Autonomous feedback-based preprocessing using classification likelihoods

(Bachelorproject)

Joost Baptist, s2218755
Lambert Schomaker*

August 15, 2014

Abstract

In pattern recognition and optical character recognition (OCR) specifically, input images are presented to the classification system, they are preprocessed, features are extracted and then the image is classified. This is a sequential process in which decisions made during early preprocessing affect the outcome of the classification and potentially decrease performance. Hand-tuning the preprocessing parameters is often undesirable, as this can be a complex task with many parameters to optimize. Moreover, it is often desirable to minimize the amount of human intelligence that ends up in an autonomous system, if it can be expected that new variants of the data would require new human knowledge-based labor. A different approach to preprocessing in OCR is proposed, in which preprocessing is performed autonomously and depends on computed likelihood of classification outcomes. This paper shows that by using this approach, color, scale and rotation invariance can be achieved, as well as high accuracy and precision. The performance is solid and reaches a plateau even when noise in the data is not fully accounted for.

1 Introduction

The input of pattern recognition systems often comes from sensors in the real world. The information received from these sensors is noisy and often contains redundant information and irrelevant variations, such as variations in rotation, scale and color. The input is preprocessed in an attempt to minimize noise and irrelevant variations. After preprocessing the input, features are extracted and the pattern recognition system then makes a classification decision based on these features. This is the general structure of a pattern recognition system [6]. The process of pattern recognition is a sequential process, which means decisions made early on greatly influence decisions made during classification. Suboptimal preprocessing can lead to decreased performance.

The parameters of the preprocessing methods are often set by hand, which poses two problems. First, setting the parameters by hand means human intelligence ends up in the pattern recognition system. This is generally undesirable in autonomous systems, especially if it can be expected that new variants of the data would require new human knowledge-based labor. Second, and more important, the number of parameters can quickly increase with the number of preprocessing methods, and these preprocessing methods are often interdependent. Finding the optimum in this utility landscape by hand is a complex task, and a brute-force approach is computationally very expensive and thus unfeasible.

Optical Character Recognition (OCR) systems, which recognize characters in images, often have to deal with input images that contain irrelevant variations in, e.g., scale and rotation. These systems can not rely on structure analysis (such as described in [7]), but they must be invariant to variations by relying on invariant features such as image moments (e.g., the moments of Hu) [5] [9] or SIFT. See [3] for a review on feature extraction methods in OCR. The use of invariant features entails the risk

*University of Groningen, Department of Artificial Intelligence

of ignoring information that is actually relevant for the classification task. Alternatively, OCR systems can actively and dynamically normalize the input images so that irrelevant variations are eliminated or at least minimized. The input can be preprocessed in many different ways to achieve normalization. Each preprocessing action requires one or more parameters, and most preprocessing actions are interdependent. The actions the system may take include adjusting brightness, contrast, rotation, and perspective, resizing, thresholding, and more.

Dynamic normalization requires a quality criterion, i.e. the estimated probability that pattern x is of class c: $P(c|x)$. However, as an initial experiment we chose to use a template matching method using correlation, where the resulting correlation value is used as the quality criterion.

2 Methods

An OCR system was developed that aims to optimize the preprocessing parameters in a fixed number of iterations by maximizing the matching result of a template matcher. To test the system, a realistic, varied dataset was generated.

2.1 Dataset

The dataset consists of 780 computer-generated images of the English alphabet in uppercase Helvetica (A-Z, 26 classes) with dimensions of 50×50 pixels. Background noise and intensity, perspective, height, width, rotation, transparency and fill color (black or white) is varied randomly, in order to approximate real-world data. Figure 1 gives an impression of the dataset.

Each character initially has a black or white fill color (by a 50% probability) and has minor variations in perspective. Each character is randomly rotated within a range of -60 to 60 degrees. Rotating characters by 90 degrees or more can introduce ambiguity (e.q. an upside down 'q' becomes a 'b'), which cannot be solved without context. Each character's scale varies between 80% and 120%. Each character has a transparency of 50% to 100% (where 100% means not transparent), which causes the background to become visible through the character. Finally, the images in the dataset contain noisy backgrounds, with three levels of background intensity: light, average and dark. Of each character, there are 10 images with a dark background, 10 with an average background and 10 with a light background.

2.2 Implementation

The classification system consists of four main components:

1. **Preprocessor**. The first step is to preprocess input images. Preprocessing actions include rotating, resizing and thresholding the image.
2. **Template matcher**. The preprocessed image is matched with each class' template, or prototype, which is a black character on a white background (see Fig. 2).
3. **Classifier.** The classification decision depends on the highest match found by the template matcher.
4. **Optimization loop.** The classifier, preprocessor and template matcher are embedded in a stochastic optimization loop, which aims to find the best classification in a fixed number of iterations.

2.2.1 Preprocessing

- **Resize:** x, y. x and y are scalars that determine how much the image is resized horizontally (x) and vertically (y).

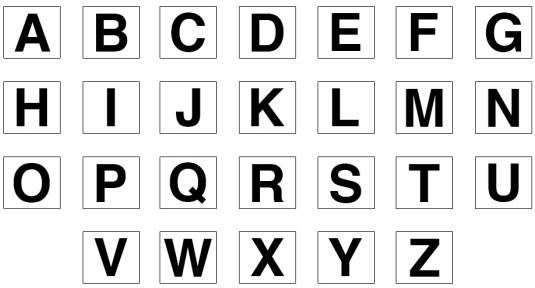

Figure 2: Templates used for template matching. Borders are not present in actual template images.

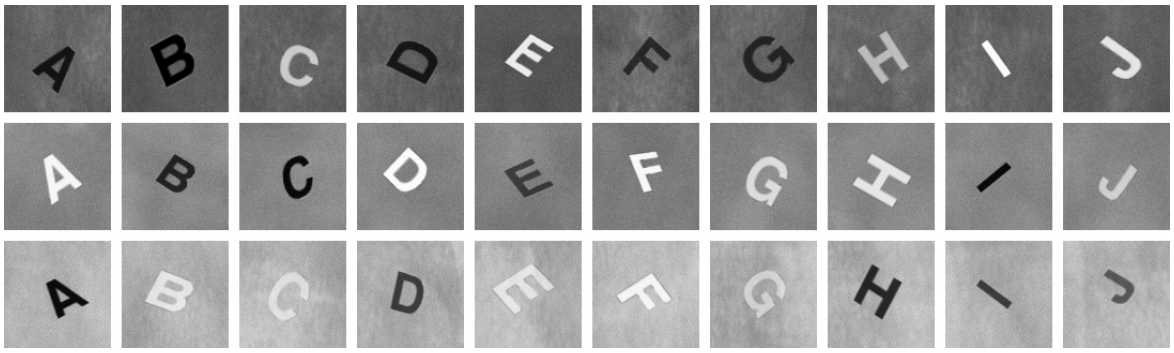

Figure 1: Impression of the dataset. The dataset contains 780 images (30 per character) of the uppercase English alphabet in Helvetica. Background intensity, perspective, rotation, fill color and noise are varied randomly.

- **Rotate:** ϕ. Rotates the image by ϕ degrees. Corners are extrapolated using a reflection method.
- **Threshold:** θ. Thresholds image at θ, where intensities smaller than or equal to θ are set to 0, and intensities greater than θ are set to 255.

These three methods allow a color, scale and rotation dependent matching algorithm to work well with a dataset of images with varying color, scale and rotation, which means that the system can effectively become invariant to these variations. The minor variations in perspective are not normalized (there is no perspective transformation preprocessing action), and could be perceived as unexplainable noise in the data as it often occurs in the real world.

2.2.2 Template matching

The system uses a conventional template matcher, where a template is slided over the input image in the horizontal and vertical directions, and a match is calculated for each position. The match that is calculated is the standard Pearson correlation, defined as:

$$\rho(A, B) = \frac{\text{cov}(A, B)}{\sigma(A)\sigma(B)}. \tag{2.1}$$

Taking into account the fact that the template image T is slided over the input image I, the match $r(x, y)$ is defined as:

$$r(x, y) = \frac{\sum_{x',y'}(T'(x', y') \cdot I'(x + x', y + y'))}{\sqrt{\sum_{x',y'} T'(x', y')^2 \cdot \sum_{x',y'} I'(x + x', y + y')^2}}, \tag{2.2}$$

where $T' = T - \overline{T}$ and $I' = I - \overline{I}$.

The value of $r(x, y)$ is the correlation between the input image and the template image at position (x, y). It is a number between -1 and 1, where 1 indicates perfect correlation and -1 indicates perfect anti-correlation. Using this, the best match between T and I, $R(T, I)$, is defined as:

$$R(T, I) = \max |r(x, y)|. \tag{2.3}$$

2.2.3 Classification

The class of input image I is determined by the class of the highest matching template image. Let T_i denote a template image of class i, then the class of image I, $c(I)$, is defined as:

$$c(I) = \underset{i}{\text{argmax}}\ R(T_i, I) \tag{2.4}$$

2.2.4 Optimization

In order to maximize $R(T_i, I)$, the input image I has to be preprocessed in such a way that it resembles the template image T_i as much as possible. Let $P(I, \Theta)$ denote the preprocessing function with preprocessing parameters $\theta_1, \theta_2, ..., \theta_n \in \Theta$, that returns the preprocessed image. The classification function becomes:

Table 1: Experiment parameters: the ranges in which the preprocessing parameters are stochastically optimized.

Method	Param	Range
Rotate	ϕ	$-60 \leq \phi \leq 60$
Resize	x	$0.8 \leq x \leq 1.2$
Resize	y	$0.8 \leq y \leq 1.2$
Threshold	θ	$0 \leq \theta \leq 255$

$$c(I) = \underset{i}{\operatorname{argmax}}\ R(T_i, P(I, \Theta)). \tag{2.5}$$

The goal is to find a set of parameters Θ and a template T_i such that $R(T_i, P(I, \Theta))$ is maximized. This optimization problem is solved using random sampling. The function to optimize is:

$$c(I) = \underset{i,\Theta}{\operatorname{argmax}}\ R(T_i, P(I, \Theta)). \tag{2.6}$$

Note that if we assume the preprocessing parameters are discrete, s is the number of evaluation steps per parameter and n is the number of parameters, there are s^n different solutions. As s^n can quickly become a large number, a brute-force approach is not feasible. As an example, assume 4 parameters and 100 evaluation steps per parameter, this would amount to $100^4 = 10^8$ complete preprocessing runs in the case of brute-force grid search. We will explore how quickly Monte-Carlo based optimization will yield acceptable results.

2.3 Experiment

To test the system's performance, an experiment is conducted that optimizes the preprocessing parameters in ranges as defined in Table 1. The system's performance is expressed by the accuracy and precision. Besides performance, the error and computation time are also measured.

3 Results

The prior probability $P(c|x)$, is $\frac{1}{26}$, or approximately 0.038. The achieved accuracy and precision of the system are shown in Figure 3a. The figure shows that for $N < 200$ both accuracy and precision increase rapidly to approximately 0.93. The accuracy and precision finally reach approximately 0.97 for $N = 1000$. The computation time increases linearly with the number of iterations N. Figure 3b shows that the computation time is approximately 1 second per 100 iterations.

The average, absolute errors of the obtained rotation and resize parameters are shown in Figures 3c and 3d respectively. Both figures show that the error decreases rapidly for $N < 200$ and then start to converge. Neither preprocessing action reaches an error of 0.

Table 2: Misclassification rates for $N = 1000$. Read as "G was misclassified as C 3.33% of the time." Table only includes classes that were misclassified and the classes they were misclassified as. The actual number of classes is 26 (A-Z).

	B	C	I	L	N	O	Q
G		3.33%				3.33%	3.33%
O		3.33%					
Q						30%	
S	3.33%						
W			3.33%				
Z				3.33%	3.33%		

Table 2 shows the misclassification rates for the experiment run with 1000 iterations. It contains only the classes which were misclassified at least once, of which there are 6, and the classes which they were misclassified as, of which there are 7. In reality there are 26 classes, but to keep the table compact the remaining classes are not included. The table shows that most characters were misclassified only sporadically, except for the 'Q', which was incorrectly classified as an 'O' 30% of the time.

4 Discussion

4.1 Results

The results show that high performance can be achieved in a relatively small number of iterations. The system requires approximately 200 iterations for a correct classification rate greater than 90%,

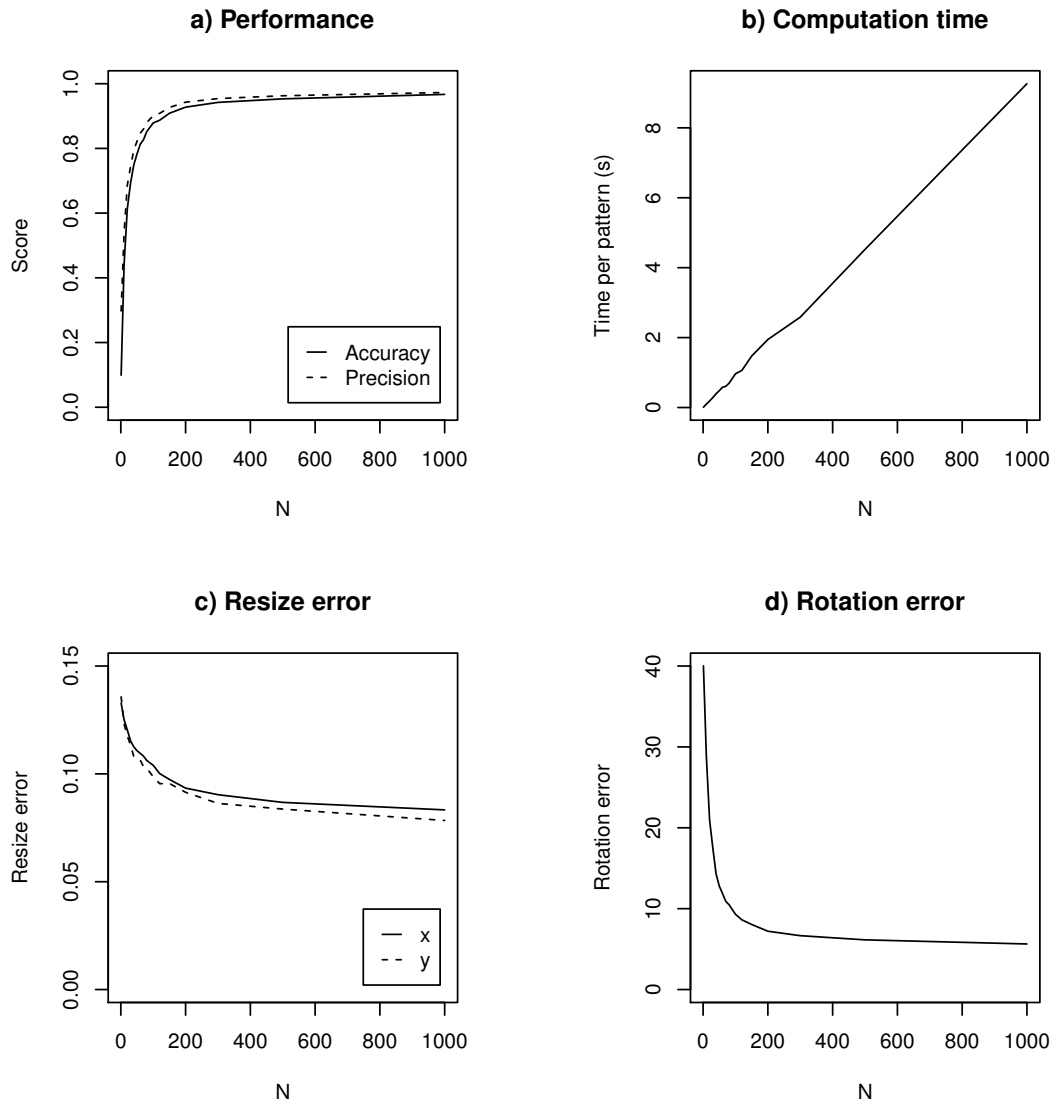

Figure 3: a) Performance. Number of iterations N vs. accuracy and precision of the system. The plot shows a rapid increase in performance for $N < 200$, after which the performance slowly converges to its maximum of approximately 0.97 for $N = 1000$. b) Computation time. The computation time is linear with N and increases by approximately 1 second per 100 iterations. Run on a single processor. c) Rotation error. Absolute difference between obtained rotation angle and true rotation angle. d) Resize error. Absolute difference between obtained scale and true scale.

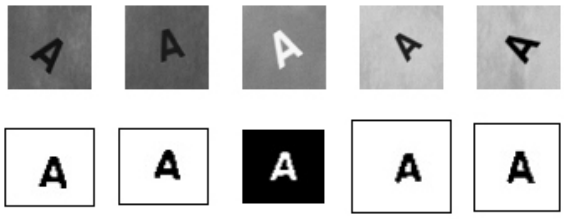

Figure 4: Impression of how the input images were preprocessed for an optimal match. The result is a normalized dataset.

even though the search space is very large. Monte-Carlo based optimization requires very few iterations compared to brute-force search, at the cost of at most a slight performance hit.

The results also show that rotation, scale and color invariance is achieved. The effect of optimization is that input images are preprocessed in such a way that they resemble the template image as much as possible. Consequently, a normalized dataset is generated that could serve as input to other classifiers or different systems. Figure 4 gives an impression of the normalized dataset. Note that inversed images can easily be flipped by storing the sign of the matching result before taking the absolute value during template matching (see Equation 2.3), and using the sign to determine whether the image should be inversed or not.

4.2 Template matching

Template matching can be of limited practical value as it requires the input fonts to be similar to the template fonts. This can be overcome by adding more templates per character, at the cost of more computation time. Alternatively a different feature extraction method could be used, such as those discussed in [3].

4.3 Optimization

The advantage of stochastic optimization is that, given a sufficient number of iterations, it produces a representative sample of the full population. This increases the likelihood a high quality set of preprocessing parameters is found. A disadvantage is that, especially with a small number of iterations, it might ignore obvious nearby local or global optima, and thus provide a suboptimal solution that could easily be improved by, e.g., a simple hill-climbing, evolutionary [4] or particle swarm [8] approach.

Figure 5 shows the optimization landscapes for rotation, scale and thresholding. Each plot shows the preprocessing parameter's landscape where all other preprocessing parameters are fixed and set to optimal values as obtained by the system during the experiment, and the landscape where the other preprocessing parameters are fixed and set to random values.

Figure 5a gives the rotation landscape and shows that given the obtained optimal parameters, the global maximum is found at an angle of approximately 320 degrees. When all other parameters are set to random values, there is no longer a single global maximum, but rather, three local maxima. Figure 5b, which gives the resize landscape, shows that a clear global maximum disappears when the other parameters are set to random values. Figure 5c gives the threshold landscape and is arguably the least dependent on the other preprocessing parameters. The global maximum remains in the same place when the other parameters are set to random values, although it becomes less extreme.

The main conclusion that we can draw from Figure 5 is that optimization parameters are highly interdependent; each parameter belongs to a conditional distribution. It is very difficult to find the optimum of one parameter if the other parameters are suboptimal. Moreover, the quality of an obtained parameter always depends on the values of the other parameters. As a consequence of this and the size of the parameter space, an optimization method that focuses on (multi-dimensional) exploration is necessary. Random sampling, as this paper shows, seems to be a good candidate, though evolutionary, hill-climbing and particle swarm optimization approaches may be possible candidates for future research.

Alternatively, a more advanced Markov Chain Monte-Carlo (MCMC) algorithm might yield better results. MCMC methods approximate the desired distribution based on the state of the Markov chain after a large number of steps (see [1] for a review). As discussed, the optimization of the preprocessing parameters involves sampling from conditional distributions. Gibbs sampling [2] is useful when the joint distribution is difficult to sample from while the individual conditional distributions

can easily be sampled from. Gibbs sampling methods sample from one conditional distribution at a time, during which the other variables (preprocessing parameters) remain fixed. The idea is that it is easier to simulate a sequence of univariate conditional distributions which can be used to approximate the full joint distribution than to simulate the full joint distribution directly. The problem of optimizing the preprocessing parameters seems very well suited to the Gibbs sampling method. Future research could point out whether or not this approach leads to better performance and whether or not computation times remain acceptable.

4.4 Advantages and disadvantages of dynamic normalization

Dynamic normalization systems require no training in advance, because training is essentially done live, during optimization. The advantage of this approach is that the system is not influenced by peculiarities in a training set. The flip side is that dynamic normalization systems require a quality criterion, which limits the number of potential feature measurement and classification methods to those that provide a quality criterion, such as a template matching result.

An important advantage of dynamic normalization systems over invariant feature systems is that dynamic normalization systems generate a normalized dataset as a by-product. Image moments do not give sufficient information to reconstruct the original image [9]. The normalized dataset provides insight into decisions made by the system and can also serve as input to classifiers that are not invariant to variations in input (e.g., neural networks) or other systems.

4.5 Applications

Dynamic normalization systems can be applied in any environment in which invariance to irrelevant variations is necessary, and are not limited to recognition of optical characters. Notable applications include mobile applications (such as Android and iOS applications) that aim to recognize patterns in camera images, where irrelevant variations, e.g., in scale and rotation, are inevitable.

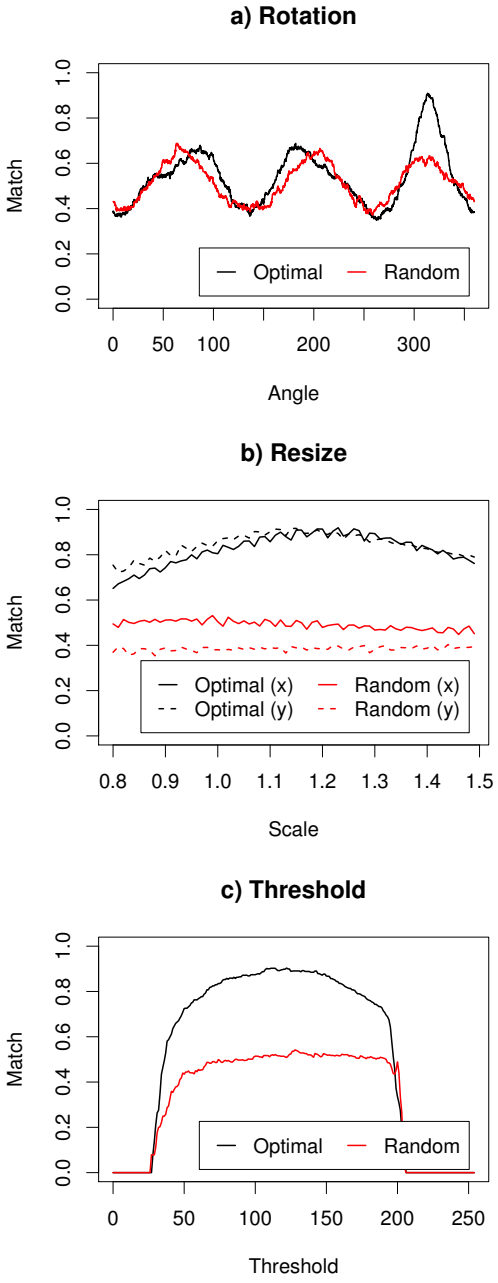

Figure 5: Optimization landscape of the rotate, resize and threshold actions. The black lines show the landscape where all other preprocessing parameters are set to optimal values, as found by the system during the experiment. The red lines show the landscape where all other parameters were set to random values. When the other parameters are set to random values, the optimization landscape changes drastically and maxima may disappear.

5 Conclusions

5.1 Summary

This paper has shown that dynamic normalization systems can become invariant to irrelevant variations in input, achieve high performance, and do not require an excessive amount of computation time. An important observation is that dynamic normalization systems can come up with workable solutions in a relatively low number of iterations. Another advantage is that dynamic normalization systems generate a normalized dataset as a by-product, which can serve new purposes.

5.2 Future work

Future work could focus on alternative optimization methods. As discussed in the previous section, good alternatives to simple random sampling may exist. Specifically, we have high hopes for Gibbs sampling. More preprocessing actions that manipulate perspective, line thickness, and more could be implemented to allow more variations in the input, where a new balance must be found between these preprocessing actions and desired computation time. Template matching limits the system to a subset of possible fonts. The template matching method could be extended to allow multiple font styles in the input, or an alternative to template matching could be implemented. Lastly, future work could focus on comparing performance of systems that employ dynamic normalization versus systems that use invariant features.

References

[1] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

[2] George Casella and Edward I George. Explaining the gibbs sampler. *The American Statistician*, 46(3):167–174, 1992.

[3] Øivind Due Trier, Anil K Jain, and Torfinn Taxt. Feature extraction methods for character recognition-a survey. *Pattern recognition*, 29(4):641–662, 1996.

[4] David B Fogel. An introduction to simulated evolutionary optimization. *Neural Networks, IEEE Transactions on*, 5(1):3–14, 1994.

[5] Ming-Kuei Hu. Visual pattern recognition by moment invariants. *Information Theory, IRE Transactions on*, 8(2):179–187, 1962.

[6] Anil K Jain, Robert P. W. Duin, and Jianchang Mao. Statistical pattern recognition: A review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(1):4–37, 2000.

[7] Shunji Mori, Ching Y Suen, and Kazuhiko Yamamoto. Historical review of ocr research and development. *Proceedings of the IEEE*, 80(7): 1029–1058, 1992.

[8] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

[9] Michael Reed Teague. Image analysis via the general theory of moments*. *JOSA*, 70(8):920–930, 1980.