Semi-Interactive Optical Recognition of Integrated Circuits

by

Arjen Hoogesteger
ICA-3701573

A thesis submitted to the Faculty of Science of
Utrecht University
in partial fulfillment of the requirements for the degree of

Master of Science

Universiteit Utrecht

Faculty of Science
Utrecht University

November 2012

Abstract

In this thesis I propose a collection of machine vision techniques that could be exploited for the purpose of optically recognizing an integrated circuit (IC) in a semi-interactive manner. It will cover the entire machine vision chain, from the segmentation of the input, to the classification of the extracted information. First the seeded region growing (SRG) algorithm is discussed for segmentation purposes. Following that, I describe how a nearest neighbor classifier and a combination of an out of the box optical character recognition (OCR) solution and the Levenshtein distance can be used to provide input for a new classifier which I named the Intentionally Biased Weighed Voting Classifier (IBWVC). Experiments related to each of the individual topics have been conducted on a small self-assembled dataset. The results show that plain nearest neighbor classification is already rather accurate. Still, improved accuracy can indeed be achieved in a, by assumption, convenient semi-interactive manner by applying the IBWVC.

machine vision · segmentation · classification · semi-interactive · voting · integrated circuits

Contents

Preface

Ultimately, the reason for this thesis to exist is the artificial Tic Tac Toe opponent that I implemented over two years ago. This made me gain interest in the field of artificial intelligence which in turn moved me to enroll myself in the Technical Artificial Intelligence Master's programme at Utrecht University. The document you hold before you, this thesis, is what I look to conclude the programme with. It contains the research I conducted at Alten PTS, the company in the technical automation branch that provided me with the opportunity to work on the machine vision topic which this thesis is about.

Before we dive into details I like to especially thank both of my supervisors, Gerard Vreeswijk at Utrecht University and Jeroen Snijders at Alten PTS, for their patience, aid and advice over the past couple of months. Of course I also thank everyone else who positively contributed to the pleasant experience I had at either Alten PTS or Utrecht University.

Last, but definitely not least, I once more owe thanks to my mom and dad for their loving support and for providing me with everything I needed that allowed me to get this far.

Thank you.

Arjen Hoogesteger

Capelle aan den IJssel
November 9, 2012

1 Introduction

Over the past few years smartphones have become increasingly available and increasingly popular. This increase gave rise to numerous mobile applications that are machine vision related. One such mobile application is Google Goggles, which enables users to feed Google's search engine with images as search query. Another mentionable example of one such application, of a slightly different kind, is Layar. This application enables users to reveal hidden information in camera footage which is a popular example of what is called augmented reality.

At Alten PTS they noticed the frequently occurring issue where embedded software engineers encounter an integrated circuit (IC) for which the specifications are unknown to these engineers. Of course often, but not always, these engineers have a computer at their disposal which they can use in an attempt to identify the IC in question. As long as the imprint of the IC is fully readable this can often be done with a substantial rate of success. If this is not the case however, which is not uncommon, these engineers are pretty much left clueless.

The idea was put forward to enable a modern smartphone to optically identify integrated circuits. Ideally, engineers would take out their smartphone and use its camera for the purpose of identification. An engineer would take a picture of an integrated circuit, the application would extract the IC from the picture and compare it amongst a database of known ICs. Finally it would, like Google Search does when you provide it a query, provide the engineer with a list of possible matches sorted by likeliness. Datasheets containing specifications for these ICs could be linked to these results such that engineers can obtain these by downloading them.

In this thesis I propose a collection of techniques, with which I experimented, that should contribute in the search for a suiting machine vision solution to this problem. Machine vision, being an artificial intelligence (AI) subtopic, does not strictly infer a fully automatic solution. The pictured solution may very well contain a couple of manual operations as long as the overall performance benefits from them and with the restriction that the solution remains convenient to use.

The research question that is central to this thesis and which, by answering it, should at least get us closer towards a solution is defined as follows:

> Through which applicable machine vision concepts and techniques can we **localize**, **extract** and **classify** integrated circuits on an arbitrary printed circuit board?

In section 2 I globally cover the machine vision concept. We will see that this concept can be decomposed such that we visualize it as a chain of processes that need to be subsequently executed. I use this same section to explain the pictured solution, and how my research and this envisioned solution relate to the field of artificial intelligence (AI).

The subsequent sections 3 and 4, each cover research I conducted regarding the machine vision sub-processes. In section 3 I discuss the seeded region growing algorithm which I exploited for the purpose of image segmentation. In our specific case, the purpose is to obtain a segmentation for the input image such that we can extract the IC from our input image. In section 4 I discuss classification techniques that enable us to compare visuals of ICs. Upon extracting the IC of interest from our input image these techniques enable us to compare this IC with data of ICs that was previously persisted in a database. In order to do so reliably, I introduce a classifier which I named the Intentionally Biased Weighed Voting Classifier (IBWVC). Throughout both sections I provide examples of algorithms and discuss results that were obtained by conducting experiments.

Finally, with section 5 I conclude this thesis with answers to the research question, a discussion about my experiments, the obtained results and suggested future work.

2 Machine Vision

> *That's a very nice rendering, Dave. I think you've improved a great deal. Can you hold it a bit closer? That's Dr. Hunter, isn't it?*
>
> HAL 9000

Before we dig into the details of the project I will use this section to provide you with some general background information on the concept of machine vision. Machine vision as a process could be divided into three distinct sub-processes. For each of these sub-processes I will provide a brief explanation. Finally I will use this section to elaborate on the chosen approach and how this approach relates to the field of artificial intelligence (AI).

2.1 Definition

Snyder and Qi [24] define **machine vision** as

> "the process whereby a machine, usually a digital computer, automatically processes an image and reports 'what is in the image' "

Through this definition we can explain the fundamental distinction that exists between **image processing** and machine vision. An image processing system applies one or more image processing techniques to an input image such that it results in another version of the input image. Examples concern contrast stretching or brightness scaling for instance. A machine vision system on the other hand, attempts to output information about the contents of an image.

Machine vision could be considered a combination of image processing and machine learning. Image processing techniques are usually exploited for the purpose of input preparation like object and feature extraction, whereas machine learning techniques are exploited for classification purposes. Figure 2.1 describes a basic and common chain of processes that makes a machine vision system.

Upon obtaining an image, usually by means of a digital camera, segmentation is performed to extract the region(s) of interest. Subsequently, features that are required for classification purposes are extracted from the region(s) of interest. Finally the extracted set of features serves as input to the classification process which results in information on the region(s) of interest. In our case this should result in information that tells us which kind of IC we deal with in the obtained picture.

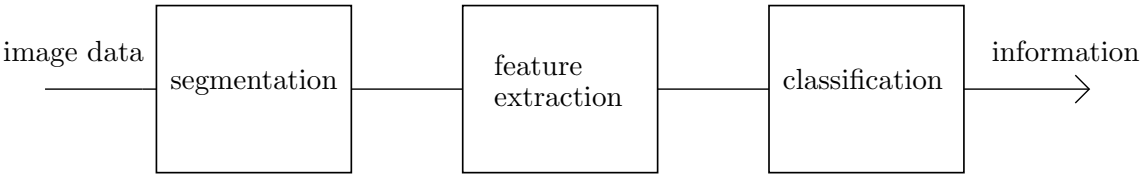


Figure 2.1: A chain of sub-processes that is common for machine vision systems.

In this thesis I put emphasize on the segmentation process and the classification process. Feature extraction is briefly discussed since it is crucial to the classification process. In the next couple of paragraphs I will provide a brief explanation of the purpose of each of these sub-processes.

2.2 Segmentation

The IC localization part of the research question concerns an image processing part of the solution. For most images a large part of the image data will be of no interest to our machine vision solution. In our case, the image in which our IC of interest resides, could for example also contain surrounding electronic components (e.g. capacitors and resistors) or a substantial part of it could simply be made up of the printed circuit board to which the IC is attached.

Using image processing techniques the goal of the segmentation process is to reliably extract the part(s) of the image data that show our IC of interest. Upon doing so we successfully localized our so-called region of interest.

2.3 Feature Extraction

After we have extracted our region(s) of interest from the input image we aim to extract the features of interest from these regions. Evidently this requires feature selection first. Which IC specific features can we extract from images of ICs and which are able to serve as suitable input for the classification process?

Feature extraction is another process that is image processing related. Several image

processing techniques could be performed such that we obtain another version of the image from which we can conveniently extract these features of interest.

Since feature extraction is of crucial importance to the classification process and this thesis does not cover automatic feature extraction in detail, my experiments relied on a manual feature extraction operation.

2.4 Classification

The final sub-process in the machine vision chain concerns the classification of what was previously marked as the region of interest. Which ICs were discovered in the processed image? This part of the solution requires the features, which were extracted in the previous process in the chain, as input. These features are used to compare the query instance with features from instances which have previously been collected and persisted in a database. For the purpose of comparison, machine learning techniques are exploited.

It is rare for classification techniques to report a perfect match. Therefore, we look for the system to provide a list of classes with which the query instance matches to a certain degree in which the best match ranks first, the second best match ranks second, etc. The user ultimately determines the quality of the classification result that was obtained. A process very similar to what happens when you provide internet search engines with a query.

2.5 Intended Approach

According to Marvin Minsky artificial intelligence

> "is the science of making machines do things that would require intelligence if done by men"

If we accept this definition of AI it is fair to assume that the machine vision discipline can be considered one of many sub-disciplines of the field of AI. Traditionally this definition causes the image of machines that completely replace one or several humans given a specific task. I adopted a slightly different approach in which the presence of human intelligence remains crucial. The aim was to come up with a conveniently usable semi-interactive solution with focus on reliability rather than aiming for a fully automatic solution for which focus would surely be put on overcoming complexity issues.

In Turings Tango [19], Bennie Mols, a Dutch researcher and scientific journalist, describes why he believes computer intelligence will never surpass human intelligence. It describes what he refers to as the Turing Tango with which he provides a different view on AI than the traditional Turing Test does. According to a Turing Tango, the goal of AI should not concern the creation of a machine that meets every aspect of our intelligence. Instead, it comes down to finding the ideal human-machine combination, in which we use computer intelligence to complement our intellectual shortcomings and vice-versa.

In order to find a suiting balance in the tasks that need to be performed per entity, we need a clear view of their qualities and their limitations. Humans generally perform better at recognizing patterns than computers do for instance. Computers on the other hand are flawless at quickly performing a huge amount of arithmetic operations or performing search operations on huge stacks of data. A computer's tirelessness is another mentionable strength.

With this concept in mind this thesis provides a solution in which manual actions were favored over automatic actions as long as the computer benefits from them and they do not decrease the solution's usability in a drastic manner.

3 Segmentation

Divide et impera

Julius Caesar

Before there is anything we can analyze and classify we need to process our input image such that we clear it of irrelevant data. In other words, we aim to isolate our object of interest from its environment, the relevant foreground from the irrelevant background. A result like this can be obtained by finding a segmentation for an image such that a single segment or a combination of connected segments make up our object of interest. In our case an integrated circuit would be the foreground area we are interested in and the background would be the printed circuit board and every other component on it.

Several types of segmentation techniques exist out of which I had to make a choice of one that seemed promising in terms of some of its properties. I will use this section to elaborate on the segmentation technique which I chose to exploit, why I chose to do so, the observations I did using this technique, the pros, the cons and the simple adaptation I made to the algorithm in favor of some domain knowledge.

I will use the following couple of paragraphs to justify my choice for the seeded region growing (SRG) algorithm. In these paragraphs I will elaborate on the motives that made me choose this algorithm. It is no obvious matter of right or wrong. Of course another algorithm could just as well have been chosen as long as there were the motives to defend that choice. In short, a combination of the algorithm's core idea, its elegance, its simplicity and a few problems I foresaw with other type of algorithms made me make my choice.

Through the course of years many segmentation algorithms have been invented, many of which are variations on algorithms that, at that point, already existed. Also, many of these variations have been designed for specific purposes. Fortunately we can roughly divide the segmentation algorithms in four groups namely threshold-based, region-based, edge-based and hybrid algorithms. The hybrid algorithms are the ones that cannot be assigned a single category and aim to combine best of multiple.

Threshold-based techniques determine for each pixel individually which class it belongs to given a certain threshold. A threshold rule might be something as simple as "if the intensity value of a pixel is between 100 and 150, this pixel is part of our region of interest". An algorithm that takes care of segmentation of this type, can be defined in a rather straightforward manner. We could loop over all individual pixels and assign each of them a class label by applying our rule or our set of rules. Determining our threshold(s) is however not that straightforward. In an industrial environment that does not change over time we could manually determine them. In other rapidly changing environments, especially in terms of light and shadow, we could end up having to manually adjust our threshold repeatedly before each use. This is of course undesirably time-consuming. Algorithms that automatically attempt to determine the threshold(s) exist and often make use of the information we can extract from an image's histogram. Another notable issue with threshold techniques is the fact that they generally do not make use of the spatial information in the image. As a result, each individual pixel is assigned a label. The collections of pixels that should make up the segments, are however still to be decided.

Edge-based techniques operate on the gradient of an image. The steeper the gradient between two pixels, the more likely it is to represent an edge. Clearly, a technique like this suffers from similar issues as the threshold-based techniques do. Namely, when is a found edge strong enough to be considered an edge? Or, when is it to be considered nothing but noise? Again this requires us to have some sort of threshold for us to be able to make a decision in this. A threshold which is greatly influenced by the environment's ambient light. Apart from that we can often not trivially determine how to connect the found edges such that they form closed regions.

Region-based techniques begin with a collection of elemental regions that they then begin to grow, merge or split. This process is guided by a rule or a set of rules that aim to keep the pixels in a region homogeneous in some sense and to some extent (e.g. color or intensity). The seeded region growing algorithm I will introduce in the next section is one of these types of algorithms. It requires a selection of elemental regions (seeds) which will grow until each pixel in the image is part of one of these regions. Therefore, the amount of seeds determine the amount of segments the algorithm will end up in. The algorithm depends on distance measures instead of thresholds which makes it less vulnerable to variance in environment light. From a practical point doing the seed selection manually does not come with any disadvantages. In contrary, it is preferred over automatic seed selection methods that exist as it provides the user a way to guide the algorithm. In our case, our aimed application targets modern smartphones with which we would capture an image of an integrated circuit. From a practical point of view it also makes perfect sense for a user to tap the integrated circuit of interest. Discovering the areas of interest, in our case the integrated circuits, would take additional computing time whereas the user will almost instantly be able to point out the area of interest to the computer.

The book by Snyder and Qi [24] is an excellent source to read more on the type of image segmentation techniques that were just discussed.

3.1 Seeded Region Growing

The seeded region growing algorithm is a somewhat older region-based algorithm, first described by Adams and Bischof [1], however still actively used as a segmentation algorithm.

Mehnert and Jackway [17] proposed an improved version of the algorithm which they named the improved seeded region growing (ISRG) algorithm. The ISRG algorithm improves on the fact that the resulting tessellation of the SRG algorithm is dependent on the order of pixel processing. Simply put, given a certain image and a certain set of initial seeds, the output of the SRG algorithm is not guaranteed to be consistent if you apply the algorithm several times and there happens to be variation in the order of processing of the pixels. I used the original algorithm for this project as it is the more simple version of the algorithm and the improved version did not improve on anything that I assumed to be significant for my solution. In addition to that, improving the algorithm also made the algorithm perform more slow.

More recent publications exist in which the SRG algorithm is exploited to serve their specific purposes. Shih and Cheng [22] proposed an automatic seeded region growing algorithm for color image segmentation based on the original SRG algorithm. In the same year Fan *et al.* [8] published a paper in which the SRG algorithm is analyzed and several additions and improvements to the algorithm are proposed. Even more recent Avazpour *et al.* [2] used the SRG algorithm to find a tessellation for CT images. One of their conclusions about the algorithm is that despite the algorithm's ease of implementation, it can very well be used to obtain quality segmentation results. Next I will explain the details of the algorithm and take you through a simple example scenario. By the end of this section it should be clear that the combination of simplicity and effectiveness indeed makes for an elegant segmentation algorithm.

Algorithm

The algorithm starts out with some initial set S of n sets of pixel coordinates and an input image I, where

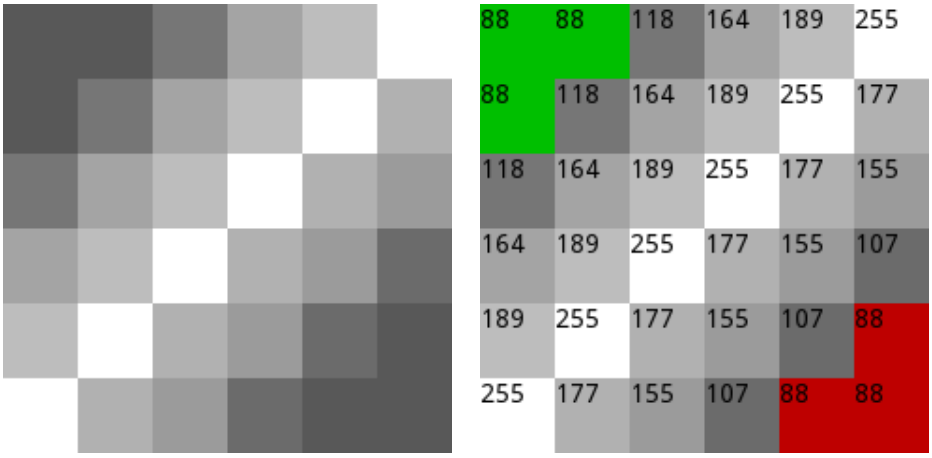
$$S = \{s_1, \ldots, s_n\} \tag{3.1}$$

Each of these sets of pixel coordinates s_i is referred to as a seed, and the pixel coordinates originate from the input image I. For our example we will consider the image in Figure 3.1(a) our input image. For now we will, like in the original publication, consider intensity

images only. The input image can be represented by a two-dimensional array which contains an intensity value for each possible coordinate pair $p = \langle x, y \rangle$. Therefore, we define each seed such that

$$s_i = \{p_1, \ldots, p_n\} \tag{3.2}$$

The two-dimensional data structure allows for retrieval of an intensity value $I(p)$ given a certain coordinate pair that remains within the image's width-height boundary.



(a) 6 x 6 intensity image (b) intensity values and seed selections

Figure 3.1: Two images of an example 6 x 6 intensity image. In (a) we display the original image on which we will, by means of example, apply the SRG algorithm. Image (b) shows the same image only this time the intensity values per pixel are displayed and two seeds have been selected.

As the name of the algorithm suggests, it makes each distinct seed $s_i \in S$ grow into a single distinct region. It might not be very straightforward how both terms, seed and region, relate to each other. Basically a seed is the set s_i in its initial state. Over time this set will be adjusted such that the seed set undergoes a transition from $s_i \rightarrow s_i'$ such that $s_i' = s_i \cup p$ and p is the first pixel to be added to the seed set. Over time similar transitions will repeatedly take place for the resulting set s_i' as long as the algorithm is not finished. Any set s_i' that results from a seed set s_i is referred to as a region. In other words, regions are the sets of pixels that result from applying the SRG algorithm to the sets of pixels called seeds.

In Figure 3.1(b) we see the same image as in Figure 3.1(a). In addition, this time some properties have been marked. The numbers show the intensity value per pixel. The top left corner's coordinate in the image is $(0, 0)$, therefore, the intensity $I(\langle 0, 0 \rangle) = 88$. The green and red areas mark the initial seeds. For the green seed, for instance, we have that $s_{green} = \{\langle 0, 0 \rangle, \langle 1, 0 \rangle, \langle 0, 1 \rangle\}$. These seeds have been selected manually with the structure of the image in mind. Clearly the image consists of two areas separated by a white diagonal line. Ideally these two seeds would grow such that both resulting regions meet at the white line. This, we will see, turns out to be almost the case, which clearly demonstrates the importance of proper seed selection. If we would have selected only a

single seed for instance, the only region the algorithm would come up with would simply be the entire image.

At this point the algorithm is basically set to go and label a single pixel at a time until all pixels in the image have been labeled. Labeling a pixel refers to the process of assigning a pixel (its coordinates) to one of the available seeds. If a pixel is labeled, it is part of a seed. Otherwise it is unlabeled and yet to be assigned one. More formally, let p be a pixel in the neighborhood of S and f_{lbl} be the function of p that determines whether or not it is labeled, then

$$f_{lbl}(p) = \begin{cases} true & \text{if } p \in \bigcup_{i=1}^{n} s_i, \\ false & \text{otherwise.} \end{cases} \tag{3.3}$$

Pixels that are candidates to be labeled are those that are in the neighborhood of the seeds. The neighborhood of a set of pixels P is defined in a straightforward manner. It is the set of pixels N that contain all pixels that are directly connected in a horizontal, vertical or diagonal manner, to at least one of the pixels in P, such that the property $N \cap P = \emptyset$ holds.

The algorithm works with the unlabeled neighborhood of S. It processes one pixel at a time until each pixel of the image at hand has been labeled and a tessellation for the image has been found. The unlabeled neighborhood is a set of pixels that are in the neighborhood of S and which are unlabeled.

The order in which the candidates are added to a seed is determined by a distance measure. A distance measure which determines the distance from an unlabeled pixel to a seed in terms of some measurable property other than the distance on the raster. The algorithm as is, exploits the distance in terms of intensity. It keeps track of the mean intensity value m_i per seed s_i. The first time the mean is calculated is right upon seed selection. The mean intensity is defined such that

$$m_i = \frac{\sum_{p \in s_i} I(p)}{|s_i|} \tag{3.4}$$

After the first time the mean has been calculated, each time a pixel is added to the seed, the value m_i is simply recalculated such that the property in Eqn. (3.4) is maintained. The distance from a candidate pixel p to a certain seed s_i can now simply be determined by applying the distance measure δ such that

$$\delta(p, s_i) = |m_i - I(p)| \tag{3.5}$$

Figure 3.2 shows the first step after having selected seeds and having determined the mean

value per seed. For each pixel in the unlabeled neighborhood of the seed s_i the distance to s_i is calculated. It is not unthinkable for a pixel to be in the unlabeled neighborhood of two distinct seeds at the same time. In this case, the pixel is considered neighbor to the seed it is closest to and therefore the minimum of these two distances is calculated. In case of Figure 3.2 this is not the case. For both of the seeds it holds that $m_{green} = m_{red} = 88$. Distances for all pixels in the unlabeled neighborhood of both seeds have been computed and displayed in blue.

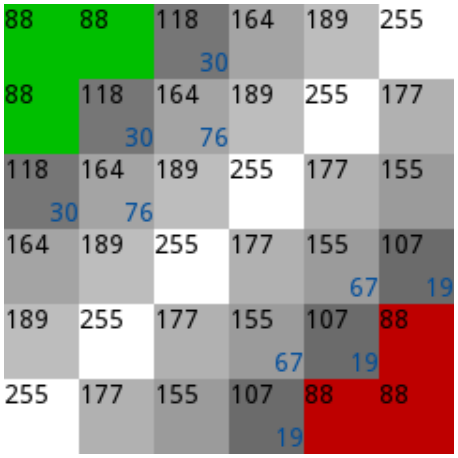


Figure 3.2: The blue numbers display the distance from the pixels to the seed each pixel is directly connected to.

Next, these unlabeled neighbors, are stored in a sequentially sorted list (SSL) which maintains an ascending order in terms of the computed distance. Then, the top pixel is selected, removed from the SSL and assigned to the seed it borders. In case of our example, the pixels for which the distance is 19 form the top of the SSL, and one of them is arbitrarily selected to be labeled. Figure 3.3 shows the result of this process. Remember that after labeling a pixel, the region's corresponding mean is recomputed.

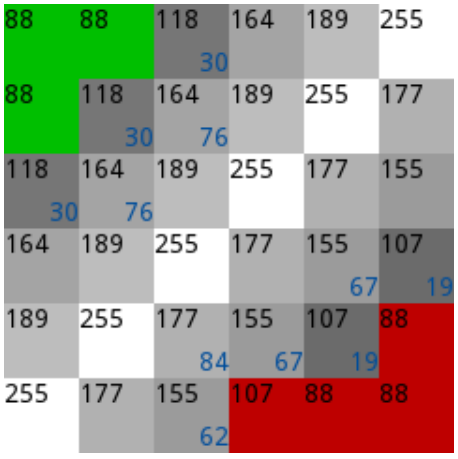


Figure 3.3: The result of labeling the first pixel atop the SSL.

In case this results in new unlabeled neighbors, which was the case in Figure 3.3, for all of these new neighbors the distance to the neighboring seed is calculated and these new

neighbors are inserted in the SSL accordingly. This entire process of popping the top of the SSL and assigning the pixel to a seed, is repeated until the SSL is empty.

If the SSL is empty, all pixels have been assigned to a seed and we have found a segmentation for the input image.

At some point the algorithm reaches a state in which all of the pixels, are not necessarily in, but have been inserted in the SSL. In case of our example this has been visualized in Figure 3.4(a). The only thing left to do is labeling each of the pixels that remain in the SSL. Note that even updating the running mean for a seed has become superfluous. At no point in the algorithm we update the position of a pixel in the SSL. In other words, we never recompute the distance for the unlabeled neighbors. The reason this is not done in the SRG algorithm is speed. The algorithm has been designed with performance in terms of speed in mind. Therefore the distance per pixel is only computed once and the SSL ordering is simply maintained on insertion of a new pixel. This makes for a rather fast algorithm.

(a) no new additions to SSL anymore (b) first pixel flagged as boundary

Figure 3.4: In (a) we display the first state from which no more pixels will be inserted in the SSL. In (b) we reached the next state after the state in (a) in which the first of the pixels has been flagged as a boundary pixel (the black pixel).

In Figure 3.4(b) a state is visualized which we have not previously seen. A state which happens to be the direct successor of the state in Figure 3.4(a). This time one of the pixels has been flagged as a boundary pixel. This concept of boundary pixels is rather straightforward. Pixels that border more than a single seed are flagged as being a boundary pixel between these multiple regions. This is done for visualization purposes. It is a convenient way of visualizing the different resulting segments. If we prefer to do so, we could of course, like mentioned before, assign the pixel to the seed it is closest to instead. We would then, however, in order to visualize our results, need to find other ways.

Figure 3.5 concludes our SRG toy example. The black line forms the boundary between

the two regions. Clearly it is not exactly the result you could have hoped for as the boundary line contains pixels that should belong to the top left region instead. The white line for instance, which is part of the image, should ideally be the only area in the image that is flagged as the boundary between the two grown regions. The fact that pixels are also considered neighbors when they are directly connected in a diagonal way makes this situation impossible. Mistakes of this kind turn out to be only minor mistakes however. For this example it is the size of the image that makes it seem a significant amount of mistakes. Only a few pixels of error in a small image like in Figure 3.5 quickly seems rather substantial. These mistakes are not however, considering that I experimented with images that are built of several thousands of pixels.

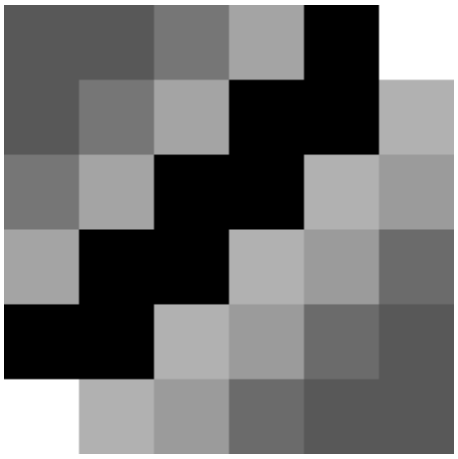


Figure 3.5: The segmentation result in which the black line displays the found boundary.

Changing the algorithm such that we do not flag pixels as boundary pixels does not correct these type of mistakes either. In this case, one of both regions would end up containing the white pixels. Basically, we face a situation in which the image contains three regions and not two. Ideally the white line, which could for example be part of a background area, would be considered a region of its own. However, the nature of the algorithm prevents this from happening as long as we stick with the two seeds we selected for this example.

At the same time, this same nature, provides very much desirable behavior in some other situations. Namely, it allows the algorithm to engulf noise in images, instead of it to be considered a separate region. Still, on the other hand, especially when we consider manual seed selection, chances are you forget to place a seed or you lack accuracy in placing them and end up with a couple of regions too few with respect to the structure of the image.

Algorithm 1 describes the flow of the SRG algorithm in pseudocode. It is assumed that the function calls that are made are straightforward and self-explanatory. The procedure's argument I concerns the image, the the argument S the set of seeds and the argument ssl an empty SSL.

Algorithm 1 Seeded region growing algorithm

```
1: procedure SeededRegionGrowing(I, S, ssl)
2:     for all s_i ∈ S do
3:         Insert(ssl, UnlabeledNeighbors(s_i, I))
4:     end for
5:     while ¬Empty(ssl) do
6:         y ← Pop(ssl)
7:         ln ← LabeledNeighbors(y, I)
8:         if PixelsEquallyLabeled(ln) then
9:             l ← GetLabel(ln)
10:            Label(y, l)
11:            UpdateMeanForSeedWithLabel(l)
12:            un ← UnlabeledNeighbors(y, I)
13:            while ¬Empty(un) do
14:                n ← Pop(un)
15:                if ¬In(ssl, n) then
16:                    Insert(ssl, n)
17:                end if
18:            end while
19:        else
20:            FlagAsBoundary(y)
21:        end if
22:    end while
23: end procedure
```

Observations

For the initial results I used the algorithm as it was originally described. No changes. The original algorithm was designed to work with intensity images. An intensity image or grayscale image can be obtained from a color image in a rather straightforward way, as we will see shortly. However, such image leaves us with insufficient information. From an intensity image, it is not possible anymore to tell color and lack of color apart as each pixel becomes a tint of gray ranging from black (no brightness) to white (total brightness).

First I will explain the transition from a color image to an intensity image. This, in combination with the SRG algorithm and a set of test images forms the basis for the initial results and observations. Most of the information which I will provide about color and color spaces, you can also find in the image processing introductory book by Burger and Burge [6]. This book turns out to be a very accessible source if you are interested in image processing and have few or no knowledge in this discipline.

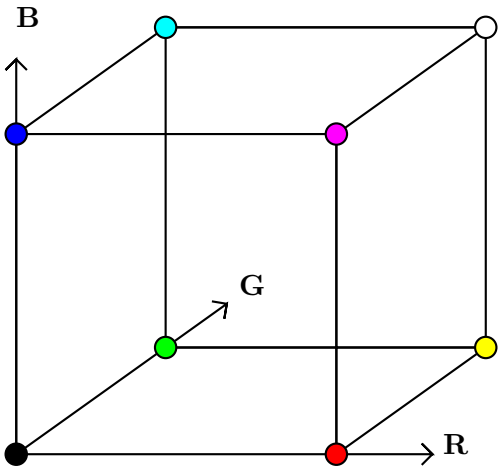


Figure 3.6: A visual representation of the RGB color space.

Most commonly, when a digital camera is used to capture an image of a scene, the result is stored as a 24-bit or 32-bit RGB image. This has nothing to do with, and is not to be confused with, the file format, such as GIF and PNG are. In case of a 24-bit RGB image, RGB refers to the color space being used to represent the color in, and the image is of 24-bit depth, which tells you the amount of bits that are available to represent a single pixel's color. Basically the depth specifies the precision with which we are able to represent such color. In the RGB space, color is represented by the three distinct components red, green and blue which is visualized in Figure 3.6. In case of the 24-bit example a single byte is reserved per component. This leaves space for $256^3 = 16{,}777{,}216$ possible colors for which we can not trivially determine how close a certain combination of RGB values is to be regarded to as being gray(ish). The one thing we do know for certain is that, if

$$R = G = B \tag{3.6}$$

holds, the color is hueless/gray. This property is exploited to make the transition from an RGB color image to a grayscale image. As we will see, this process is rather straightforward. It is essential to have clear however as it emphasizes the issue we face if we are to say something about color difference between pixels or areas of pixels given an intensity image.

To convert a RGB image into an equivalent grayscale image we should compute the intensity value Y for each RGB pixel. Next all we need to do is replace each individual RGB component's value by the value of Y, meaning

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} \leftarrow \begin{pmatrix} Y \\ Y \\ Y \end{pmatrix} \tag{3.7}$$

The simplest way to find a value for Y is to average over the three RGB components. The

simplest equation that achieves this is where

$$Y = \frac{R + G + B}{3} \tag{3.8}$$

This is the simplest, but not a very sophisticated way. It turns out that we, us humans, do not perceive each of the color components equally bright. We seem to perceive green and red a lot brighter than we do blue for instance. As Figure 3.7 shows us, this is not taken into account in Eqn. (3.8).

(a) (b) (c)

Figure 3.7: If we apply Eqn. (3.8) on both colors in (a) and (b), both results would equal the intensity displayed in (c).

(a) (b)

Figure 3.8: If we apply Eqn. (3.9) on Figure 3.7(a) our result is (a). In (b) the result of applying the same equation on 3.7(b) is displayed.

This therefore makes it more sophisticated to assign a weight to each of the RGB components instead of treating them as equals like we did thus far. We therefore prefer to use that

$$Y = w_R \cdot R + w_G \cdot G + w_B \cdot B \tag{3.9}$$

We could of course use this equation to obtain the same result as with Eqn. (3.8) by setting all weights to $\frac{1}{3}$. For my experiments I adopted the weights that are recommended in ITU-BT.709 [12] for digital color encoding. In this recommendation it says that the weights should be set such that

$$\begin{aligned} w_R &= 0.2126 \\ w_G &= 0.7152 \\ w_B &= 0.0722 \end{aligned} \tag{3.10}$$

Figure 3.8 shows the results of applying Eqn. (3.9) to the colors in Figure 3.7(a) and 3.7(b). It should be evident that, no matter the method of obtaining an intensity image, there is no way of telling color and lack of color apart from such image. We will never be able to reconstruct the original color image from an intensity image. By converting from 24-bit RGB to 8-bit intensity we drastically decrease the size of our color space and several distinct RGB combinations could very well result in the exact same intensity upon

convergence.

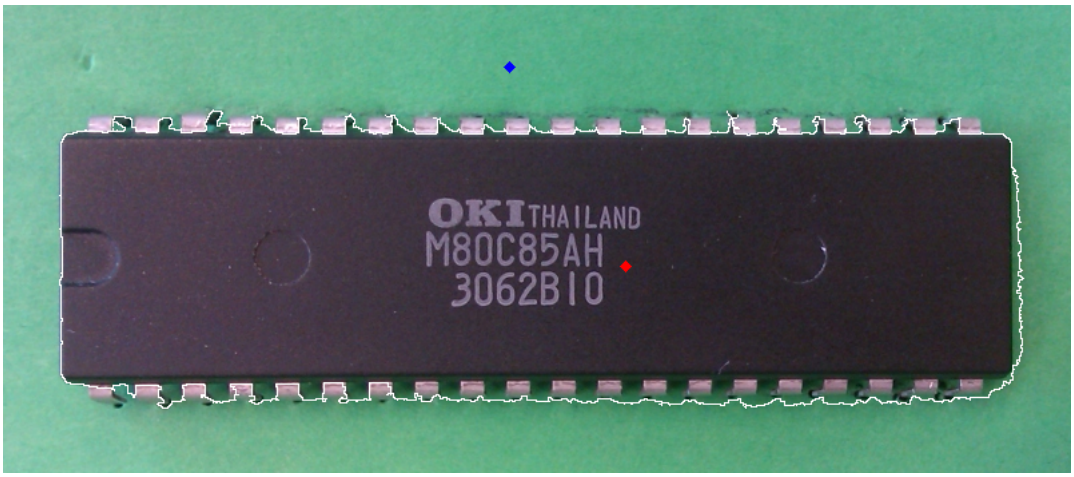


Figure 3.9: Example result of the original SRG algorithm applied to the OKI chip.

Example results of a few first experiments are shown in Figure 3.9 and 3.10. In both the images the red dot indicates the foreground seed that has been placed to grow the IC area. The blue dot indicates the background seed which has been placed such that it grows a region that contains everything that remains. The white line indicates the resulting boundary which separates both regions.

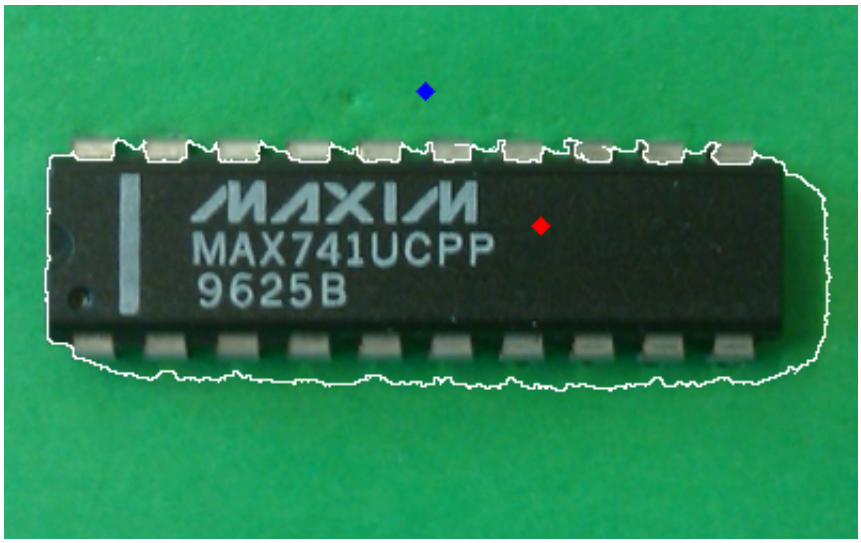


Figure 3.10: Example result of the original SRG algorithm applied to the MAXIM chip.

The first notable issue concerns shade. Both images, especially the image in Figure 3.10, clearly show mistakes that are being made in case of shadow present in the image. This is simply due to the fact that the intensity value for very dark shades of green, are closer to intensity values for dark shades of gray, then they are to intensity values for the surrounding lighter shades of green. As mentioned before, when we transform a color image to its corresponding intensity image difference in color is no longer taken into account.

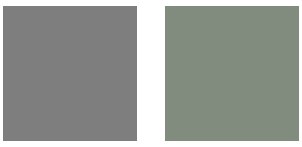
For the same reason the pins of both ICs are repeatedly marked as part of the background region whereas they should ideally be considered part of the foreground region. In contrast, the areas between the pins, are often wrongly marked as part of the foreground region whereas these areas should in fact be considered part of the background region. In the next paragraph we aim to improve on these particular issues.

3.2 Embedding Domain Knowledge

To improve on the initial results I decided to embed some domain knowledge in the algorithm. The characteristics that could be used include the fact that integrated circuits, by assumption, are always

- rectangular shaped;
- gray tinted.

I decided to go with a fairly simple adaptation in favor of the knowledge we have concerning the color characteristics of an integrated circuit. As we will see, an adaptation of this kind does not require changes of substantial impact on the original algorithm. It seems is assumed to expect that an extension for the algorithm, to somehow incorporate domain knowledge about the shape of our region of interest on the other hand, could be quite impactful.



(a) (b)

Figure 3.11: In (a) we show a color for which we can say with total confidence that it is gray as all RGB components are valued equal. For (b) it is not quite that trivial, not all RGB components are valued equal. Which leaves us wondering, is it gray, green, grayish or greenish?

Remember that our aim is to improve on the initial segmentation results and that it would be a waste of useful information if we were to stick with the original algorithm for which we need to convert the input color image to an intensity image. Sticking with the RGB representation of our image is however also troublesome. The RGB representation does not allow us to trivially determine whether a certain RGB combination $RGB = \langle R, G, B \rangle$ represents a tint of gray. For Figure 3.11(a) for example, where $RGB = \langle 126, 126, 126 \rangle$ we can say with complete confidence that it is a gray tint. For Figure 3.11(b) where $RGB = \langle 130, 140, 126 \rangle$ it is not quite that clear anymore which color this concerns. One might call it a shade of gray, another might name it a shade of green.

An approach could be discretizing the RGB space such that it allows us to take the three RGB channels as arguments and results in a color name such as "light green", or "dark blue" for instance. This problem of finding suitable English names for RGB values such that colors are easy for us to identify automatically, has been addressed many years ago by Berk *et al.* [3] when they presented their Color Naming System (CNS). In follow up on this, Farhoosh and Schrack [9] suggested a mapping from the CNS to colors in the HLS color space. An accurate mapping similar to this could allow us to abstract from the RGB values and instruct the computer to extract only the areas labeled "gray", "dark gray", or "grayish" for instance. A system like this would indeed allow for something very like this, unfortunately it does not seem like a trivial extension that is easily applied to the SRG algorithm.

Nevertheless this concept still deserves some brief attention. This rather old research addresses the problem of the gap between how humans deal with color and how computers do. For many machine vision applications, a smooth transition between how we quantify colors and how computers represent them could be very useful. An example where something similar has been attempted was published by Das *et al.* [7]. In their publication they describe a method that allows for searching a database, which contains images of flowers, by color name. Another publication by Liu *et al.* [15] exists in which a similar, more generic, system is described. Unfortunately in both publications they first perform segmentation and afterward determine the color name per region. It would be more interesting to have a system in which the segmentation algorithm is guided by the queried color names.

This leaves us with more information on the problem known as the semantic gap, yet not with a solution to our problem. In the paper by Farhoosh and Schrack however, the HLS color space is exploited for the purpose of color naming. As it turns out, this color space can also be of use to our solution if we adopt a slightly different approach. Instead of trying to attach a color name to a color, we are really interested in measuring a given RGB color's distance δ_{gray} to the arbitrary gray tint it is closest to. As turns out, a distance measure like this is easier defined if we represent our colors in HLS space since we could ignore one of the HLS components for this purpose. If we were to achieve something similar in RGB space we would need all three available components.

In HLS a color is represented, just as in RGB, by three distinct channels. In HLS we have that H determines the hue of a color, L the lightness and S the saturation. The values for the channels, lightness and saturation lie in the range $[0, 1]$. In case of $L = 0$, this means total absence of light whereas $L = 1$ means the opposite. The S value determines the saturation or in other words the amount of color where $S = 0$ means no saturation, or, no color. Again, $S = 1$ means the opposite. The value for H should be in the range of $[0, 360]$ as it represents an amount of degrees. Of course H could also be represented in the range of $[0, 1]$ which was the case for my experiments.

No matter the choice of color space, either RGB or HLS, in both spaces we are capable of defining the exact same color. In other words, a color represented in HLS can also be represented in RGB and vice-versa. Conversion methods from RGB to HLS and from HLS to RGB to exist. We will not deal with the details of these methods.

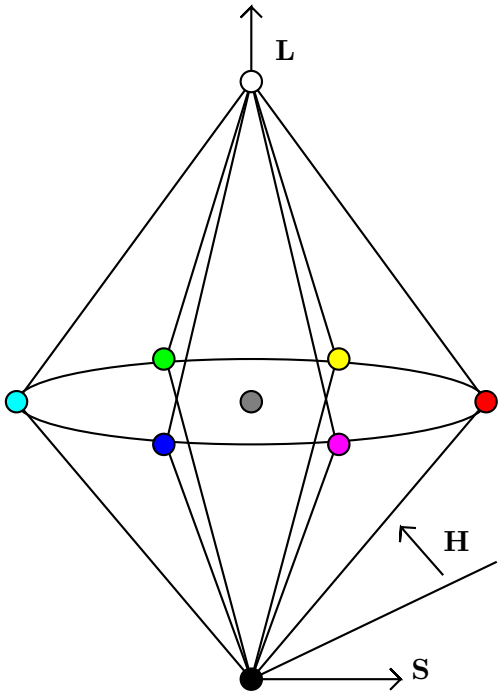


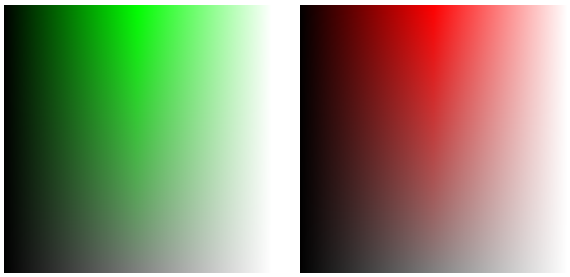
Figure 3.12: A visual representation of the HLS color space.

The HLS space is visualized quite different compared to the RGB space, as illustrated in Figure 3.12. Instead of a cube, the space is illustrated as two cones stacked on top of one another[1]. This color space is, along with the HSV color space, most commonly used among artists and designers as the preferred space for color picking purposes. It is less commonly used in image processing applications.

The HLS space has the convenient property that it allows us to ignore the hue channel if all we are interested in is whether or not a certain combination can be regarded to as being gray. This property is visualized in Figure 3.13. In both of the images in this figure we fixed the hue channel to some value in contrast to the lightness and saturation channels. The lightness channel ranges from 0 to 1 over the horizontal axis of each image, while the saturation channel ranges from 0 to 1 over the vertical axis. The simple conclusion we can draw from these images is that, no matter the value for the hue channel, if the lightness channel is valued either 0 or 1 we are guaranteed to deal with a gray tint. The same goes for the saturation channel. If the saturation channel for a certain HLS combination is set to 0, then the HLS combination is also guaranteed to represent a gray tint. In fact, in

[1]Since the range of S does not depend on the value for L, the shape of the color space should really be a cylinder. The only difference is that this would add more black, and more white, to the color space. From a practical point of view it is perfectly possible to define a color for which $L = S = 1$, for instance.

these situations the combinations in question are actually guaranteed to be white or black, which are, recall the property in Eqn. (3.6), also considered gray tints.



(a) $H = 120°$ (b) $H = 0°$

Figure 3.13: Two visualizations of the HLS space for which the hue value is fixed.

This makes that we are interested in the minimum distance from a specific point in HLS space to the lightness extremes or the saturation minimum. This is at least a decent approximation. As is clearly shown in Figure 3.13, the colored area is not square shaped. It is more parabolic shaped. Due to complexity issues that would surely arise when we were to consider the more complex parabolic shape we will only consider this approximation.

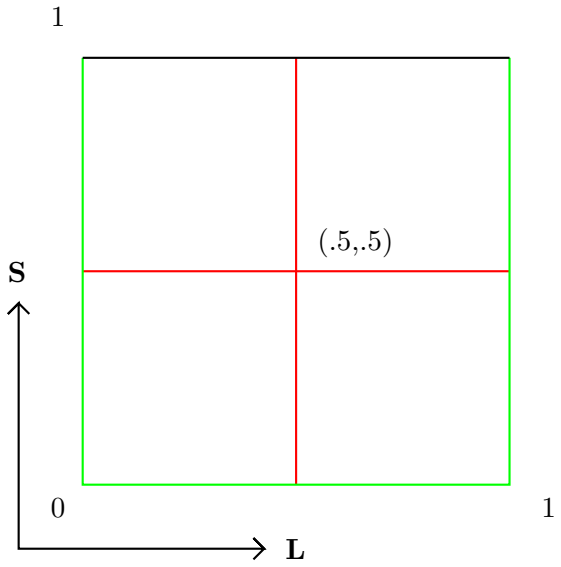


Figure 3.14: Schematic representation of the HLS space. The red line shows the division of the space we need to approximate the distance δ_{gray}. We define δ_{gray} as the minimum distance to the green line.

The resulting function that calculates our distance δ_{gray} is given in Algorithm 2 and is based on the idea pictured in Figure 3.14. First we cut the color space in four even quarters. For the top two quarters, the part where $S \geq 0.5$, it counts that we are only interested in the minimum of the distances to $L = 0$ and $L = 1$. Basically we need to determine on which side of the vertical red line we are first. The distance to the horizontal green line, to $S = 0$, is only relevant if our point of interest lies in either of the two lower quarters for which $S < 0.5$. In the next paragraphs the experiments including this new distance measure and their corresponding results are reviewed.

Algorithm 2 Gray distance approximation

```
1: procedure DistGray(L, S)
2:     L' ← min(L, 1 − L)
3:     return min(L', S)
4: end procedure
```

Observations

In case of the original algorithm we could transform an input image to its corresponding intensity image. This provides us with a convenient way of visualizing what a computer "sees" when we consider an intensity image. We could visualize something similar for our gray distance measure.

Recall that the gray distance for each pixel which we just defined is in the range of $[0, 0.5]$. We could transform a given input image into a corresponding gray distance image in which we color each pixel according to its computed gray distance. For this example I chose to map the intensity range, from white to black, to the gray distance range. In other words if a pixel is at the maximum gray distance of 0.5 we picture it white, whereas we picture it black if it is at the minimum gray distance of 0. We do this by applying the procedure DistGray(L, S) in Algorithm 2 to each pixel and we subsequently multiply the result by 2 and again multiply this result by 255. The first multiplication translates the obtained value such that it, for convenience, is in the range of $[0, 1]$, whereas the second multiplication translates this value such that it is in the range of $[0, 255]$ which evidently corresponds to an 8-bit intensity value.

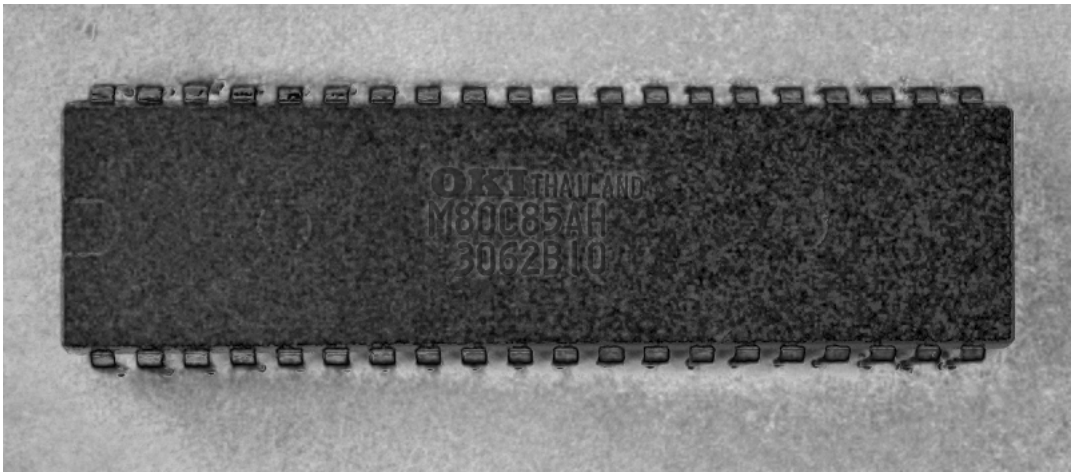


Figure 3.15: A visualization of what the computer "sees" when we consider a gray distance image.

A translation like this results in a picture as in Figure 3.15. Clearly, the IC is assigned a darker shade of gray than the surrounding background. The darker an area in an image like this, the more this area is to be considered gray colored.

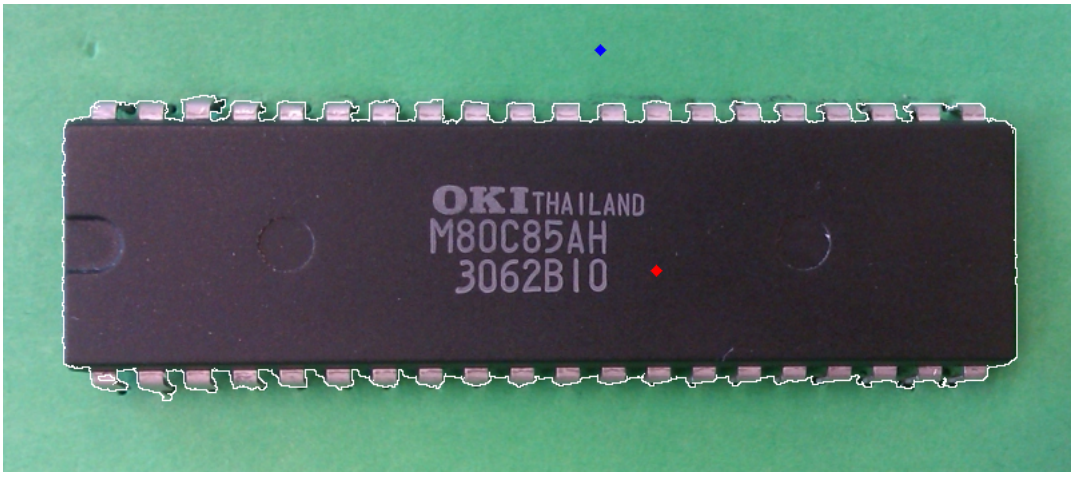


Figure 3.16: Example result of the modified SRG algorithm applied to the OKI chip.

Recall the results we had earlier for intensity images. If we compare Figure 3.9 and Figure 3.16 which both show the same picture, we observe a fair bit of difference. In contrast to the result obtained by the original algorithm, the adaptation we made seems to make it a little less sensitive when it comes to shaded areas. However, the most notable improvement concerns the selection of the pins of the IC.

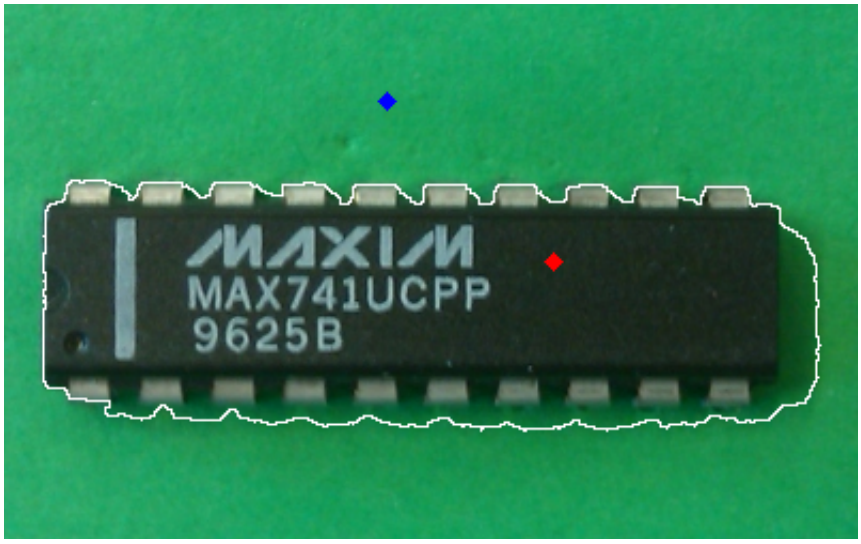


Figure 3.17: Example result of the modified SRG algorithm applied to the MAXIM chip.

In Figure 3.16 the pins are marked as part of the IC and the areas in between are marked as background whereas in Figure 3.9 the opposite was the case. This is desired behavior and it is due to the fact that our gray distance does not make distinction in a dark shade of gray or a light shade of gray.

After applying the adapted algorithm to the picture we have seen before in Figure 3.10, we obtain the result in Figure 3.17. Like in the previous result in Figure 3.16, this result also shows the improvement when it comes to marking the pins as part of the IC instead

of as marking them as part of the background. The most notable property of this result however is the fact that the algorithm remains prone to errors due to shaded areas. Of course, the algorithm's semi-interactive nature allows us to manually correct this by placing additional background seeds in the shaded areas. Still, an adaptation to the algorithm such that issues like this are taken care of automatically would be desirable since we want to limit the user's actions as much as possible and only include actions that are small and convenient for the user to perform.

4 Classification

Votes should be weighed not counted

Johann Friedrich Von Schiller

In the previous chapter we discussed a technique that allowed us to dissect input images such that what our areas of interest remain. This section will focus on the comparison of these areas of interest to what we already know. This concerns the machine learning and pattern recognition aspect of the machine vision discipline. From an AI perspective this is assumed the more interesting of the two sections (the previous section and this one) that deal with our proposed machine vision solution.

Automated classification of measurable phenomena, is one of the subdisciplines faced by data miners [10]. Data mining is the process of revealing previously unknown patterns in large data sets. In classification problems the dataset consists of entries that describe the known instances of a specific class. Upon obtaining new instance data, for which we do not yet know which class it belongs to, the aim of classification techniques is to accurately predict the corresponding class, based on the knowledge we already have.

For our classification solution I decided to exploit a nearest neighbor (NN) technique which is part of the so-called instance-based learning techniques. Unlike many other machine learning techniques commonly used for classification purposes, like support vector machines (SVM) or artificial neural networks (ANN) for instance, instance-based techniques do not require the construction of a model in order to generalize beyond the training samples. Instance-based techniques directly work with the training samples they have been provided with. Upon receiving a new query instance for which the class is unknown, they provide a method to compare the query instance with the training samples and a classification mechanism which provides the possibility to decide on the class label that should be tagged to the query instance. This way, instead of generalizing over the entire training space and evaluating each query in a global fashion, the query instances are evaluated locally each time [18].

One of the major advances of these techniques over these other learning-based techniques

is that there is no risk of overfitting a model. No model is generated. We speak of overfitting when a model fits the training data too well. Assuming most training data consists of noise as well as true data, in case of overfitting, noise ends up being modeled as well. Modeling noise of course has a negative effect on the quality of the classification results [4]. Obtaining a model for a dataset, such that it is in good balance and therefore not overfitted, is no trivial business however.

Another advance over learning-based classifiers is that the dataset to compare against is easily extended. Say we discover an IC for which we do not have data stored in our database yet. In this case, we simply take a few sample pictures, extract the features that are required by our solution, and add this newly obtained training data to the database. We will not have to recompute our model which, depending on the size of the dataset, quickly becomes a rather time consuming occupation. There is, however, a drawback to this approach as well. All computations are done at classification time. In this case classification itself could be computationally expensive and quickly become more expensive as the database grows in size.

The other technique often applied for the purpose of image classification, is the support vector machine technique which I briefly mentioned already. Snyder and Qi [24] state that SVMs appear to provide performance superior to other classification techniques when it comes to image classification. Boiman *et al.* [5] claims differently however. They devoted their work to defending nearest neighbor based image classification. They explain that both techniques have different properties and show that it depends on, how, and in which situation, each of the techniques is applied.

Most important reason for choosing a nearest neighbor approach is the simplicity [5] of the approach. The nearest neighbor approach is therefore easily grasped and implemented. Because of this the k-Nearest Neighbor algorithm (KNN), which we will discuss in a later section of this chapter, was easily adapted and fit in the overall solution such that it offered a positive contribution in pursuit of our desired solution.

I used the KNN algorithm for classification purposes that only require a few simple appearance features. The assumption was made that in order for our solution to be effective it should at the least be possible for our application to extract these features. The text, which could occasionally be read from an IC's imprint, is not among these features. Still, in case we are capable of reading this text, there is really no other property to an IC that allows for more accurate classification.

In an attempt to still exploit the textual features on an IC such that they contribute to the classification results in a useful manner, I designed what I named the Intentionally Biased Weighed Voting Classifier (IBWVC). The aim of this classifier is to combine results of multiple classifiers in such manner that one of them functions as the bias while the

remaining classifiers only exist to allow refinement of the results in a semi-interactive way.

First, we will briefly analyze and discuss the IC features which are considered usable, and discuss the extraction of these features. After this, I explain the KNN algorithm and how I modified it such that it effectively contributes to the classification solution. Next, I explain how I took an existing and freely available optical character recognition (OCR) solution, treated it as a black box, and used the results to perform classification solely based on textual features. In the remaining paragraphs I explain how I combined the KNN and the OCR results and translate this to the more abstract solution, I named the IBWVC. Throughout the course of this section I discuss some mentionable results that were obtained through experiments.

4.1 Feature Selection

If we compare two integrated circuits ourselves and try to discover whether or not they are of the same type, it would probably not take us very long. At least, not in most cases. Among the properties that allow us to do so quickly, are the general appearance of the integrated circuits under consideration and their imprint. Figure 4.1 shows a schematic display of an IC. In fact, it shows a schematic display of one of the sample ICs which I used for my experiments.

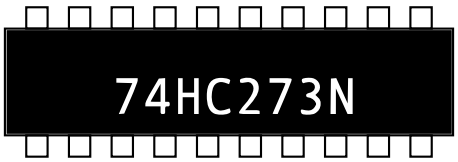


Figure 4.1: A schematic top-view of an example integrated circuit.

Among the general appearance properties are the amount of visible pins, the absolute size, the width-height ratio, etc. Some of these properties are easier effectively exploited than others.

The pin count for instance, depending on the amount of pins and the amount of spacing between the pins, requires careful counting. One of the main issues with this feature is that lots of ICs of different classes count a common amount of pins. Because of this, even if we are able to count the pins accurately, this feature is likely to only help us shrink our search space a little. Another thing that complicates effectively exploiting this feature concerns the visibility of pins. Namely, before we could even start counting pins, we do also need to accurately assure pins are visible at all. Especially for the more modern ICs, pins cannot always be counted without flipping the IC over.

According to Boiman *et al.* [5] images are often described by a collection of local image descriptors. Examples of these descriptors are the so-called SIFT [16] keys. SIFT, which

is the abbreviation for Scale Invariant Feature Transform, is a method for image feature generation. It transforms images into large collections of local feature vectors. These feature vectors are invariant to image translation, scaling and rotation. Apart from that, they are partially invariant to illumination changes and affine or 3D projection.

In case of our solution we do not need a computer to generate features. This is especially useful when one is interested in classifying images that belong to an arbitrary amount of very diverse categories. For instance, one image might contain a house, another might contain a collection of flowers, another image might contain something again entirely different, etc. There exist example data sets that provide a diverse collection of images and serve as image classification benchmark sets. Among these data sets are the Caltech-101 and the Caltech-256 set for instance, which are also used for benchmark purposes in [5].

Because our database contains only images of ICs I was able to manually look for features that are, by assumption, commonly available among all ICs. We assume that

- all ICs are square-shaped;
- all ICs contain an imprint which, if readable, is a property that is fully decisive in what kind of IC it concerns;
- imprints on ICs are placed by a machine and therefore assumed always to be equally positioned with respect to where the imprints for the particular class of ICs should be positioned;
- manufacturing often results in some form of damage to the imprint, which often makes the imprint (partially) unreadable.

This collection of assumptions and the small set of test samples that I had at my disposal, allowed me to manually compare ICs and determine a small set of features which can be applied to describe the appearance of an IC. Unlike SIFT features, the chosen features are not invariant to image translation or rotation. They are however, like SIFT features, invariant to scaling. In a practical setting this means we do not have to take the distance, at which we take a picture from, in consideration. It does require us to keep image translation and rotation into account however.

Figure 4.2 shows a schematic image of the features I decided to extract. Basically it shows an image in which the red frames are the parts we would like to be able to reconstruct given that the width and height of the image are specified. The outer red frame tells us about the width and height of the IC. More specifically, since we aim to obtain scaling invariant features, it tells us about the width-height ratio. The inner red frame tells us about the imprint area and how this relates to the IC as a whole, in terms of appearance.

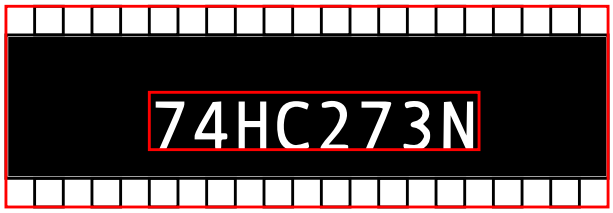


Figure 4.2: A top-view of an example integrated circuit in which the marked areas denote the areas we look to represent in a set of features.

It provides us the information on where the imprint area is located, what the imprint's width-height ratio is and what the imprint's size is relative to the overall size of the IC as a whole.

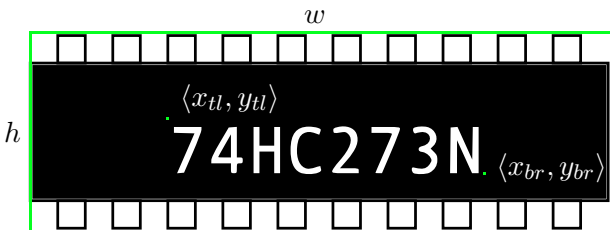


Figure 4.3: A top-view of an example integrated circuit in which the set of features which we look to extract are marked.

The green marks in Figure 4.3 display the parameters which we need to estimate/extract from the input image in order to enable us to reconstruct an image like the image in Figure 4.2. Now, lets define w as the width of the query image and h as the height of the query image. These are the first two parameters which we require to enable us to reconstruct the red lines in Figure 4.2. The other points we are interested in are the coordinates for two of the corner points of the imprint area. We are interested in obtaining the top-left $\langle x_{tl}, y_{tl} \rangle$ coordinates and the bottom-right $\langle x_{br}, y_{br} \rangle$ coordinates of this area, since these two points allow use to define another width and height parameter for the imprint area as well. At the same time the top-left coordinates allow us to determine the relative positioning of the imprint area. The values for these coordinates are relative to the values for the coordinates of the top-left coordinates of the outer frame. These coordinates are considered $\langle 0, 0 \rangle$.

This leads to a total of five features which are further described in Table 4.1. In Eqn. (4.1) you find how each of them is computed.

4.2 Feature Extraction

Now that we have defined the features of interest we are left with a few concerns. Just how do we extract these features from the input image? The w and h parameter are simply defined as the width and height of the input image. This information is obtained as a result of the segmentation part, which we performed earlier. Basically all we are left with

Feature	Description
whratio	A single real number which denotes the amount of pixels for each horizontal pixel of the entire IC area and is therefore defined as the IC's width-height ratio.
impwhratio	A single real number which denotes the amount of pixels for each horizontal pixel of the imprint area and is therefore defined as the imprint area's width-height ratio.
imprelsize	A single real number in the range of $[0, 1]$ which defines the size of the imprint area relative to the full IC area. Basically it defines the percentage of the IC that is used as imprint area.
imprelx	A single real number which denotes the relative horizontal distance for the imprint top-left corner to the IC area's left edge.
imprely	A single real number which denotes the relative vertical distance for the imprint top-left corner to the IC area's top edge.

Table 4.1: A brief overview of the exploited appearance features.

is extracting the two coordinates $\langle x_{lt}, y_{lt} \rangle$, and $\langle x_{br}, y_{br} \rangle$.

For my proof of concept I relied on manual extraction of these coordinates of interest. In contrast to the segmentation part and the classification part, as we will see in a few sections, this part would preferably take place in an automatic manner. In other words, in order for our solution to cover each aspect of the problem domain we should also aim to find a way to automatically extract text from our input images. Instead, I put emphasis on the semi-interactive aspects of the solution and details of automatic feature extraction are beyond the scope of this thesis.

$$\begin{aligned} whratio &= \frac{h}{w} & imprelx &= \frac{x_{tl}}{w} \\ impwhratio &= \frac{y_{br} - y_{tl}}{x_{br} - x_{tl}} & imprely &= \frac{y_{tl}}{h} \\ imprelsize &= \frac{(x_{br} - x_{tl}) \cdot (y_{br} - y_{tl})}{w \cdot h} \end{aligned} \tag{4.1}$$

Publications on automatic text extraction from images exist. Neumann and Matas (2011) [20] for instance, published an article on text localization in real-world images. As an example application they mention image databases filled with real-world images, Google Images for instance. Textual information in these images can be used for annotation purposes, which can then be exploited for search purposes.

Another fairly recent publication on the same topic by Pan *et al.* [21] exists in which they

mention some of the difficulties certain solutions face and explain the hybrid approach they took in order to overcome these difficulties. Like the publication by Neumann and Matas, they used the ICDAR 2003 image dataset to evaluate their algorithm. Both solutions show promising results when applied to this dataset.

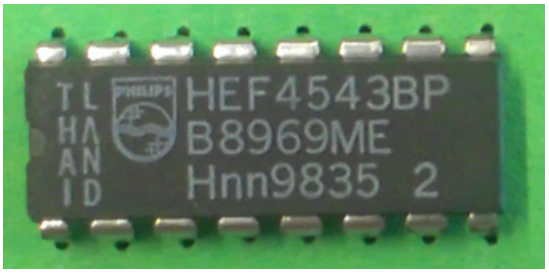


Figure 4.4: An integrated circuit for which the imprint, apart from just text, also contains a logo.

A slightly more exotic publication by Jamil *et al.* [13] describes an edge-based approach for localization of Urdu text in video. They claim the literature to be rich in describing text detection solutions for Latin or Chinese alphabets yet not for Urdu text.

Like Jamil *et al.* we face a problem slightly different than a conventional text localization problem. In our case we are especially interested in localizing the imprint area. In Figure 4.4 for instance, the imprint area also consists of the image of a logo. A suitable solution should not have trouble with marking a logo, as is the case here, as part of the imprint area as well.

Figure 4.5 shows a picture of an IC in which the imprint not only contains a logo but in which the text has also been marked. A mark like this is not entirely uncommon. It for instance has been used to distinguish between ICs that have already been programmed and ICs that are yet to be programmed. The solution should, like in case of Figure 4.4, have no problem localizing the imprint area because of a flaw like this. Difficulties like this could very well allow for a separate case study, which is entirely devoted to accurately determining the imprint's location.

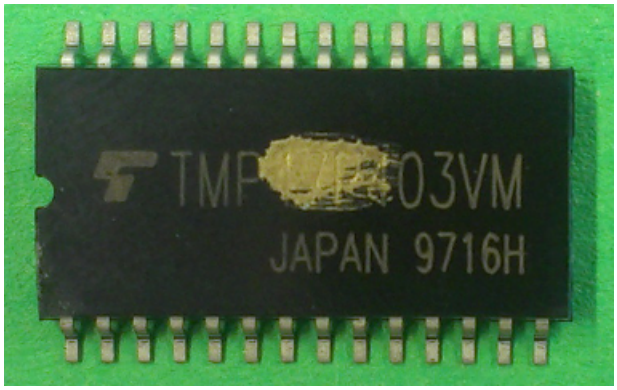


Figure 4.5: An example of a case in which the assumption of always having an imprint which is not always completely readable holds.

The feature that remains of course is the text which we can find in the imprint areas of pretty much all ICs. We make a distinction between the appearance features which we described in this section and this textual information. We do so since we assume the text to be not always readable nor does it seem trivial to combine textual information with these five real valued features as should become clear in the remaining sections of this chapter.

4.3 Nearest Neighbor Appearance Classifier

The k-Nearest Neighbor (KNN) [18] algorithm is an instance-based learning algorithm that is used for classification purposes. I will use this paragraph to go over this algorithm and explain how I adopted it such that it serves the solution well. First, I will describe the original algorithm and mention some of the difficulties it contains. Afterwards, I will elaborate on how I applied the algorithm such that it suits our needs in terms of matching integrated circuits on the basis of the appearance properties, which we discussed in the previous paragraphs.

Properties

Unlike learning methods, instance-based methods do not require construction of a model in order for them to be able to generalize beyond the training samples. In other words, there is no risk of overfitting a model since we do not construct any [5]. Instead, generalization beyond the training samples is postponed until a new query instance is given. The basic principle is to directly relate a query instance to the data of the training instances. Because the generalization beyond the training samples is postponed until a new query instance is obtained, these sorts of methods are sometimes also referred to as lazy learning methods [18].

Apart from not taking the risk of overfitting a model, these type of methods hold a few other notable properties. One of them is in fact directly related to this property of not approximating a model. This creates the possibility to instantly append the training data with data of newly obtained instances. In case of learning methods we would, after appending a new instance to the collection of training instances, have to recompute our model as well, in order for the newly added information to be taken into account for the next classification. In other words, it allows for cheap, in terms of time required, extension of the database on which we perform our queries, when we obtain new IC information. This makes that we can easily extend our database with information of newly observed ICs.

Another mentionable disadvantageous property is directly related to the aforementioned. Namely, the fact that we do not estimate a model from the training data does cause us to do all required computations at query time. As the set of training samples becomes larger queries become more expensive in terms of time consumed.

To overcome an excessive amount of computation time, methods exist in which tree-based search structures [4] are computed from the dataset that allow nearest neighbors to be found or approximated in an efficient manner. It should be clear that construction of these search structures takes away the advantage over learning-based techniques of not having to compute a model. Since, for each change to the training data we would then have to recompute the search structure.

Algorithm Basics

Nearest neighbor approaches assume all instances can be represented as points in a Euclidean space. Evidently, this assumption suits the features which we defined in the previous paragraphs. It makes that each instance x in the KNN algorithm can be described by a feature vector

$$\langle a_1, a_2, \ldots, a_n \rangle \tag{4.2}$$

in which case each a_i corresponds to a single real-valued feature. For our classification solution we have that $n = 5$ and each vector entry a_i is replaced by one of the features we previously defined.

whratio	**impwhratio**	**imprelsize**	**imprelx**	**imprely**	**class**
0.3672055427	0.5296052632	0.1777274246	0.3360277136	0.2452830189	2
0.8577405858	0.4611650485	0.3994285131	0.0167364017	0.2682926829	9
0.6453333333	0.2979452055	0.2799338843	0.1173333333	0.326446281	5
0.7689530686	0.7329545455	0.3848070372	0.2129963899	0.1924882629	7
0.631043257	0.2783171521	0.2726545186	0.1221374046	0.3225806452	4
...	...	...	...	...	..

Table 4.2: An arbitrary set of sample instances used for experimentation purposes.

A training set T can be defined in a straightforward manner. This training set really functions as the database which we compare query instances against. This database could well be a simple table structure. Let us define an entry in T as a list, more specifically a pair $\langle x, c \rangle$, which itself contains a feature vector x and a class label c which could be a numeric representation of a certain class. In my experimental setup for instance, the numeric class label c refers to a specific IC and is used to retrieve class-specific properties like the imprint, an image, and a datasheet that contains specifications of the IC. Table

4.2 shows an arbitrary sample set from the database that I constructed and used for experimentation purposes.

$$\delta(x, y) = \sqrt{\sum_{i=1}^{n}(x(a_i) - y(a_i))^2} \tag{4.3}$$

The Euclidean distance, for which the definition is given in Eqn. (4.3) [18], is used to compare two feature vectors x and y in terms of equality. The distance corresponds to the unique shortest distance from one point in a Euclidean space to another. In other words it is the distance we obtain if we were to use a ruler to determine the distance between two points.

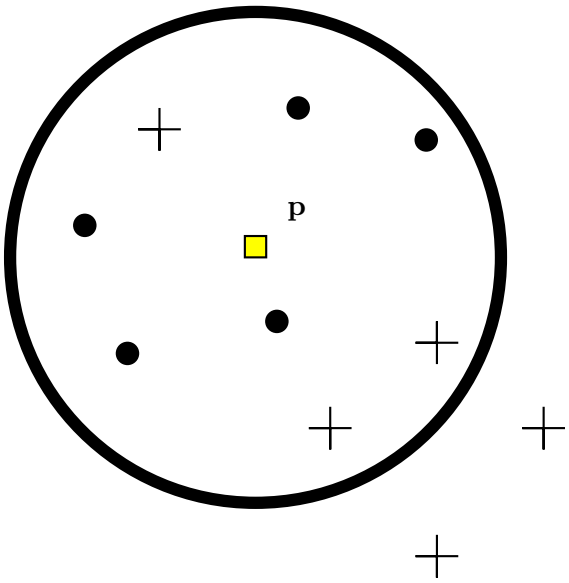


Figure 4.6: An example two-dimensional Euclidean space in which we look to classify the given query point p. The shapes, the dots and the pluses, indicate the classes to which the pictured training instances belong. The surrounding circle indicates the value of K that has been selected in order to classify p. It shows K has been set to eight and a majority of the top-K instances belongs to the dots class. According to the majority-rule, p should be considered part of the dots class.

Now that we defined instances and how we determine the distance between them, the flow of the k-Nearest Neighbor algorithm is rather straightforward. The basics are pictured in Figure 4.6. In this figure a simple example two-dimensional Euclidean space is shown. In this space, training instances are visualized as either part of the "dot" class or part of the "plus" class. The yellow square, point p, denotes the query instance of which we are to determine whether it belongs to the dots or the pluses. Finally, the circle that surrounds most of the points illustrates how K is used in the algorithm. K determines the amount of nearest neighbors we are to take in consideration for classification purposes. According to the majority class membership [4] in our example, p would be classified as part of the dots since the dot instances within the top-K selection count two more instances than the plus instances do.

Curse Of Dimensionality

As we have just seen, in essence the algorithm is rather simple and intuitive. Determining the proper value for K however and exploiting a proper set of features is no trivial business however. One of these difficulties which concerns the decision on a suiting set of features, is the the curse of dimensionality. Mitchell [18] sketches a situation in which we use a total of 20 features to describe the instances. If in a particular case only two of these features happen to be relevant to determine the class of a query instance, these features are likely to be dominated by the remaining 18 features. For instance, if we only were to consider the two relevant features, the algorithm might have classified our query instance correctly as member of class A. However, the 18 remaining irrelevant features might still cause our query instance to be falsely classified as member of class B since each feature is considered equally important.

Mitchell [18] proposes a solution to this problem in which each individual feature is assigned a weight such that less important features contribute less to the measured distance between instances. Clearly this creates the problem of determining suiting weights.

For my set of features I did not adopt this solution since the used feature set is assumed small (only 5 distinct features are used), and, more importantly, there is plenty of reason to assume dominance of a collection of irrelevant features over a remaining set of relevant features cannot occur.

Figure 4.7: Example query input for which the marked areas show the information that is extracted. The same type of information is stored in and compared by the previously specified set of features.

Lets elaborate on this a bit. In the previous paragraphs I mentioned that the selected features allow for reproducing the appearance of an IC, in terms of size, imprint location and imprint size. To be able to do so we require experimental feature data. By means of example, consider the feature data which I listed in Table 4.3. This data was obtained of instances that are of the same class as the instance in Figure 4.7.

For the purpose of reproducing we could take the data of any of the entries in Table 4.3 since each entry belongs to the same class and the data in each column only varies slightly among entries. For example, lets exploit the data in the first entry of this table. Given the data in this entry and the knowledge that Figure 4.7 has a width of 713 and a height of 226, we were able to construct the image in Figure 4.8. As these figures show, the selected

whratio	impwhratio	imprelsize	imprelx	imprely	class
0.3142857143	0.5142857143	0.0871436256	0.3846153846	0.3006993007	1
0.3163265306	0.5245098039	0.0887035818	0.3854875283	0.2759856631	1
0.2966915688	0.5138888889	0.0920433344	0.3842049093	0.2913669065	1
0.3148558758	0.5120772947	0.0856547266	0.3880266075	0.2887323944	1
0.3074468085	0.5161290323	0.0894647721	0.3808510638	0.3044982699	1

Table 4.3: Experimental feature data for instances that are member of class 1.

features allow us to accurately approximate that, which was originally marked and what made us obtain the data in the first entry of the table.

Evidently, we strictly need all of the feature parameters in order to be capable of reproducing the appearance from data. Therefore, all of our parameters are considered equally important and no relevant parameters will ever be dominated by irrelevant parameters. In fact, if we were to change one or more of the feature values we would simply hold a description of a different appearance.

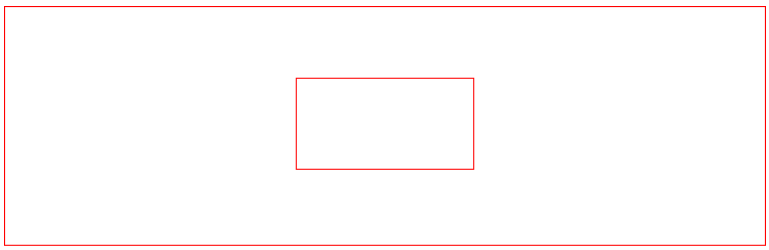


Figure 4.8: A result of visually reproducing appearance data given that the width and height are specified.

An example mistake we could have made concerns the inclusion of the text in the imprint area as a feature in our KNN search. As I mentioned before, if we obtain a perfect match in terms of imprint, we would then no longer need any of the remaining information in order for us to reliably classify the query instance. For example, if we have knowledge of a class X of ICs, which reads "74HC273N" and an OCR solution would read exactly the same text from a given query instance p, we can then confidently state that our query instance belongs to class X without having to consider any of the remaining features. In a nearest neighbor approach a setting like this would however be vulnerable to dominance of irrelevant parameters. The remaining five appearance features would after all be considered equally important.

Assume the scenario in which we have knowledge of class X and we also have knowledge of a class Y. Instances of class X and Y only show a slight, yet significant, difference in terms of text whereas in terms of remaining geometric properties they are very similar. Again assume that we obtain a perfect match for class X with respect to the text we found on the given query instance p, then we could again confidently state that p belongs to X. In a nearest neighbor approach it is however very well possible that p is (slightly) closer related to class Y with respect to the remaining features. In this case the textual

feature is likely to be dominated by the remaining features which could therefore result in erroneous classification.

This leaves us with the fact that until this point we strained from exploiting the textual feature which could be extracted from most ICs. It is an important feature, if not the most important feature. The remaining two paragraphs will therefore cover a proposal which aims to create the possibility to obtain a positive contribution of this feature to the classification result.

Application

The remaining issues relate to the value for K and the actual method of classification. Until this point we supposed a simple majority rule provides for the actual method of classification. Like in a previous example, if our query instance p results in an ordering of class labels, ranging from closest to furthest, in which class label A happens to occur the most, p is simply classified as a member of class A. In our case this approach does not suffice since we look to find an ordering of classes for which the first one in the ordering is to be the most likely suiting class, the second one the second most likely suiting class, etc. This is due to the practical aim of the application. As a list like this is ultimately what we look to provide the end-user with.

Of course a straightforward adaptation could be that we rank the classes according to how often they occur. A simple example shows that this should be considered rather unsophisticated however. Assume the following example ordering, for which $K = 6$ and the letters A, B and C are used to denote a class

$$A \succ A \succ C \succ B \succ B \succ B \tag{4.4}$$

According to this ordering and the aforementioned adaptation of the classification method we would count class A twice, class B three times and class C only once which results in the following classification

$$B \succ A \succ C \tag{4.5}$$

Evidently none of these majority based classification rules should be considered fair since the distance from the neighboring instances to the query instance is never taken into consideration. It could for instance very well be that the first two neighbors are a lot closer to the query instance than the remaining neighbors are. To overcome this, Mitchell [18] proposes a refinement in which the contribution of each of the neighbors is weighed according to the inverse square of its distance to the query instance.

For example, assume the ordering of distances to an imaginary query instance

$$0.04 \succ 0.06 \succ 0.14 \succ 0.20 \succ 0.20 \succ 0.22 \tag{4.6}$$

corresponds to this ordering of classes as in Eqn. (4.4). According to this distance-weighed adaptation we would have that the score for A is

$$\frac{1}{0.04^2} + \frac{1}{0.06^2} \approx 902 \tag{4.7}$$

Accordingly we do the same for the remaining classes that occur in the ordering of the K nearest neighbors. For class B we approximate a score of

$$\frac{1}{0.20^2} + \frac{1}{0.20^2} + \frac{1}{0.22^2} \approx 70 \tag{4.8}$$

For class C we would evidently perform the same computation, after which we order the classes in a descending manner according to the scores we just computed. This results in an ordering where

$$A \succ B \succ C \tag{4.9}$$

Mitchell [18] states that when we apply this distance-weighed mechanism there is really no harm in considering the entire space for classification purposes. In other words we could select the value for K such that it is equal to the size of the dataset. Because of this property I assumed it could at most be slightly harmful, if we were to select a value for K such that we guarantee a top-V class ordering like we obtained a top-3 class ordering in Eqn. (4.9).

In my experiments I assured that the dataset contained a total of five instances per class. Since I constructed the dataset myself, built of ICs I arbitrarily collected of whatever I could easily get my hands on, I could assure that of each class there was a total of five instances in the dataset. For experimental purposes the amount of five instances per class is simply assumed to be a suiting amount. The aim of these experiments was to provide the end-user with a top-5 class ordering and therefore I determined that K should equal 25 since this guarantees us to obtain at least a top-V class ordering for which $V = 5$.

Observations

To conclude this paragraph I leave you with a few visualizations of the small sample dataset I constructed for my experiments. In the figures 4.9, 4.10, 4.11 and 4.12 I show samples of how instances in the dataset relate to each other over different pairs of dimensions. Each of the marks in these plots correspond to a single instance of a single class. The color of these marks determine to which class an instance belongs. For instance, marks that are colored blue all correspond to instances of the same class.

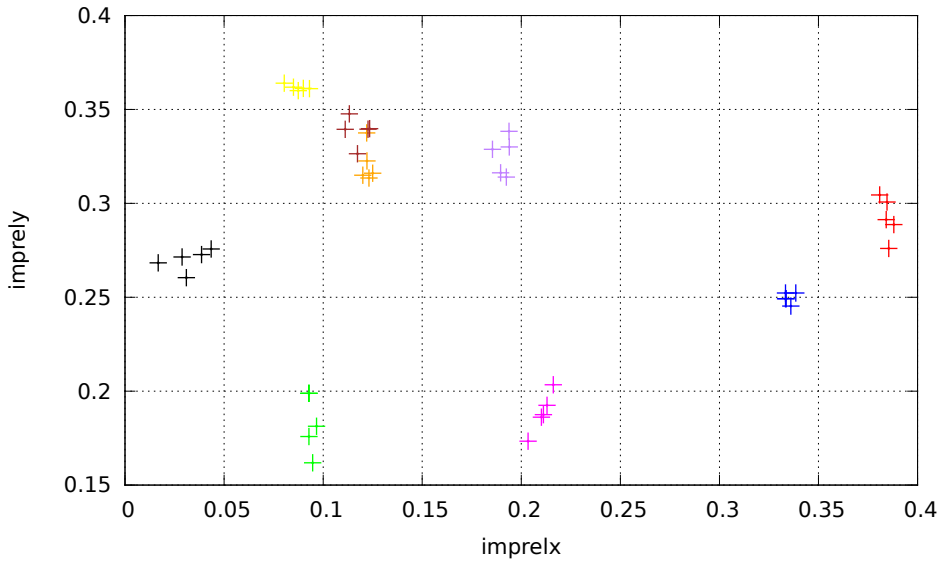


Figure 4.9: A plot of our experimental set of training instances that shows the imprint's relative x-coordinate over the x-axis and the imprint's relative y-coordinate over the y-axis.

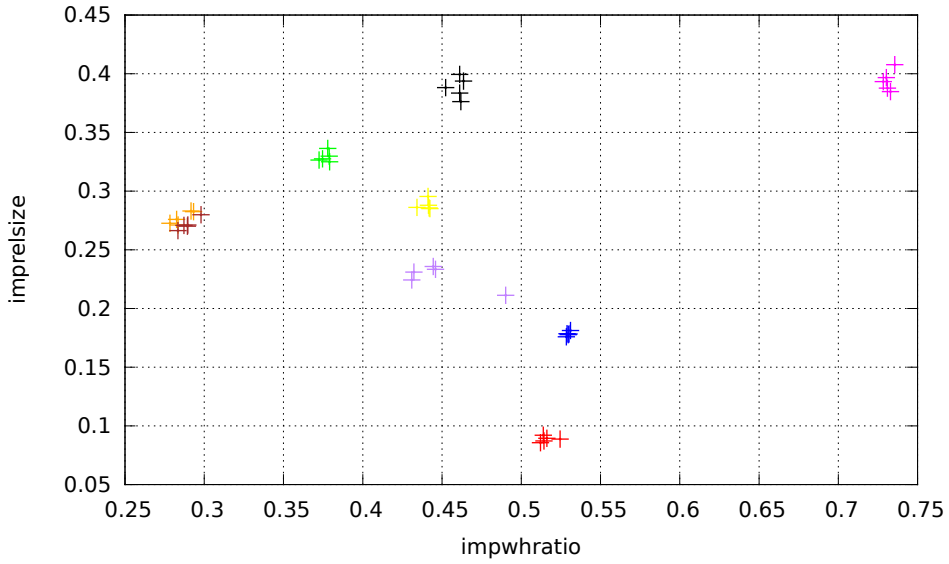


Figure 4.10: A plot of our experimental set of training instances that shows the imprint's width-height ratio over the x-axis and the imprint's size relative to the overall size of the IC over the y-axis.

These plots allow us to raise a few expectations with respect to how well the discussed KNN algorithm performs on this dataset. Its evident that, in most cases, instances of the same class are well clustered and distances between clusters are rather substantial, which should lead to accurate classification that is entirely based on the aforementioned five appearance features. This holds for all but two particular classes. The instances for which the classes are denoted by the color brown and orange overlap in all of the plots. This makes it difficult to accurately classify instances solely based on their appearance. In cases like this the text we could extract of an instance's imprint could help improve the quality of the classification results.

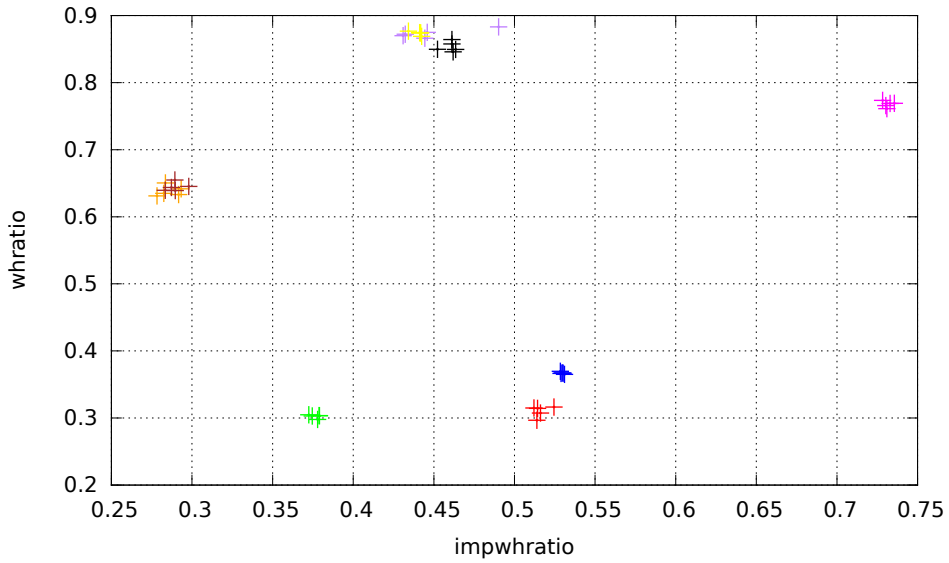


Figure 4.11: A plot of our experimental set of training instances that shows the imprint's size relative to the overall size of the IC over the x-axis and the overall width-height ratio over the y-axis.

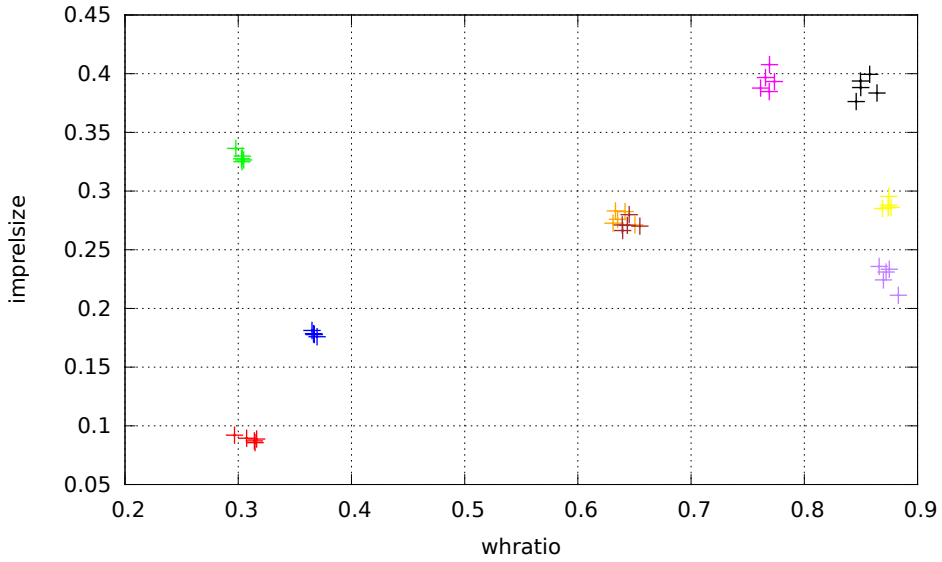


Figure 4.12: A plot of our experimental set of training instances that shows the overall width-height ratio over the x-axis and the imprint's size relative to the overall size over the y-axis.

4.4 A Levenshtein Classifier

Another way of determining the class of an IC is by reading its imprint. In fact, if available, this may be assumed to be the most reliable method of classification. If for instance, we have knowledge of a specific class of ICs for which the imprint reads 74HC273N and we encounter a query instance from which we correctly extract this same imprint we can state with certainty that this query instance is of this particular class regardless of any other features we might extract.

Reason we are however interested in the other features, as discussed in the previous paragraphs, is simple. It is not uncommon for the imprint to be hardly readable. For the solution which is described in this thesis I assumed to be able to find the imprint's area in all cases while at the same time I assumed the worst-case scenario in which the imprint itself is (partly) unreadable.

In the remainder of this section I describe the optical character recognition solution that I adopted to extract text from images and the distance measure which I used to determine the amount of similarity it shows with imprints of ICs which we already have knowledge of. Finally I briefly explain a straightforward approach to how this can be turned into a classifier.

Optical Character Recognition

The concept of optical character recognition (OCR) is rather self-explanatory. The goal of an OCR solution is to take image data as input and extract from it text that can be found in it, as is described in Figure 4.13. Whereas the concept is self-explanatory and straightforward, this does not hold for quality implementations. In this thesis I do not cover details of OCR techniques. Over the years many OCR-related research has been conducted of which I made use in terms of an existing and freely available solution.

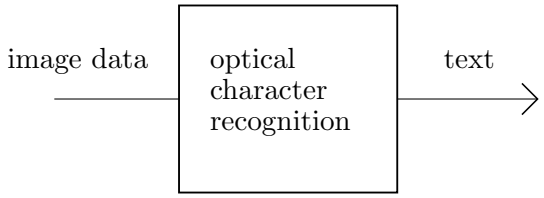


Figure 4.13: The OCR process as a black box process which we feed image data as input and obtain the extracted text as output.

I used Google's open source OCR engine named tesseract-ocr which has been around since the project was started in 1985 at HP Labs and has evolved since. It is claimed to probably be the most accurate open source OCR engine available. I used it as an out of the box solution, without any adaptations.

Figure 4.14: Example of an image which would typically serve as input to the tesseract-ocr solution.

Because it results in performance increase in terms of accuracy, the tesseract-ocr engine is provided only that part of the image that should contain text. This is no problem since

we already assume to be able to find the imprint area.

Example input is shown in Figure 4.14. Application of the tesseract-ocr engine to this sample input, shows the output

```
OKITHAILAND
M80C85AH
30623 I U
```

from which we may conclude the tesseract-ocr solution only made an acceptable small amount of mistakes for this particular example. More small experiments have been conducted and have often shown similar acceptable results. Next, we need a text comparison measure such that we can use output of this kind for classification purposes.

Levenshtein Distance

To determine a measure of similarity between two strings, different string comparator metrics exist, of which the Levenshtein distance [14] and the Jaro-Winkler distance [25] are examples.

For my proof of concept I used the Levenshtein distance which is a well-known and commonly used string comparator. The distance obtained by this comparator is often also referred to as the (minimum) edit distance. Which is due to the fact that the comparator takes two strings as input and returns the minimum amount of edits required to get from one of the strings to the other, with the restriction that only single character insertions, deletions and substitutions are allowed.

For example, if we take the extracted output from the example in the previous paragraph and we compute the Levenshtein distance from this string to the string which we should have read from the IC in Figure 4.14, we obtain a distance of 4. Clearly this is due to the fact that the extracted text contains two spaces too many and two wrongly extracted characters. Therefore, to get from one string to the other we need at least two deletions/insertions and two substitutions.

Using this distance measure we can obtain a working classifier in a straightforward manner which, like the nearest neighbor classifier, results in an ordering over the known classes. We simply extract the text of our query instance using the aforementioned OCR solution and compute the Levenshtein distance between this text and each of the class imprints which we know of. Finally we are left to ordering the known classes according to these computed distances, which leaves us with a similar classification result as we were left with by the nearest neighbor classifier.

One mentionable difference, in terms of classification, concerns that with the nearest neighbor classifier we compared an extracted set of features with sets of features of other known instances of a specific class. This time we compare a feature of the query instance directly to a feature of known classes.

4.5 Intentionally Biased Weighed Voting Classifier

Until this point we have discussed two distinct methods which we can exploit to classify images of ICs. In section 4.3 we saw that a nearest neighbor approach can effectively be used to approximate the class of an IC solely based on a small set of geometric appearance features. In section 4.4 I showed that we can exploit Google's tesseract-ocr solution, treat it as a black box and apply the Levenshtein distance measure to construct another class ordering based on the automatically extracted imprint of the query instance and the imprints of classes we know of.

This leaves us with the lack of a suiting method that combines the results of both the nearest neighbor classifier and the Levenshtein classifier, such that the both offer a positive contribution to the final classification. Actually, such that the results obtained of the Levenshtein classifier refine the nearest neighbor results. Difficulty lies in the expected quality of each of both classifiers. By assumption, the results obtained of the nearest neighbor classifier are more reliable than the other results. This section covers my solution to this problem, which I named Intentionally Biased Weighed Voting Classifier (IBWVC).

Combining Multiple Classifiers

One of the issues relates to the fact that we are left with two class orderings that may, and are likely to, differ from each other. Evidently there is no issue if both obtained class orderings are equal.

$$c_1 \succ c_2 \succ c_3 \succ c_4 \succ c_5 \tag{4.10}$$

Assume the ordering in Eqn. (4.10) is the result obtained by applying the nearest neighbor classifier to an arbitrary sample database in which each element c_i denotes a class. The aim of the IBWVC classifier is to combine this result with the result in Eqn. (4.11) which, by means of example, is a result obtained by applying the Levenshtein classifier to the same sample database.

$$c_1 \succ c_4 \succ c_5 \succ c_3 \succ c_2 \tag{4.11}$$

We face a situation similar to decision problems in multi-agent systems in which preferences of multiple distinct agents are combined such that the result is a fair reflection of the preferences for the population of agents as a whole. The book by Yoav Shoham and

Kevin Leyton-Brown [23] explains how voting methods could be of use in situations like this. These methods are similar to methods we are already familiar with from political elections. Examples of voting methods include plurality voting, approval voting and Borda voting.

Among other methods, Ho *et al.* [11] propose voting methods for ranking reordering in a multiple classifier system. They state that an ideal combination of classification results should take advantage of the strengths of the individual classifiers while avoiding their weaknesses. They consider a multiple classifier system a success when it results in a new ordering for which the true class is ranked as close to the top as possible. Additionally I assume it can be considered a success, only if the true class ranks at least equally high as it does in the result that was obtained by the best individual classifier. The experiments they conducted showed substantial improvement in classification accuracy.

Borda Count

For my proof of concept I adopted the Borda count [23]. Our first requirement is a set of participants. Each participant is required to submit its vote in the form of an ordering of preference over the candidates that are eligible for election. Recall the orderings in Eqn. (4.10) and Eqn. (4.11). A similar ordering is exactly what needs to be submitted by each Borda count participant. In our case this makes the IC classes in these same sample orderings the eligible candidates.

$$P = \{f_{p_1}, \ldots, f_{p_n}\} \tag{4.12}$$

For each participant there exists a function $f_{p_i} \in P$, as in Eqn. (4.12). Each of these functions result in an ordering over the eligible candidates. A candidate then obtains a score of $n - i$, where n denotes the candidate count and $i \in [1, n]$ which corresponds with the candidate's position in the ordering, such that $i = 1$ denotes the most preferred candidate and $i = n$ corresponds with the least preferred candidate. The total score for a candidate is obtained by summing over the scores it obtains in the orderings that were obtained from each participant. Finally the combined ordering of candidates is obtained by ordering the candidates according to these summed scores.

	c_1	c_2	c_3	c_4	c_5
f_{p_1}	4	3	2	1	0
f_{p_2}	4	0	1	3	2
	8	**3**	**3**	**4**	**2**

Table 4.4: Sample application of the Borda count.

In Table 4.4 we find a sample application of the Borda count on the two previously mentioned orderings, in which f_{p_1} corresponds with Eqn. (4.10) and f_{p_2} with Eqn. (4.11). The numbers in bold show the summed score for each class which makes this sample application of the Borda count result in the ordering in Eqn. (4.13). Note that c_2 and c_3 obtained the same score and we could therefore obtain another result which is equal to this one with the exception that we swap c_2 and c_3. For my proof of concept I assumed this issue to be of no substantial concern and arbitrarily selected a solution if more than a single one existed.

$$c_1 \succ c_4 \succ c_2 \succ c_3 \succ c_5 \tag{4.13}$$

Weighed Borda Count

In political elections it is at the very least extremely hard, if at all possible, to define preferences as either correct or incorrect. In the classification problem which we face, validity of a classifier's produced result is easily observed in a manual manner. In our case we are especially interested in the validity of the Levenshtein classifier which we assumed to possibly be inaccurate due to the possibility of noise in the imprint. We assume that the OCR solution, which is used by the Levenshtein classifier as one of its input sources, provides us with feedback on the text it extracted from the input. As we are quite capable of quick and manually judging the quality of the extracted text, we are also capable of quickly and roughly estimating the degree to which the Levenshtein classifier is expected to positively contribute to the overall classification results.

	c_1	c_2	c_3	c_4	c_5
f_{p_1}	$4 \cdot w_{f_{p_1}}$	$3 \cdot w_{f_{p_1}}$	$2 \cdot w_{f_{p_1}}$	$1 \cdot w_{f_{p_1}}$	$0 \cdot w_{f_{p_1}}$
f_{p_2}	$4 \cdot w_{f_{p_2}}$	$0 \cdot w_{f_{p_2}}$	$1 \cdot w_{f_{p_2}}$	$3 \cdot w_{f_{p_2}}$	$2 \cdot w_{f_{p_2}}$
	400	**75**	**125**	**250**	**150**

Table 4.5: Sample result obtained by assigning a weight to each participant f_{p_i}. For this example $w_{f_{p_1}} = 25$ and $w_{f_{p_2}} = 75$.

Until now we considered each of both votes as equally usable in our pursuit of an accurate classification result. Depending on the accuracy of our Levenshtein classifier this could very well result in undesirable behavior. In fact, instead of improving on the results of the nearest neighbor classifier, it is likely to harm the results. In order to make the contribution of each classifier to the overall result more flexible I mapped a weight $w_{f_{p_x}}$ to each participant function f_{p_x}.

In Table 4.5 I show an example application of this weighed adaptation of the Borda count on the same data as found in Table 4.4 in which the Levenshtein classifier contributes for 75% to the combined ordering and the nearest neighbor classifier only contributes for 25%.

The resulting ordering is shown in Eqn. (4.14).

$$c_1 \succ c_4 \succ c_5 \succ c_3 \succ c_2 \tag{4.14}$$

Automatically determining proper weights is no trivial business. In order to do so we would need a measure of quality for each classifier. Upon determining the quality of each classifier we could accordingly adjust each classifier's corresponding weight. The contribution of each classifier to the combined result is computed by Eqn. (4.15). A notable difficulty concerning the determination of the quality of the Levenshtein classifier is that, even if the classifier's most preferred candidate is at a substantial distance from what was read by the OCR solution, this does not necessarily infer faulty classification.

$$\frac{w_{f_{px}}}{\sum_{f_{p_i} \in P} w_{f_{p_i}}} \tag{4.15}$$

For my experiments I used manual selection of the weights. In order to be capable of approximating the quality of each classifier, each of them should provide the end-user with useful feedback on the process that ultimately led to the obtained results. For instance, the Levenshtein classifier needs to provide the end-user with the text that was read by the OCR solution and which was used as input to determine an ordering of classes. Evidently, if the extracted text is entirely wrong the end-user may decide to adjust the weights of the classifiers such that the Levenshtein classifier contributes for a substantial smaller amount to the final classification result than the nearest neighbor classifier does. I propose to use a common slider control like in Figure 4.15 to provide the end-user with a convenient method of doing so. Moving it to text increases the weight of the Levenshtein classifier with respect to the weight for the nearest neighbor classifier and vice versa.

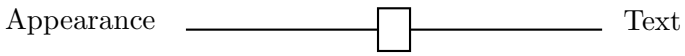


Figure 4.15: Slider control used to adjust the weights among the two combined classifiers.

Intentionally Biased Classification

Recall that I assumed the nearest neighbor classifier to be at least capable of approximating a decent classification result in order for classification to be possible at all, and therefore the Levenshtein classifier only exists to refine the ordering that has already been approximated. Therefore, to assure the weights cannot be set such that the Levenshtein classifier completely destroys the nearest neighbor results, I intentionally introduce a bias to the classification mechanism.

One of the participant functions is selected as a bias $f_\beta \in P$ and is applied to the database D such that it results in a class ordering of a certain length. This resulting class ordering

is defined as a set of candidates $C \subseteq D$.

$$\{f_{p_i} | f_{p_i} \in P \wedge f_{p_i} \neq f_\beta\} \tag{4.16}$$

Finally each remaining participant function f_{p_i}, as in Eqn. (4.16), is applied to the same database D, with the constraint that its resulting class ordering may only consist of classes c_i where $c_i \in C$. As a result, none of the remaining participant functions will ever introduce candidates that have not already been included in the ordering that was obtained through the bias function.

Observations

The described IBWVC classifier has been experimented with on a the same small dataset as we have seen before in section 4.3. Regardless of the dataset's size, results from experiments that I conducted should still provide you with useful insight in the classifier's performance and its properties.

In Figure 4.16 I show the image that served as input for the example case which I used throughout this section. In fact, the orderings in Eqn. (4.10) and Eqn. (4.11) belong to this particular instance. The text extracted by the OCR solution read

```
OKITHAILAND
M80C85AH
3062B I U
```

which is assumed to be rather accurate.

Figure 4.16: The image that served as input for the example case.

As is visible in the same figure, the imprint area has also been marked accurately which led to the results in the plot in Figure 4.17. Class c_1 is considered the best fit by both classifiers, altering the values of the weights will therefore not affect this.

Figure 4.18 shows the input image for an entirely different case. In this particular case

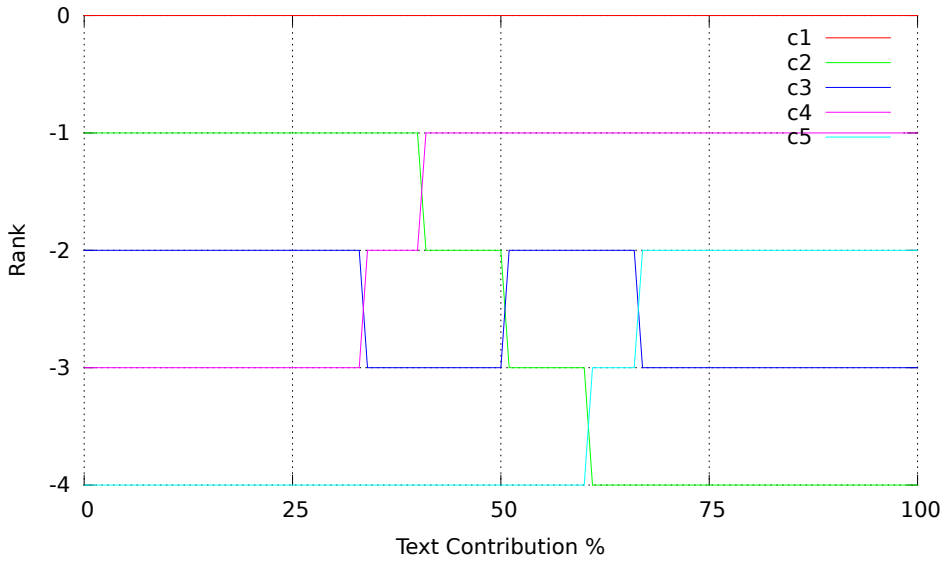


Figure 4.17: Results that show that class c_1 has been considered the best match by both classifiers and will remain so regardless of the weights.

the imprint area has not been marked entirely accurate which resulted in clear failure by the OCR solution. The text that was extracted read

```
L'--8J§I*:‘iiZT;Z‘
```

which is evidently an example of complete failure of optical character recognition.

Figure 4.18: Example input for which the OCR solution failed to correctly extract the text.

The plot in Figure 4.19 shows that depending on the amount we allow the Levenshtein classifier to contribute to the final result, it could drastically alter the ordering that was originally found by the the nearest neighbor classifier. Yet, it will never be entirely destructive to the resulting ordering, as the nearest neighbor classifier has been chosen as bias participant. The nearest neighbor classifier actually performed quite accurate once more, as class c_4, which is considered the best fitting class, truly is the best fitting class. As long as we do not allow the Levenshtein classifier to contribute over approximately 60%, this class remains the best fit in the final class ordering.

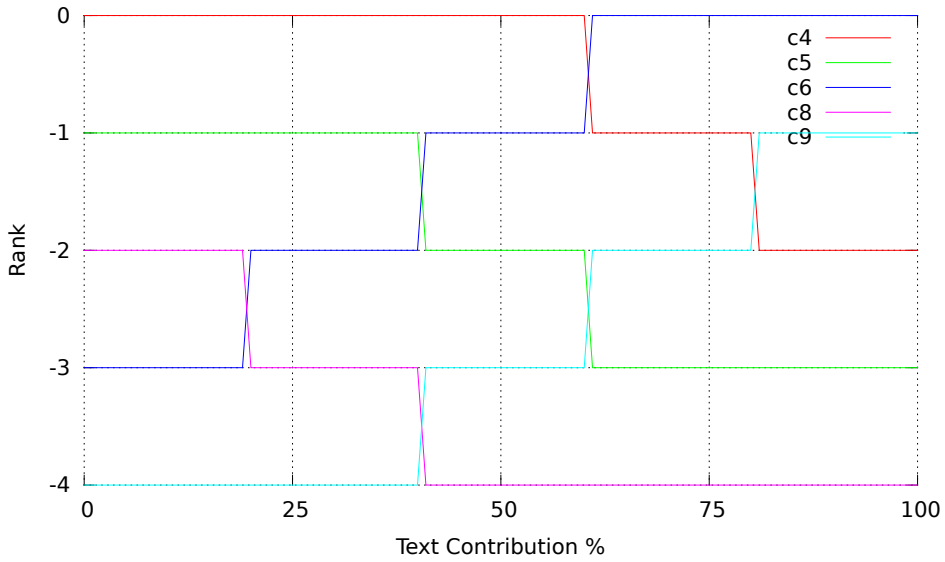


Figure 4.19: Results that show that faulty classification by the Levenshtein classifier can never be entirely destructive to the resulting order. New classes are never introduced by the Levenshtein classifier as the classification is biased in terms of appearance.

Finally we consider the example input in Figure 4.20 in which the marked imprint area is off by a substantial margin. As I mentioned before, the appearance features should however be extracted rather accurately since, when this is impossible, it is likely to assume we are unable to properly extract the text as well. This case however, turns out to be a fair example of a case in which the Levenshtein classifier offers a positive contribution to the combined classification result. Namely, the automatically extracted text

```
OKITHAILAND
N80C85AH
3062B I O
```

is a quite precise approximation of what should have been extracted.

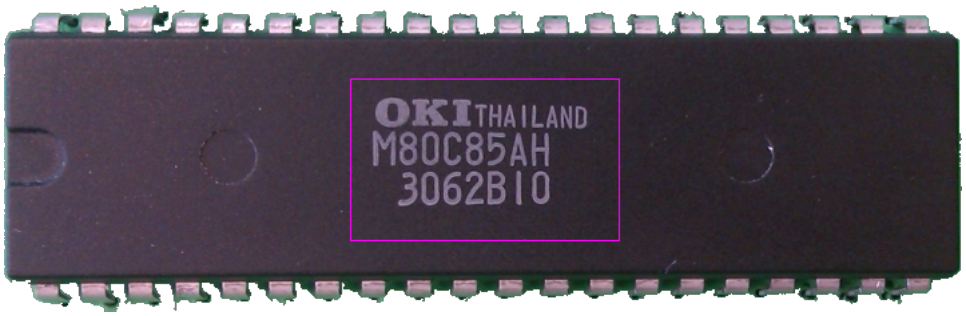


Figure 4.20: An example IC in which the imprint area has been erroneously marked which resulted in a marginal classification mistake by the nearest neighbor classifier.

The plot in Figure 4.21 shows that the Levenshtein classifier positively contributes to the final ordering if we allow it to at least contribute for approximately 25% since class c_1 is

the correct best fit for our input image.

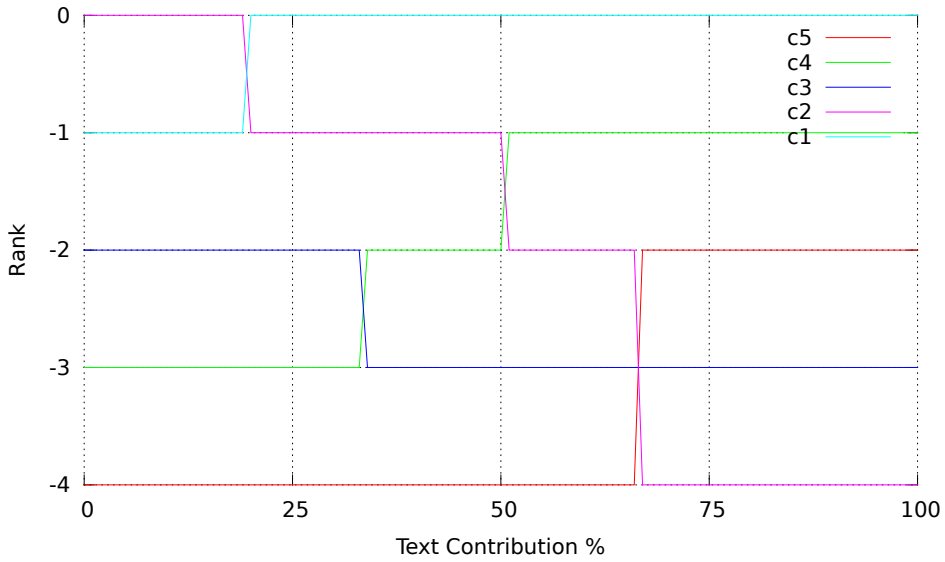


Figure 4.21: A result which shows a positive contribution from the Levenshtein classifier when we allow it to at least contribute for an approximate 25%.

5 Conclusion

To conclude this thesis I will summarize those results that were discussed throughout the previous sections which are relevant to the research question as it was originally formulated in the introduction.

> Through which applicable machine vision concepts and techniques can we **localize**, **extract** and **classify** integrated circuits on an arbitrary printed circuit board?

The subjects that were discussed in this thesis have evidently not produced a single correct answer to this question. In the next two paragraphs I will discuss the results that were obtained which were relevant for the localization, extraction and classification subtopics of the question.

5.1 Localization And Extraction

For localization and extraction purposes I proposed the usage of the seeded region growing (SRG) algorithm. This algorithm concerns an elegant and effective way to produce a segmentation for an image in a semi-interactive manner. The placement of seeds is a manual process which gives the user the possibility to correct the segmentation results if necessary. A minor extension to the algorithm has been proposed such that the algorithm takes into account that our area of interest is always gray tinted. This extension showed slight improvement in terms of marking the pins of an integrated circuit more accurately.

There remains plenty of room for improvement in the segmentation part of the solution. For instance, only experiments with ICs that were placed on a single colored background (green) have been discussed in section 3. Since only this simplified setting has been considered the part of the research question which mentions the arbitrary printed circuit board has not been answered.

Apart from that the assumption that an IC is always rectangular shaped is not taken in

consideration by the algorithm. Other issues at which the algorithm could be improved concern the fact that we would ideally not have to worry about rotation of the camera with respect to the position and placement of the ICs.

Finally, I like to conclude that manual placement of the seeds is an effective and convenient way of providing the computer with helpful information. However, ideally the user would only have to place a seed on the IC and have the computer figure out where to place the background seed(s). I therefore recommend future research to reach this ideal situation and improve on the current segmentation shortcomings.

5.2 Classification

In section 4 I proposed to exploit a simple set of appearance features which, when used by a nearest neighbor (NN) based classifier, have shown to offer an effective approximation of the class of an IC. A simple Levenshtein classifier was proposed for the purpose of classification purely based on imprint information. The Intentionally Biased Weighed Voting Classifier (IBWVC) was proposed to allow a combination of both classifiers such that the Levenshtein classifier only exists to provide refinement of the classification result that was obtained by the NN classifier. Experimental results were discussed which show that refinement of classification results can indeed be obtained through application of this new classifier.

Most important remarks to the classification results concern the size of the dataset on which experiments have been conducted. Ideally experiments could be conducted on very large datasets. It would be interesting to see if the NN results remain as useful as they proved to be for the small dataset. A large dataset would also raise questions about the performance of the classifiers in terms of computation time.

My first suggestion would however concern research of automatic extraction of the features from the input image. Image processing techniques that serve a solution to this problem should be researched such that the output of the segmentation part of the solution smoothly connects to the input of the classification part.

Bibliography

[1] Rolf Adams and Leanne Bischof. Seeded Region Growing. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, 16(6):641–647, 1994.

[2] Iman Avazpour, Ros Ernida Roslan, Peyman Bayat, M. Iqbal Saripan, Abdul Jalil Nordin, and Raja Syamsul Azmir Raja Abdullah. Segmenting CT images of bronchogenic carcinoma with bone metastases using pet intensity markers approach. *Radiol Oncol*, 43(3):180–186, 2009.

[3] Toby Berk, Lee Brownston, and Arie Kaufman. A New Color-Naming System for Graphics Languages. *Computer Graphics and Applications, IEEE*, 2(3):37–44, 1982.

[4] Christopher M. Bishop. *Pattern Recognition And Machine Learning*, volume 4. Springer, 2006.

[5] Oren Boiman, Eli Shechtman, and Michal Irani. In Defense of Nearest-Neighbor Based Image Classification. In *IEEE Conference on Computer Vision and Pattern Recognition, 2008*, pages 1–8. IEEE, 2008.

[6] Wilhelm Burger and Mark J. Burge. *Digital Image Processing: An Algorithmic Introduction Using Java*. Springer, 2008.

[7] Madirakshi Das, R Manmatha, and Edward M. Riseman. Indexing Flowers by Color Names using Domain Knowledge-driven Segmentation. In *Proceedings of the Fourth IEEE Workshop on Applications of Computer Vision (WACV'98)*, pages 94–99. IEEE, 1998.

[8] Jianping Fan, Guihua Zeng, Mathurin Body, and Mohand-Said Hacid. Seeded region growing: an extensive and comparative study. *Pattern Recognition Letters*, 26:1139–1156, 2005.

[9] Hamid Farhoosh and Gunther Schrack. CNS-HLS Mapping Using Fuzzy Sets. *Computer Graphics and Applications, IEEE*, 6(6):28–35, 1986.

[10] A. Feelders, H. Daniels, and M. Holsheimer. Methodological and practical aspects of data mining. *Information & Management*, 37(5):271–281, 2000.

[11] Tin Kam Ho, Jonathan J. Hull, and Sargur N. Srihari. Decision Combination in Multiple Classifier Systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(1):66–75, 1994.

[12] International Telecommunications Union. Parameter Values For The HDTV Standards For Production And International Programme Exchange. 1990.

[13] Akhtar Jamil, Imran Siddiqi, Fahim Arif, and Ahsen Raza. Edge-based Features for Localization of Artificial Urdu Text in Video Images. In *2011 International Conference on Document Analysis and Recognition*, pages 1120–1124. IEEE, 2011.

[14] Vladimir I. Levenshtein. Binary Codes Capable Of Correcting Deletions, Insertions, And Reversals. *Cybernetics And Control Theory*, 10(8), 1965.

[15] Ying Liu, Dengsheng Zhang, Guojun Lu, and Wei-Ying Ma. Region-based Image Retrieval with High-Level Semantic Color Names. In *Proceedings of the 11th International Multimedia Modelling Conference (MMM'05)*, pages 180–187. IEEE, 2005.

[16] David G. Lowe. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the International Conference on Computer Vision, Corfu 1999*, volume 2, pages 1150–1157. IEEE, 1999.

[17] Andrew Mehnert and Paul Jackway. An improved seeded region growing algorithm. *Pattern Recognition Letters*, 18:1065–1071, 1997.

[18] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[19] B. Mols. *Turings Tango*. Nieuw Amsterdam, 2012.

[20] Lukáš Neumann and Jiří Matas. Text Localization in Real-world Images using Efficiently Pruned Exhaustive Search. In *2011 International Conference on Document Analysis and Recognition*, pages 687–691. IEEE, 2011.

[21] Yi-Feng Pan, Xinwen Hou, and Cheng-Lin Liu. Text Localization in Natural Scene Images based on Conditional Random Field. In *2009 10th International Conference on Document Analysis and Recognition*, pages 6–10. IEEE, 2009.

[22] Frank Y. Shih and Shouxian Cheng. Automatic seeded region growing for color image segmentation. *Image and Vision Computing*, 23:877–886, 2005.

[23] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, And Logical Foundations*. Cambridge University Press, 2008.

[24] Wesley Snyder and Hairong Qi. *Machine Vision*. Cambridge University Press, 2010.

[25] William E. Winkler. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. 1990.