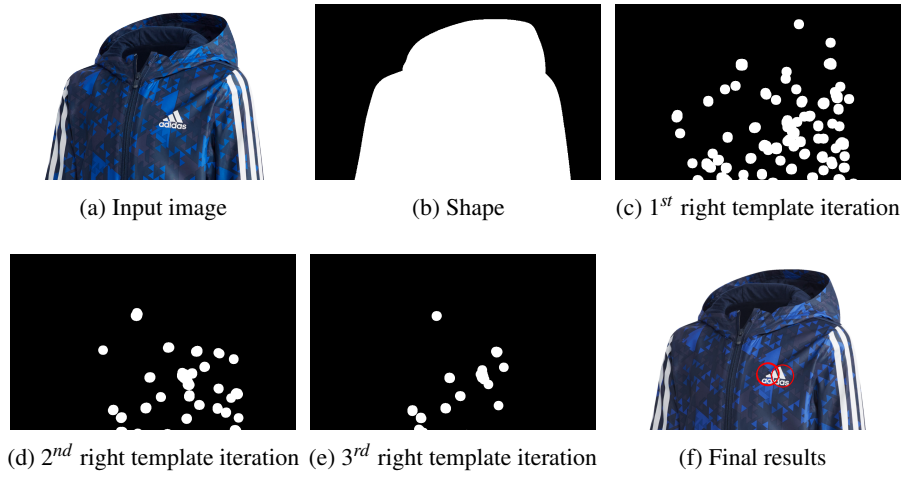


Figure 4.8: An example of logo detection using template matching.

This approach is also applicable to other kinds of logos if other types of templates and masks are provided, giving very accurate results if appropriate template transformations are performed. Nevertheless, such a method is hardly generalizable to other datasets and an increasing number of iterations is required to generalize without losing accuracy, but predictably the computational cost of the algorithm increases too. A purely qualitative and visual analysis seems to confirm that the sum of all deep learning techniques and workarounds grants more appreciable results as the logo size increases, particularly with deformed or partially visible logos. On the other hand, it shows less sensitivity than PCC in detecting tonal logos (e.g., gray logos on a slightly lighter gray background)—those that are so tonal that they are barely recognizable even by human eyes. A more accurate evaluation of the results is presented Section 4.4.

4.3 Other Features

Beside logos classification, methods for recognizing and classifying the other features are described in the following.

4.3.1 Stripes recognition

Apart from logos, another notable feature to extract is the presence, location, and properties of the peculiar Adidas three-strips. A description of brand relevant marks recognition is discussed in [28], where this approach was detailed first. The three-strips detection needs specific geometric techniques, since their recognition is based on properties of parallelism and perpendicularity of strokes that compose the design. These properties, such as parallelism, perpendicularity, intersections between lines, could be evaluated by reasoning on the edges of the input image. Hence, the input image is transformed in a line image. The chosen methodology is the use of thinning algorithms and edge detection to perform this task.

The first step, which goal is to obtain a rough approximation of the input image edges, is the computing of a gradient map. An opportune gradient map must be obtained, in order to preserve significant information about the clothing design. Clearly, colored realistic 3D-renderings or photographs do not have well-definite edges: instead, borders are recognized by discontinuity between a color and another. For this reason, a distance metric between two adjacent colors, that define different regions of the depicted clothing, should be defined. The first option would be to use the Euclidean distance of the RGB colors, but a better option consist of using a more accurate distance (such as *CIEDE2000* [64]).

As expected, input images are mapped in the RGB color space. To perform the *CIEDE2000* color distance, images must be mapped into the CIELAB color space, instead. Such a color space is also 3-dimensional, and it is capable of denoting all visible color by a human eye. It consists of two channels and a lightness value. Two channels range from red color to green (A channel), and from yellow to blue (B channel). Thus, each pixel in an image is coded by using three parameters: L, A, and B. A peculiar metric, *CIEDE2000* distance ΔE_{00} , is defined on CIELAB space by [64].

The distance ΔE_{00} is negligible between colors which are perceived as the same by our eyes, but it takes high values when these colors have “just noticeable differences”. Such a property grants that small color differences are measured precisely.

Using this ΔE_{00} distance we compute the adjacent difference of each pixel to its neighbor on the left and its neighbor below, obtaining two different real valued images: d_x and d_y , respectively horizontal gradients and vertical gradients (defined for each pixel of the input image). Finally, with the formula $g = \sqrt{d_x^2 + d_y^2}$, we compute the global gradient map, defined for each pixel (a first approximation of the input image borders). Thresholding this gradient map with an adaptive Gaussian threshold gives a binary output image representing the edges.

The obtained thresholded image is now a sort of sketch, a border image, which has to be thinned. After thinning, a pruning phase follows, to obtain all paths that compose the borders. Paths are useful to reason in geometric terms, and to match some specified patterns. In our case, the pattern to be retrieved consists of three parallel bands of the same width, i.e., six equidistant and parallel lines, in terms of paths. As for logos, the visibility of the three-stripes could be variable, and they are in the majority of cases partially hidden or deformed by the garment folds. Those visibility issues could affect their equidistance and parallelism properties. The three-stripes recognition is performed in two different steps. First, the image is segmented in order to obtain a set of significant regions where stripes are more likely to be found. Second, the algorithm performs the actual search in such significant regions, using the geometric properties of stripes.

Thanks to the preprocessing phase, a preliminary recognition of the type of the garment is performed. Then, relevant regions are extracted depending on such a type: in pants, stripes are searched on the side of the legs, while for shirts and sweatshirt, the important regions correspond to sleeves. Finding relevant regions is of utmost importance in avoiding excessive computational times in the next step. The second step, in fact, analyze all paths in each relevant region. For each point of a path, the algorithm computes its neighborhood. The direction of the line is obtained by the orientation of its associate path. Knowing the direction, the algorithm goes across each point of the path, focusing only on points of the neighborhood that are perpendicular

to the direction. A parallel path is recognized if and only if for each analyzed point in a certain range there is another point in the perpendicular direction and on the same side. Given a fixed distance, if a minimum of three and a maximum of seven parallel lines are detected in that range, the algorithm marks these lines as “three-stripes”. Such a heuristic is deliberately a rough estimate of the real geometric shape of the stripes, in order to prevent false negatives caused by occlusions, folds, deformations and so on.

4.3.2 Colors segmentation

The color, or colors, of an article is another important feature to be extracted. It is a feature by itself, as explained in Section 4.1, but it is also used in determining the contrast between stripes and the background, and the composition of clothing prints and patterns. A brief overview of this approach was previously illustrated in [28]. Fashion industries develop new color names before each season, and the colors of articles are identified by such color names. As a result, the color palette of a product is composed according to that classification, which consists of thousands of color names, collected over the years. An appropriate mapping between each color name and its color code is required to correctly perform the segmentation. In order to provide data experts with meaningful statistical information from colors classification, defining a basic color palette of about 20 or 30 color names is advisable. In [65] is proposed another approach, which tries to learn the names of colors directly from the input images. In this work, we study a different solution, i.e., the creation of a small set of color names which are not related with those of Adidas, but that groups color shades and nuances in only 26 categories. This choice means a less accurate colors segmentation, but it has the advantage of being a more generalizable approach, and of preparing the dataset for successive statistical inferences.

As for stripes, an appropriate color space must be used, with its associated color distance. The color names are then mapped into that color space. A property of photograph or realistic renderings, as the images in Adidas dataset, is that their colors are not in solid blocks, but the regions that our eyes perceive as containing the same color are full of shades and nuances, that may invalidate the result of the segmen-

tation algorithm. A distance that measures such shades as parts of the same color is then necessary. An appropriate choice, as discussed for three-stripes recognition, is the CIELAB color space, together with its color distance *CIEDE2000*. The search of three most important colors in the article is performed in three phases. Region growing [66] is used in the first phase to perform an initial rough segmentation. Such a segmentation is operated on RGB input images. In order to measure the *CIEDE2000* distance among these regions, a conversion transforms the input image into a LAB image. The distance is computed for regions two by two. For each region, the average color value is extracted, and the distance is measured between these values. Similar colors are thus grouped together, according to the CIELAB standard. The output of this second phase of the algorithm is a set of larger regions which better approximate the color area than the RGB regions. In the third and last phase, the distance between the average LAB value of each region and each color name code is computed. Hence, each region takes the name of the nearest color name. The algorithm returns the names of the three larger regions, and the percentage of covered area by such regions.

4.3.3 Deep learning based feature extraction

The extraction of prints, neck and sleeves' shape is approached through deep learning techniques. In these cases, the availability of labeled images is sufficient to adopt this approach—although there are few in number, as in the logo case—and the separation among classes is clear enough. In particular, for prints and clothing patterns recognition, a fine-tuned VGG19 model is used, as for logo classification. Details are explained in Section 4.2.1.

During the training process, depending on which feature to detect, numerous adjustments are applied, varying from the number of fully-connected layers that have been added to the starting VGG19 model, to the number of neurons for each of those, to the editing of datasets composition, or tuning parameters (e.g., the learning rate). As a matter of fact, the training dataset must be varied and consistent, and as less redundant as possible. In Section 4.4, some accuracy tables are reported, containing examples of attempts that were done in order to reach the highest possible accuracy,

implemented in the final modules.

4.4 Experimental Results

Modules of the above described system were tested separately, and experiments gain useful insights for improving the accuracy of each module. Also, the detailed feed-backs that Adidas teams provided us were taken in account in the enhancement of the feature extraction system. In this section, some of the experiments performed with CNNs are reported, using the accuracy metric, to give a sense to the main decisions behind the deep learning approach we used. Then, the results of the standard computer vision modules are detailed, in terms of confusion matrices and accuracy metrics.

4.4.1 DL modules

The very first experiment was the evaluation of a simple CNN architecture trained from scratch. The chosen architecture is a standard one where convolutional layers are followed by dense ones, as in a LeNet [67]. In Table 4.1, accuracy results are reported, varying the number of dense and convolutional layers, as well as the learning rate. A *dropout* is applied at each training epoch, to prevent overfitting. Accuracy is always measured on a validation set, consisting of different images from those in the training set. The table reported is related to the classification of large logos. Similar results were obtained for the other features.

Table 4.1: Training of a simple CNN from scratch.

# of conv. layers	# of dense layers	Learning rate	Dropout	Accuracy
4	3	0.001	0.5	0.731
4	4	0.001	0.5	0.733
5	4	0.01	0.5	0.742
5	5	0.01	0.5	0.738

Experiments with CNNs trained from scratch were intentionally left simple, since training a large deep network requires a huge amount of images and high computational time. Unfortunately, an accuracy value of about 74% is not good enough for our scope.

The fine-tuning process brought significant improvements in accuracy percentage in comparison with deep learning approaches from scratch. The main reason behind this major enhancement is the use of the pre-trained network ability to recognize low-level features, reducing the impact of the lack of images in the dataset. Although the very first attempts were accomplished adopting a VGG16 network topology, it turned out that VGG19 was eventually a more efficient starting model. In Tables 4.2 and 4.3, results of the fine-tuning with VGG16 and VGG19 architectures are shown, varying the number of frozen layers and the number of the fully-connected layers added at the end of the network.

Table 4.2: Some tests on fine-tuning with VGG16 topology and relative accuracy values.

# of added layers	# of frozen layers	# of first layer neurons	Accuracy
1	10	256	0.853
1	11	512	0.846
2	10	128	0.850
2	11	512	0.846
3	12	256	0.869
3	12	512	0.848

Table 4.3: Some tests on fine-tuning with VGG19 topology and relative accuracy values.

# of added layers	# of frozen layers	# of first layer neurons	Accuracy
2	11	256	0.889
2	12	512	0.890
3	12	512	0.893
3	13	1024	0.908
3	14	512	0.902

In Table 4.4, a further investigation on VGG19 fine-tuning was made, to choose the best value of the learning rate (i.e., the one that achieves better results in terms of accuracy, in an acceptable number of epochs).

Table 4.4: Learning rate parameter variations.

Learning rate	Accuracy	# of epochs
0.01	0.824	14
0.001	0.947	8
0.0001	0.68	19

An example result for the deep learning approach is shown in Table 4.5, for large logo types. The network recognize correctly **93%** of the images in a dataset of 86 unseen images, chosen by Adidas data experts (how the 86 number come out is explained later, in Section 4.4.2). The classification of these images is acceptable, with only a little misclassification of linear and original (BOS) logos. Table 4.5 shows the confusion matrix, with prediction classes on the first column and the reference on the first row. The number of images per class and their frequency percentages are also indicated.

Table 4.5: Results on large logo detection on an unseen validation dataset, performed by Adidas team.

Prediction / Reference	BOS	TREFOIL	LINEAR	Frequency %
BOS	46	0	0	53.5
TREFOIL	0	24	2	32.6
LINEAR	0	4	10	13.9

As another example, the very different problem of prints recognition is analyzed. The validation dataset contains 250 images, and the overall accuracy of the printed classifier is **94.4%**. The confusion matrix, as well as class frequencies are reported in Table 4.6.

Table 4.6: Results on prints classification on an unseen validation dataset, performed by Adidas team.

Prediction / Reference	PRINTED	NORMAL	Frequency %
PRINTED	213	7	88.0
NORMAL	7	23	12.0

Finally, a legitimate question would be the reason why a VGG19 should be used, stating that many other architectures are available, and some of these architectures are also better rated than VGG19 on surveys like [68]. An experiment on middle sized logos was made in order to determine the most appropriate network topology in our context. Results are shown in Table 4.7.

Table 4.7: Results on medium logo detection, using different architectures.

Network	# of frozen layers	total # of layers	Validation accuracy
VGG19	15	29	0.97
DenseNet	369	714	0.90
InceptionV3	164	318	0.95
NASNetLarge	401	776	0.89
ResNet50	94	182	0.54

4.4.2 CV modules

In logo detection, the type is not the only class to be predicted. The system searches first for large logos, and if nothing is found, the medium and the small logo searches are performed in the same cascade logic. Adidas data experts test this logic on a dataset of 290 images, and then they analyze separately the results of each classifier on logo types. In Table 4.8, predictions are compared to references in the confusion matrix, and the frequencies for each class is indicated.

Table 4.8: Results on logo size detection on an unseen validation dataset, performed by Adidas team. Overall accuracy is **96.6%** on 290 images.

Prediction / Reference	REGULAR	MEDIUM	LARGE	Frequency %
REGULAR	194	4	2	67.9
MEDIUM	0	4	0	3.1
LARGE	3	1	82	29.0

The type classification of the 86 over 95 images that depict large logos was shown in the previous section in Table 4.5. Performance of the template matching approach gains **92.0%** overall accuracy on small logos on the 200 images that were classified as REGULAR. The confusion matrix is shown in Table 4.9. It is worth noting that there are no small linear logos. In this particular dataset, there are none, but normally these logos are not recognized by our template matching algorithm. This is because a linear

logo template must depict a sub-part of both BOS and trefoil logos (the “adidas” writing on the bottom of the logo, as shown in Figure 4.1), and this fact may generate a huge number of false positives.

Table 4.9: Results on small logo detection on an unseen validation dataset, performed by Adidas team.

Prediction / Reference	BOS	TREFOIL	Frequency %
BOS	168	1	91.5
TREFOIL	15	16	8.5

Regarding the three-stripes module, results were obtained by comparing the output of the classification with a ground truth of 1375 images. Such a dataset contains also 8% of images that does not belong to neither `FULL LENGTH` nor `PARTIAL` classes. The overall accuracy of stripes length is **80%** on that dataset.

Colors, instead, do not have a precise accuracy value, since both the color names and the coverage percentage are subjective attributes. On the contrary, the color module in combination with the prints and three-stripes classifiers gains measurable results. In the first case, the color module has to recognize the type of “composition”, choosing among `SOLID`, `COLOR BLOCKS`, and `SMALL INSERT` classes, as outlined in Section 4.1 and in Figure 4.5. On the 213 not-printed products recognized using the prints classifier, as in Table 4.6, the overall accuracy is **70.4%**. The confusion matrix and class frequencies are shown in Table 4.10.

Table 4.10: Results on not-printed color combination on an unseen validation dataset, performed by Adidas team.

Prediction / Reference	SOLID	COLOR BLOCKS	SMALL INSERTS	Frequency %
SOLID	115	0	0	77.9
COLOR BLOCKS	8	20	2	14.0
SMALL INSERTS	43	10	15	8.0

The most frequent misclassification, in this case, regards solid color garments. The reason is the presence of small regions, such as logos, of a different color. Better results are achieved in the classification of tonal or contrasted stripes. As a matter of fact, the distance between the colors of the stripes and the background is used at this scope, and gains **83%** accuracy on the 1375 images dataset.

Chapter 5

Conclusions

*I tipi grossi come te mi piacciono,
perché quando cascano,
fanno tanto rumore.*

– Eli Wallach

In Chapter 3 we proposed a sketch vectorization system that proved its correctness and viability at treating different input formats for complex hand-drawn sketches. It has been tested with real fashion sketches, artificial generated pictures with added noise, as well as random subject sketches obtained from the web. The line extraction algorithm outperforms the state of the art in recall, without sacrificing precision. It is also faster, more accurate and easier to adapt to one's needs due to the modular nature of the system. The unbiased thinning helps in representing shapes more accurately. It has proven to be better than the existing state-of-the-art approaches in retaining detail, in particular with difficult, sharp shapes. Still, it exhibits the same speed and robustness of a classical thinning strategy. The discussed path extraction algorithm provides a complete treatment of the conversion of thinned images into a more suitable, compact representation. Finally, Schneider's algorithm for vectorization has been improved in the quality of its results. Experiments show a noticeable reduction in the number of generated control points (by a 10-30% ra-

tio), while keeping good runtime performance. In conclusion, the current version of the proposed framework has been made available as an Adobe Illustrator plugin to several designers at Adidas exhibiting excellent results. This further demonstrates its usefulness in real challenging scenarios.

In Chapter 4 main motivations, techniques, methods, and final results of a feature extraction software system were presented. The application domain is the fashion creative workflow of apparel industries, and in particular the visual analysis of the large amount of data obtained as outcomes of designer works, prior to the actual fabrication of the garment. The application was conceptualized and developed in synergy with Adidas AGTM data expert teams, whose highlight main directions and most important features. The study had pointed out the diversity of features to be extracted, and the emerging of some challenging research problems. Above all, the lack of datasets and an insufficient literature on fashion domain led to the implementation of some state-of-the-art techniques in a personalized approach, until a robust and reliable system had been proposed. The novelty of this work resides mainly in the domain of application, although original methods were used in every system modules. Extensive tests and experiments were performed both by the author and by Adidas teams. As a matter of fact, the system was refined at experimental time, showing finally good accuracy results in all its tasks. Right now, the system is used by Adidas worldwide to classify and extract features from the last ten years of production. More than 400.000 product images have been analyzed, and the system has been massively distributed on AWS servers to enhance its throughput. The system performance has deemed to be comparable with the tests performed in Section 4.4 (tests performed with just 2.000 images). These images mainly include 3d product renderings but also include challenging real world pictures. An interesting future development could be the introduction of more cases/classes (e.g., other types of logos, or other types of clothes, such as bra or tank tops), thus refining the search to a wider variety of products.

5.1 Future works

Machine learning, besides being a powerful analysis tool, can be a source of inspiration for designers and creatives. In this short section we will discuss some preliminary works of the author on the subject of Image to Image Transformation and Image Generation applied to creativity.

5.1.1 Shoe colorization

The first proposal is image colorization applied to raw sketches. As input to our system we present a paper sketch of a shoe. As output the network will generate a complete colored rendering of the shoe. This is very useful both as a tool of inspiration for a designer, as well as a guide to a non-expert user in designing his or her own shoe. The network will learn specific recurring patterns from a history of training images, and apply them to the output rendering. This way, even a non-expert is guided in avoiding common “errors” and applying recurring color-part pairings. An expert user can benefit from the system for quick prototyping, after the creation of a sketch, to obtain a simple, yet realistic and complete, colorization of such sketch.

This task has been performed, and learned, by an encoder-decoder network (UNet [69]), without the need of a Discriminator (popular alongside GANs [70]). Moreover, unlike Cycle GANs [71], the image transformation is learned as a paired image transformation, i.e. each input image has a 1:1 correspondence to a single output image (pixel by pixel). The training of such system has been performed with a dataset of 20.000 high-resolution shoe renderings courtesy of Adidas AGTM, see Figure 5.1 for an example.

The interesting part is what we used as an input sketch for the training, due that such database does not have a ground truth dataset of the original sketches. To generate the input sketches to be learned by the network we performed a sort of Self-Supervised learning, in the sense that we “recreated” the input sketch of a shoe by performing an edge-detector on the rendered picture. In Figure 5.2 you see an example.

In fact the network, during the training phase, is learning somewhat the inverse



Figure 5.1: A shoe from the Adidas AGTM high-resolution shoe dataset (1200x600 pixels, 20.000 images).

of an edge-detection operator. To further guide the network in color generation (and to allow a user to provide inputs at inference time) we drew some little colored lines over the sketch using the original shoe rendering color. This way the network will learn to complete the sketch coloring given some hints or preferences of the user. If the network is integrated in a drawing tool, the user will be able to see real-time coloring due to some input color strokes.

Figure 5.3 shows some preliminary results of the inference performed by the network. It can be seen that the network follows the input hints closely, while also providing unseen color suggestions as a guidance to the user.

5.1.2 Attribute modification

As hinted in the introduction, a useful task for the fashion industry would be applying a different style or pattern to an existing garment to produce a variation of an old model.

More generally there is a need of image to image translation due to some style constraints. The author, alongside some colleagues, is working on a proposal of Few Shots image to image translation using GANs and Meta Learning. While the subject



Figure 5.2: A shoe from the Adidas AGTM dataset and its correspondent sketch, generated with edge-detection. Also note the color hints in the right, bottom parts of the sketch. Those will guide the network in reconstructing the color of each part.



Figure 5.3: Some input output examples of transformations performed by our U-net. The input has not been previously seen by the network (e.g. it is a validation sample).

of image to image has already been studied from papers like StarGAN [72] and independently few shot image classification has been studied in [73], to our knowledge our work is the first proposal of using Meta Learning for image to image translation.

The first immediate benefit of using Meta Learning is that such an approach does not involve labels and classes, as opposed to methods such as StarGAN that heavily relies on them. Each image to image transformation task is learned by simply adding its relative task to one of the the meta-learning passes; there is no need to modify the network topology.

Moreover this approach seems to be very promising in the context of few-shot learning, i.e. learning from few examples of a new, unseen, class. This can be especially beneficial when applying a new style to an existing apparel product, because we could have few samples of the product from which we want to “copy” the style (this is especially the case when the “inspiration” product comes from a competitor...).

In Figure 5.4 you see some preliminary results of the proposed work. The task domain, in this case, is facial attribute modification, the domain in which StarGAN operates, and to which we compare, but can be easily ported to our creative field. In this preliminary test, the network correctly learns to transform a face, changing an

attribute into another. Reported example transformations are: blond hair, pale skin, eyeglasses, mustache and gray hair.

This work is already available as an Arxiv preprint: [74].



Figure 5.4: Facial attributes generation by our proposed few shot image to image translation method.

Bibliography

- [1] Xavier Hilaire and Karl Tombre. Robust and accurate vectorization of line drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(6):890–904, 2006.
- [2] Edgar Simo-Serra, Satoshi Iizuka, Kazuma Sasaki, and Hiroshi Ishikawa. Learning to Simplify: Fully Convolutional Networks for Rough Sketch Cleanup. *ACM Transactions on Graphics (SIGGRAPH)*, 35(4):121, 2016.
- [3] Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. Fidelity vs. simplicity: a global approach to line drawing vectorization. *ACM Transactions on Graphics (TOG)*, 35(4):120, 2016.
- [4] Edgar Simo-Serra, Satoshi Iizuka, and Hiroshi Ishikawa. Mastering sketching: adversarial augmentation for structured prediction. *ACM Transactions on Graphics (TOG)*, 37(1):11, 2018.
- [5] Bo Li, Yijuan Lu, Afzal Godil, Tobias Schreck, Masaki Aono, Henry Johan, Jose M Saavedra, and Shoki Tashiro. *SHREC’13 track: large scale sketch-based 3D shape retrieval*. 2013.
- [6] Dov Dori and Wenyin Liu. Sparse pixel vectorization: An algorithm and its performance evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(3):202–215, 1999.
- [7] Jiqiang Song, Feng Su, Chiew-Lan Tai, and Shijie Cai. An object-oriented progressive-simplification-based vectorization system for engineering draw-

- ings: model, algorithm, and performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(8):1048–1060, 2002.
- [8] Alexandra Bartolo, Kenneth P Camilleri, Simon G Fabri, Jonathan C Borg, and Philip J Farrugia. Scribbles to vectors: preparation of scribble drawings for cad interpretation. In *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling*, pages 123–130. ACM, 2007.
- [9] Yushi Jing, David Liu, Dmitry Kislyuk, Andrew Zhai, Jiajing Xu, Jeff Donahue, and Sarah Tavel. Visual search at Pinterest. In *Procs. of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1889–1898. ACM, 2015.
- [10] Nikhil R Pal and Sankar K Pal. A review on image segmentation techniques. *Pattern Recognition*, 26(9):1277–1294, 1993.
- [11] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.
- [12] Arnold WM Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, 2000.
- [13] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Procs. of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, volume 2016-January, pages 779–788. IEEE, 2016.
- [14] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

- [15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Procs. of the 28th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015)*, pages 1–9. IEEE, 2015.
- [16] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [17] Tony Lindeberg. Edge detection and ridge detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):117–156, 1998.
- [18] Carsten Steger. An unbiased detector of curvilinear structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(2):113–125, 1998.
- [19] Gioacchino Noris, Alexander Hornung, Robert W Sumner, Maryann Simmons, and Markus Gross. Topology-driven vectorization of clean line drawings. *ACM Transactions on Graphics (TOG)*, 32(1):4, 2013.
- [20] Xueting Liu, Tien-Tsin Wong, and Pheng-Ann Heng. Closure-aware sketch simplification. *ACM Transactions on Graphics (TOG)*, 34(6):168, 2015.
- [21] Mikhail Bessmeltsev and Justin Solomon. Vectorization of line drawings via polyvector fields. *arXiv preprint arXiv:1801.01922*, 2018.
- [22] Jakub Fišer, Paul Asente, Stephen Schiller, and Daniel Šỳkora. Advanced drawing beautification with shipshape. *Computers & Graphics*, 56:46–58, 2016.
- [23] Song-Hai Zhang, Tao Chen, Yi-Fei Zhang, Shi-Min Hu, and Ralph R Martin. Vectorizing cartoon animations. *IEEE Transactions on Visualization and Computer Graphics*, 15(4):618–629, 2009.
- [24] Takeo Igarashi, Satoshi Matsuoka, Sachiko Kawachiya, and Hidehiko Tanaka. Interactive beautification: a technique for rapid geometric design. In *ACM SIGGRAPH 2006 Courses*, page 8. ACM, 2006.

- [25] Gunay Orbay and Levent Burak Kara. Beautification of design sketches using trainable stroke clustering and curve fitting. *IEEE Transactions on Visualization and Computer Graphics*, 17(5):694–708, 2011.
- [26] Henry Kang, Seungyong Lee, and Charles K Chui. Coherent line drawing. In *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, pages 43–50. ACM, 2007.
- [27] Jiazhou Chen, Gael Guennebaud, Pascal Barla, and Xavier Granier. Non-oriented mls gradient fields. In *Computer Graphics Forum*, volume 32, pages 98–109. Wiley Online Library, 2013.
- [28] Luca Donati, Eleonora Iotti, and Andrea Prati. Computer vision for supporting fashion creative processes. In *Recent Advances in Computer Vision*, pages 1–31. Springer, 2019.
- [29] Ali Borji, Ming-Ming Cheng, Huaizu Jiang, and Jia Li. Salient object detection: A survey. *arXiv preprint arXiv:1411.5878*, 2014.
- [30] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Procs. of the 2001 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2001)*, volume 1, pages I511–I518. IEEE, 2001.
- [31] Thomas Blaschke. Object based image analysis for remote sensing. *ISPRS Journal of Photogrammetry and Remote Sensing*, 65(1):2–16, 2010.
- [32] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *Procs. of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2012)*, pages 3354–3361. IEEE, 2012.
- [33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition

- Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
- [34] Paul Viola and Michael J Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.
- [35] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Procs. of the 2005 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005)*, volume 1, pages 886–893. IEEE, 2005.
- [36] David G Lowe. Object recognition from local scale-invariant features. In *Procs. of the 7th IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157. IEEE, 1999.
- [37] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (SURF). *Computer vision and image understanding*, 110(3):346–359, 2008.
- [38] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Procs. of the 27th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2014)*, pages 580–587. IEEE, 2014.
- [39] Ross Girshick. Fast R-CNN. pages 1440–1448, 2015.
- [40] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. 2017-January:6517–6525, 2017.
- [41] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
- [42] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

- [43] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.
- [44] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *Procs. of 14th the European Conference on Computer Vision (ECCV)*, pages 21–37. Springer, 2016.
- [45] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.
- [46] Karl Pearson. Note on regression and inheritance in the case of two parents. *Procs. of the Royal Society of London*, 58:240–242, 1895.
- [47] Satoshi Suzuki and Abe Keiichi. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.
- [48] Rafael C Gonzalez and Richard E Woods. *Digital image processing, 3rd edition*. Pearson Education, 2007.
- [49] Khalid Saeed, Marek Tabedzki, Mariusz Rybnik, and Marcin Adamski. K3M: A universal algorithm for image skeletonization and a review of thinning techniques. *International Journal of Applied Mathematics and Computer Science*, 20(2):317–335, 2010.
- [50] Yung-Sheng Chen. The use of hidden deletable pixel detection to obtain bias-reduced skeletons in parallel thinning. In *Procs. of the 13th International Conference on Pattern Recognition*, volume 2, pages 91–95. IEEE, 1996.
- [51] Luca Donati, Simone Cesano, and Andrea Prati. An accurate system for fashion hand-drawn sketches vectorization. In *Procs. of the 16th IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2017.

- [52] Jiazhou Chen, Gael Guennebaud, Pascal Barla, and Xavier Granier. Non-oriented MLS gradient fields. In *Computer Graphics Forum*, volume 32, pages 98–109. Wiley Online Library, 2013.
- [53] Philip J. Schneider. Graphics gems. chapter An Algorithm for Automatically Fitting Digitized Curves, pages 612–626. Academic Press Professional, Inc., San Diego, CA, USA, 1990.
- [54] Du-Ming Tsai and Chien-Ta Lin. Fast normalized cross correlation for defect detection. *Pattern Recognition Letters*, 24(15):2625–2631, 2003.
- [55] John P Lewis. Fast normalized cross-correlation. *Vision interface*, 10(1):120–123, 1995.
- [56] Johann Heinrich Lambert. *Photometria: sive de mensura et gradibus luminis, colorum et umbrarum*. sumptibus viduae E. Klett, typis CP Detleffsen, 1760.
- [57] David G Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision, 1999*, pages 1150–1157. IEEE, 1999.
- [58] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.
- [59] TY Zhang and Ching Y. Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.
- [60] Yung-Sheng Chen. The use of hidden deletable pixel detection to obtain bias-reduced skeletons in parallel thinning. In *Proceedings of the 13th International Conference on Pattern Recognition*, volume 2, pages 91–95. IEEE, 1996.
- [61] Joon H Han and Timothy Poston. Chord-to-point distance accumulation and planar curvature: a new approach to discrete curvature. *Pattern Recognition Letters*, 22(10):1133–1144, 2001.
- [62] Pengbo Bo, Gongning Luo, and Kuanquan Wang. A graph-based method for fitting planar b-spline curves with intersections. *Journal of Computational Design and Engineering*, 3(1):14 – 23, 2016.

- [63] Patsorn Sangkloy, Nathan Burnell, Cusuh Ham, and James Hays. The sketchy database: Learning to retrieve badly drawn bunnies. *ACM Transactions on Graphics (TOG)*, 35(4):119:1–119:12, July 2016.
- [64] Gaurav Sharma, Wencheng Wu, and Edul N Dalal. The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. *Color Research & Application*, 30(1):21–30, 2005.
- [65] Joost Van De Weijer, Cordelia Schmid, Jakob Verbeek, and Diane Larlus. Learning color names for real-world applications. *IEEE Transactions on Image Processing*, 18(7):1512–1523, 2009.
- [66] Rolf Adams and Leanne Bischof. Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):641–647, 1994.
- [67] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Procs. of the IEEE*, 86(11):2278–2324, 1998.
- [68] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2017.
- [69] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- [70] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.
- [71] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceed-*

- ings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.
- [72] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8789–8797, 2018.
- [73] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.
- [74] Tomaso Fontanini, Eleonora Iotti, Luca Donati, and Andrea Prati. Metalgan: Multi-domain label-less image synthesis using cgans and meta-learning. *arXiv preprint arXiv:1912.02494*, 2019.

Acknowledgments

Grazie ad Andrea Prati per l'opportunità concessa e la costante dedizione per il nostro laboratorio.

Grazie ai miei colleghi di IMPlab: Federico, Tomaso, Eleonora, Leonardo e Akbar. Il loro entusiasmo è per me fonte di ispirazione quotidiana.

Grazie a tutti i tesisti che negli anni mi hanno aiutato o fatto pensare. Grazie ai professori del Dipartimento di Ingegneria e Architettura, sempre disponibili e rispettosi.

Grazie al Perca e al Pierca per gli insegnamenti di vita che mi hanno dato.