A Reinforcement Learning Agent for Minutiae Extraction from Fingerprints

Asker M. Bazen [a]
a.m.bazen@el.utwente.nl

Martijn van Otterlo [b]
otterlo@cs.utwente.nl

Sabih H. Gerez [a]
s.h.gerez@el.utwente.nl

Mannes Poel [b]
mpoel@cs.utwente.nl

[a] Dept. of Electrical Engineering, Signals and Systems,
[b] Dept. of Computer Science, TKI,
University of Twente,
P.O. box 217 - 7500 AE Enschede - The Netherlands

Abstract

In this paper we show that reinforcement learning can be used for minutiae detection in fingerprint matching. Minutiae are characteristic features of fingerprints that determine their uniqueness. Classical approaches use a series of image processing steps for this task, but lack robustness because they are highly sensitive to noise and image quality. We propose a more robust approach, in which an autonomous agent walks around in the fingerprint and learns how to follow ridges in the fingerprint and how to recognize minutiae. The agent is situated in the environment, the fingerprint, and uses reinforcement learning to obtain an optimal policy. Multi-layer perceptrons are used for overcoming the difficulties of the large state space. By choosing the right reward structure and learning environment, the agent is able to learn the task. One of the main difficulties is that the goal states are not easily specified, for they are part of the learning task as well. That is, the recognition of minutiae has to be learned in addition to learning how to walk over the ridges in the fingerprint. Results of successful first experiments are presented.

Keywords

Image processing, fingerprint recognition, minutiae extraction, image exploring agents, reinforcement learning, neural networks.

1 Introduction

Fingerprint recognition has received increasingly more attention during the last years. However, some challenges have to be solved before the performance of fingerprint recognition systems is sufficient for large scale high-security applications. One of the problems to be solved is the robust extraction of minutiae from a fingerprint image. In this paper, the design of an agent that extracts the minutiae from a fingerprint by means of reinforcement learning is discussed.

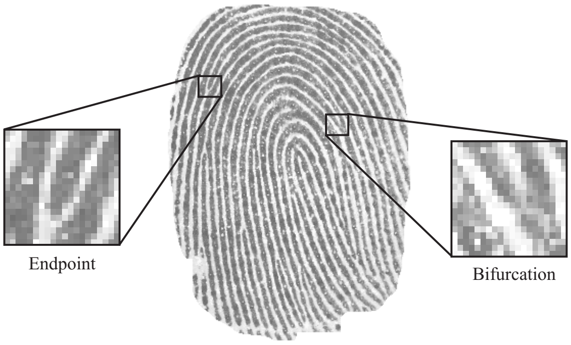

Figure 1: Example of a fingerprint and two minutiae.

In Figure 1, a fingerprint is depicted. The information carrying features in a fingerprint are the line structures, called *ridges* and *valleys*. In this figure, the ridges are black and the valleys are white. The *minutiae*, ridge-endings and bifurcations, provide the details of the ridge-valley structures. Minutiae are used for fingerprint *matching*, which is a one-to-one comparison of two fingerprints.

One of the first steps in fingerprint recognition is the extraction of the minutiae from the fingerprint image. The classical approach uses a number of image processing steps for this task [3]. First, the fingerprint image is filtered for noise suppression. Then, a threshold is applied to the image in order to obtain a binary image. Next, the ridges are thinned to 1 pixel width by morphological operations. From this skeleton, minutiae extraction is a straightforward task. However, this method is highly sensitive to noise and bad image quality. This results in the extraction of many false minutiae. Therefore other, more robust, methods have to be investigated.

In this paper, an agent-based approach is proposed to extract the minutiae from a fingerprint image. It has been shown that it is a good policy to follow the ridges in the fingerprint until a minutia is found. Maio and Maltoni [7] presented an agent that takes small steps along the ridge. Jiang et al. [4], enhanced the agent by using a variable step size and a directional filter for noise suppression. This results in a rather complex system, especially since robustness is required. A much simpler solution is to use a agent that *learns* the task. By using a variety of training examples, a robust system can be obtained. Van der Meulen et al. [8] used genetic programming to evolve a minutiae extracting agent. In the approach of this paper, the agent is trained by means of *reinforcement learning* (RL).

The rest of this paper is organized as follows. In Section 2, reinforcement learning is explained. In Section 3, it is discussed how to apply RL to the problem of minutiae extraction. Finally, in Section 4, some experimental results are presented after which they are discussed in Section 5.

2 Reinforcement Learning

Reinforcement learning (RL) [5, 10] is a powerful learning technique in domains where there is no *instructive* feedback, as in supervised learning, but only *evaluative* feedback. Agents are trained by rewarding and scaffolding, expressed in terms of real numbers.

In short, a RL agent learns in the following way. First, it perceives a *state* s_t. Then, on the basis of its experience, it chooses its best *action* or, with a small probability, a random action, action a_t. This action is rewarded by the environment with *reinforcement* r_t, after which the agent perceives the newly entered state s_{t+1}. This *interaction* continues until the agent enters a *terminal state*, i.e. its *episode* ends. Terminal states are either states in which the agent's goal is satisfied, or states in which the episode is terminated externally, possibly because of an illegal action. One of the main difficulties is that non-zero rewards are usually sparse and are sometimes only given at the end of the episode.

The goal of the agent is to maximize its reward by learning the optimal *policy*, i.e. a mapping from states to actions. This can be done by learning *value functions.* Value functions reflect the *expected cumulative reward* the agent receives by following its policy. In RL, one generally uses two kinds of value functions: $V : S \to \mathbb{R}$ (state values) and $Q : S \times A \to \mathbb{R}$ (state-action values). In this paper we use the latter to reflect the *expected cumulative reward resulting from executing action a in state s and thereafter following the policy.* This mapping is approximated on the basis of interaction with the environment. The policy can easily be found by choosing in each state the action that maximizes the Q-value, i.e. the *greedy* action.

Q-Learning is a commonly used *off-policy* algorithm for learning the action values. In this paper, however, we use the related algorithm Sarsa[1] [9, 10], which is an *on-policy* TD algorithm:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{1}$$

In this algorithm, $Q(s_t, a_t)$ is updated in the direction $[r_t + \gamma Q(s_{t+1}, a_{t+1})]$. The *discount factor* γ determines how future rewards are weighted. The next state and action, s_{t+1} and a_{t+1}, are determined by the policy. The update of the approximation of one state-action pair uses the approximation of other state-action pairs. This is called *bootstrapping.*

For selection of the actions that are taken, ϵ-greedy action selection is used. This selection criterion chooses an exploratory action with probability ϵ and the greedy action, having the highest Q-value, otherwise.

Usually, when the state-action space is reasonably small, we can store all the Q-values in a simple lookup-table. Since the state-space in our problem has a very high dimension (144, which might be reduced to 50 by linear feature extraction as discussed in Section 3), a function approximator for storing the values is used.

[1] Sarsa stands for ***S****tate*-***A****ction*-***R****eward*-***S****tate*-***A****ction*, which are the necessary elements for performing the update in eq. 1

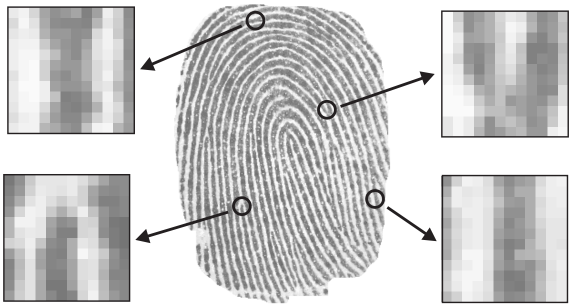

Figure 2: Situatedness: local view of the agent of 12×12 pixels.

The function approximator has one continuous output, $Q(s, a)$. We use a *multi-layer perceptron* (MLP) neural network for the approximator. At each step, the Q-function is updated by backpropagating the error in the right-hand side of (1).

For Sarsa, the update of the weights $\mathbf{w}$ of the neural network is given by:

$$\Delta \mathbf{w} = \eta \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \nabla_{\mathbf{w}} Q(s_t, a_t) \tag{2}$$

where η is the learning rate and $\nabla_{\mathbf{w}} Q(s_t, a_t)$ is a vector of output gradients.

Training the neural network is performed by offering examples $((s_t, a_t),\ r_t + \gamma Q(s_{t+1}, a_{t+1}))$ to the neural network. These examples can be obtained by letting the agent interact with its environment, whereby the neural network determines the agent's actions. The training itself can be done either on-line [9] or off-line [6]. In this paper, we will use on-line adaptation of the network.

3 RL for Minutiae Detection

This section describes how RL can be applied to the minutiae extraction problem. The *goal* of the agent is to follow a ridge and stop at a minutia (see Figure 1). Furthermore, the minutia should be found in as few steps as possible in order to minimize the computational time needed for minutiae extraction. This is a typical example of an *episodic task*, which terminates when a minutia is found.

The agent only has a local view of the fingerprint image around it, e.g. it is *situated* in its environment. It can observe the gray scale pixel values in a segment of $n \times n$, for instance 12×12, pixels around it. The orientation of the agent is normalized by rotating the local view, such that the forward direction is always along the ridge-valley structures. For this purpose, the *directional field* is used [1]. The local view is illustrated in Figure 2. This introduces considerable *a priori* knowledge. The agent is always aligned with the directional field, and this puts restrictions on the state space of the agent. Local views in which the agent has a direction other than approximately aligned with the ridges in the fingerprint do not occur.

As was explained in Section 2, the dimensionality of the state space requires function approximation which is e.g. provided by a multilayer perceptron. Furthermore, the length of the feature vector, which contains the pixel values in the local view, is reduced by a *Karhunen-Loève transformation* (KLT) [2]. The KLT transforms feature vectors to a new basis that is given by the eigenvectors of their covariance matrix. Then, only those KL components that correspond to the largest eigenvalues are selected. This way, the largest amount of information is preserved in the smallest transformed feature vector. This has two useful effects. First, the length of the feature vector is reduced, which simplifies the learning process. Second, the KL components that carry the least information and therefore represent the noise, are discarded. This noise suppression method eliminates the need of a directional filter, which would require approximately 5 times as many computations.

The *actions* of the agent are the moves that it can make in the coordinate system of its local view. Since the agent is always approximately aligned with respect to the directional field, the forward action is always a move along the ridge-valley structures, and the agent does not need rotational or backward actions. To be able to achieve its goals, moving as fast as possible along the ridges and stopping at minutiae, the agent may move up to 4 pixels forward at each step and up to 1 pixel to the left or to the right. The left-right actions are necessary for keeping on the ridge. Although the directional field puts the agent in the right direction, it is not sufficient to keep the agent exactly on the ridges. The action results in a new position in the fingerprint, after which a new local view is extracted at that position.

The *reward structure* is another key element in the definition of an RL experiment. It determines the optimal actions for the agent to take. Setting up the right reward structure is very important for learning the (right) task. First, the agent receives rewards for staying at the center of a ridge. This is implemented by manually marking the ridge centers and using an exponential Gaussian function of the distance of the agent to the ridge center. Second, a much higher reward is given near the endpoints to be detected. Again a Gaussian function of the distance to the minutia is used. This structure encourages the agent to move in as few steps as possible to the endpoint by following a ridge and then to stop moving in order to receive the higher reward forever. This is enforced even more by scaling the reward such that large steps along the ridge receive a higher reward than small steps, while the opposite applies at endpoint locations. The reward structure around a ridge with an endpoint is shown in Figure 3. The peak in the reward corresponds to the endpoint of the ridge and the line of higher reward correspond to the ridge in the fingerprint.

The agent can be *trained* by selecting a ridge, initiating the agent at some location on that ridge, and defining the reward structure with respect to that ridge, including the target minutia that the agent should find. Then, the agent starts moving and the network is updated until the episode ends when the agent moves too far from the target ridge. It is worth noticing that during one episode the agent is trained to follow a certain ridge and it only gets rewards for being near that particular ridge. Therefore, the reward structure is different for each

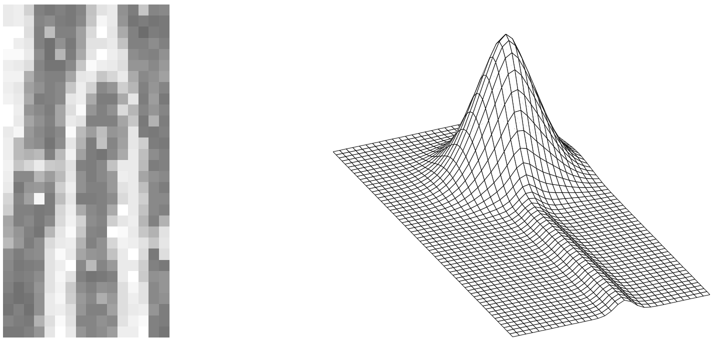

Figure 3: Reward structure around a ridge with an endpoint.

different ridge that is used for training.

During *testing* and actual *use*, agents are initiated at a large number of positions in the image, for instance on a regular grid. They start moving according to their policy and follow the ridges. However, there is no clear *termination* criterion for the agents, since it is not known in advance which minutia they should find. To overcome this problem, endpoints are detected by counting the number of successive small steps. After 5 small steps, the agent is terminated and an endpoint is detected. Bifurcations are detected by keeping a map of the trajectories of all agents. When an agent intersects the path of another one, the agent is terminated and a bifurcation is detected.

4 Experimental Results

In this section, the setup of the training experiments and its results are presented. The training is performed on the fingerprint image of Figure 1. In this fingerprint, all ridges and all minutiae have been marked manually. For each episode, one ridge is selected and the agent is initiated at a random position on that ridge. Then, the reward structure is calculated for that ridge as explained in Figure 3. Along the ridge, $\sigma^2 = 1$ and the amplitude is 1, while at a minutiae, $\sigma^2 = 10$ and the amplitude is 10. Next, the agent is trained by the Sarsa algorithm as explained in Section 2. Finally, the agent is terminated if it is more than 7 pixels from the indicated ridge center.

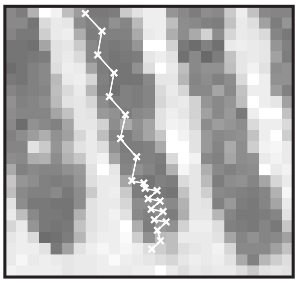

Figure 4: Some Path.

A multitude of experiments have been performed to find the optimal setup of the algorithm. At this stage, no definitive parameter values can be given, although some numbers are presented here to give a first impression. For one training session, 50,000 episodes were used. During training, the exploration parameter decreases from $\epsilon = 0.01$ to $\epsilon = 0$. The size of the local view of the agent was taken as 12×12 pixels, which was reduced to a feature vector of length 50 by the KL transformation. The actions were constrained to a grid of discrete values up to 1 pixel to the left or to the right and up to 4 pixels forward. The multi-layer perceptron had 1 hidden layer of 22 neurons and the learning rate was $\eta = 10^{-2}$. The discounting factor was set to $\gamma = 0.9$.

Testing was performed on another fingerprint as described in Section 3. Starting points have been selected at a regular grid and the agents follow their policy until termination. Human inspection of the results, shown in Figures 5 and 4, indicates that the agent follows the ridges and that all minutiae have been detected. The agent takes relatively large steps along the ridges, while the step size decreases near the endpoints. Furthermore, it intersects its own path at bifurcations. However, the figure also shows a number of false minutiae. These might be eliminated by further training of the agent on other fingerprints and fine-tuning of the parameters. Another possibility is the application of post-processing techniques to eliminate false minutiae structures.

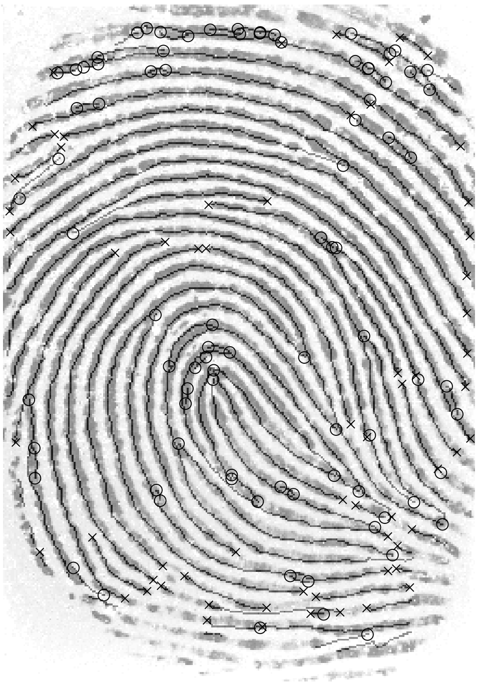

Figure 5: Extracted minutiae.

5 Conclusions

In this paper we showed that reinforcement learning is a useful and intuitive way to tackle the problem of robust minutiae extraction from fingerprints. This new combination is now in its proof-of-concept phase. It has been shown that an adaptive agent can be trained to walk along the ridges in a fingerprint and mark minutiae when encountered. The system uses straightforward reinforcement learning techniques. There is still much room for fine tuning parameters and algorithms. We are planning to do an experimental study on a variety of parameters and other learning algorithms, like Q(λ)-learning, as well. Comparison of this method with other methods will follow as soon as a final setup of the algorithm will be fixed.

The use of *value-based RL* algorithms can turn out to be not the best choice in our application. The similar local views on different places on the ridges creates a kind of *partial-observable* state representation, which did not create severe problems in our case though. It might be better to search directly for a policy by using *policy gradient* [11] methods or to use relative Q-values instead. We plan to investigate this as well.

Further research into the combination of adaptive agents and fingerprint matching should focus on the correct retrieval of all minutiae in the fingerprint, especially in corrupted and noisy images. Robustness should solve for example the problem of deciding whether a line was broken due to image corruption or that a real endpoint, thus a minutia, was found.

The use of multiple agents on the same fingerprint could be an interesting extension. For example, when an agent encounters a bifurcation, it should split into two agents both following one ridge departing from the bifurcation. Also, multiple agents could decide as a group that a minutiae is found, by approaching from different sides with different local views.

References

[1] A.M. Bazen and S.H. Gerez. Directional field computation for fingerprints based on the principal component analysis of local gradients. In *Proceedings of ProRISC2000, 11th Annual Workshop on Circuits, Systems and Signal Processing*, Veldhoven, The Netherlands, November 2000.

[2] A.K. Jain. *Fundamentals of Digital Image Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[3] A.K. Jain, L. Hong, S. Pankanti, and R. Bolle. An identity-authentication system using fingerprints. *Proc. of the IEEE*, 85(9):1365–1388, September 1997.

[4] X. Jiang, W.Y. Yau, and W. Ser. Detecting the fingerprint minutiae by adaptive tracing the gray-level ridge. *Pattern Recognition*, 34(5):999–1013, May 2001.

[5] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[6] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning Journal*, 8(3/4), 1992. Special Issue on Reinforcement Learning.

[7] D. Maio and D. Maltoni. Direct gray-scale minutiae detection in fingerprints. *IEEE Trans. PAMI*, 19(1):27–39, January 1997.

[8] P.G.M. van der Meulen, H. Schipper, A.M. Bazen, and S.H. Gerez. PMDGP: A distributed object-oriented genetic programming environment. In *Proc. ASCI Conference 2001*, Heijen, The Netherlands, May 2001.

[9] G.A. Rummery and M. Niranjan. On-line Q-Learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University, Engineering Department, 1994.

[10] R.S. Sutton and A.G. Barto. *Reinforcement Learning: an Introduction.* The MIT Press, Cambridge, Massachusetts, 1998.

[11] R.S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.