# Theory, Programming and Simulation of Neural Networks (Prof. Ruedi Stoop)

## Rotation, scale and translation invariant features for character classification

Philipp Benner <philipp.benner@ini.phys.ethz.ch>

July 24, 2009

The purpose of this project is to evaluate a method of feature extraction for rotation, scale and translation invariant character classification based on the Fourier-Mellin transform. Feature extraction results in information loss as the phase of the Fourier coefficients is dropped. This project should answer the question of how important this information is and whether this method can still be used for character classification. A second issue is the discrete nature of pictures which will add noise to the extracted features (after the log-polar mapping and after rotation or scaling). An artificial neural network trained with backpropagation is used here as a filter. It is expected that the accuracy of the method will drop as the number of different characters to be classified is increased, due to increased overlap of the clusters in feature space.

## Introduction

Classification of characters is a difficult task, especially if the characters differ in size, alignment and position. Learning from such data with a neural network is generally speaking not possible, because the error landscape is not suited for gradient descent methods. The goal of a good feature extraction method is to end up with features that vary little under scaling, rotation, and translation of the characters. This simplifies the error landscape and eases the classification task. The method presented here is based on the Fourier-Mellin transform which in theory extracts features that are invariant under all three transformations. There are however some drawbacks which need to be addressed. Firstly, only the magnitude of the Fourier transform is used and the phase is droppen which results in a loss of information. This may cause different classes to overlap in feature space as the number of classes is increased. Secondly, the theory is based on continuous data, images however are discrete which especially makes the log-polar mapping difficult. This adds noise to the invariant features that needs to be filtered out by the learning algorithm.

## Method

This section briefly presents feature extraction based on the Fourier-Mellin transform. Let $f(x_1, x_2)$ be an image with Cartesian coordinates $x_1$ and $x_2$. The two-dimensional discrete Fourier transform and its inverse are given by

$$\begin{aligned} \mathcal{F}(k_1, k_2) &= \sum_{x_1=0}^{N_1-1} \sum_{x_2=0}^{N_2-1} f(x_1, x_2) e^{-j2\pi x_1 k_1/N_1 - j2\pi x_2 k_2/N_2} \\ f(x_1, x_2) &= \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} \mathcal{F}(k_1, k_2) e^{+j2\pi x_1 k_1/N_1 + j2\pi x_2 k_2/N_2}, \end{aligned} \tag{1}$$

where $N_1 \times N_2$ denotes the size of the image. The following properties of the Fourier transform are relevant to feature extraction.

- Translation property (translation with offsets $a$ and $b$)

$$f(x_1 + a, x_2 + b) \Leftrightarrow \mathcal{F}(k_1, k_2) \underbrace{e^{-j(ak_1 + bk_2)}}_{|\cdot| \Rightarrow 1} \tag{2}$$

- Reciprocal scaling (scaling by a factor of $\rho$)

$$f(\rho x_1, \rho x_2) \Leftrightarrow \frac{1}{\rho} \mathcal{F}(\frac{k_1}{\rho}, \frac{k_2}{\rho}) \tag{3}$$

- Rotation property (rotation through an angle $\Theta$)

$$\begin{aligned} &f(x_1 \cos\Theta - x_2 \sin\Theta, x_1 \sin\Theta + x_2 \cos\Theta) \Leftrightarrow \\ &\mathcal{F}(k_1 \cos\Theta - k_2 \sin\Theta, k_1 \sin\Theta + k_2 \cos\Theta) \end{aligned} \tag{4}$$

From the translation property it follows that the magnitude of the Fourier transform of an image is translation invariant. Of course, dropping the phase neglects information which might be required for the classification of characters.

A log-polar mapping in combination with the Fourier transform is used to get scale and rotation invariance. The log-polar mapping $\mathcal{L}$ of Cartesian coordinates $x$ and $y$ is given by

$$\begin{aligned} x &= e^{\mu} \cos \Theta \\ y &= e^{\mu} \sin \Theta \end{aligned} \tag{5}$$

which has the following properties:

- Scaling is converted to translation

$$(\rho x, \rho y) \Leftrightarrow (\mu + log\rho, \Theta) \tag{6}$$

- Rotation is converted to translation

$$\begin{aligned} &(x\cos(\Theta+\delta) - y\sin(\Theta+\delta), x\sin(\Theta+\delta) + y\cos(\Theta+\delta)) \Leftrightarrow \\ &(\mu, \Theta+\delta) \end{aligned} \tag{7}$$

So the log-polar mapping converts scaling and rotation to a translation. And because the magnitude of the Fourier transform is translation invariant, a combination of both can be used to extract features which are invariant under rotation, scaling and translation.

*Proof.*

$$\begin{aligned} I_1 &= [|\mathcal{F}| \circ \mathcal{L} \circ |\mathcal{F}|] f(x_1, x_2) \\ I_2 &= [|\mathcal{F}| \circ \mathcal{L} \circ |\mathcal{F}| \circ \mathcal{R}(\Theta) \circ \mathcal{S}(\rho) \circ \mathcal{T}(\alpha, \beta)] f(x_1, x_2) \\ &= [|\mathcal{F}| \circ \mathcal{L} \circ \mathcal{R}(\Theta) \circ |\mathcal{F}| \circ \mathcal{S}(\rho) \circ \mathcal{T}(\alpha, \beta)] f(x_1, x_2) \\ &= [|\mathcal{F}| \circ \mathcal{L} \circ \mathcal{R}(\Theta) \circ \mathcal{S}(\frac{1}{\rho}) \circ |\mathcal{F}| \circ \mathcal{T}(\alpha, \beta)] f(x_1, x_2) \\ &= [|\mathcal{F}| \circ \mathcal{L} \circ |\mathcal{F}|] f(x_1, x_2) \\ &= I_1 \end{aligned} \tag{8}$$

□

Remark: Computing the Fourier transform of a log-polar mapped image is equivalent to taking the Fourier-Mellin transform which is given by

$$\mathcal{F}_{\mathcal{M}}(k_1, k_2) = \int_{-\infty}^{\infty} \int_{0}^{2\pi} f(e^{\mu} \cos \Theta, e^{\mu} \sin \Theta) e^{i(k_1 \mu + k_2 \Theta)} \mathrm{d}\mu \, \mathrm{d}\Theta \tag{9}$$

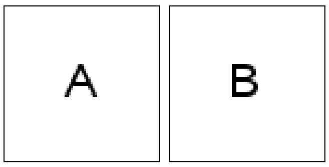
So

$$|\mathcal{F}| \circ \mathcal{L} \circ |\mathcal{F}| \equiv |\mathcal{F}_{\mathcal{M}}| \circ |\mathcal{F}|. \tag{10}$$

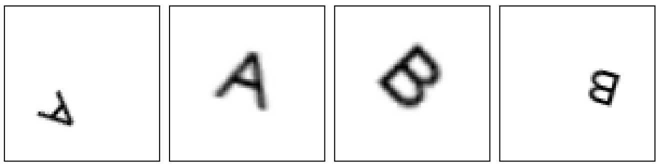
The Fourier-Mellon transform is not explicitly used as computing the Fourier transform is in general more efficient.

## Data set

The data set with two characters was generated from the following set of images



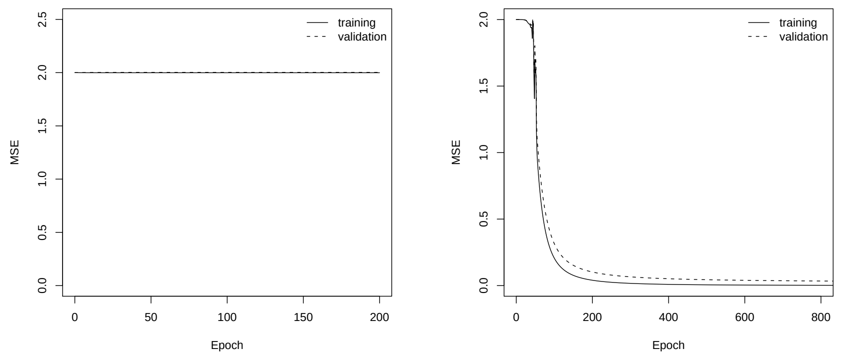
which was later extended to ten characters. The size of the images was chosen to be $64 \times 64$ so that a fast Fourier transform can be used directly (fft requires the length of the input to be a power of two). Sample images from the training set are shown in the following:



The images were either randomly translated and scaled or translated and rotated to ensure that the characters do not lie outside the image. One can already see that scaling and rotation adds a lot of noise to the image which needs to be filtered in the classification step.

## Learning from invariant features

Firstly, a training set containing randomly rotated, scaled and translated images of two characters was used. When training a neural network it is important to use a validation set on which the learned model is tested to ensure good generalisation. Otherwise it might happen that the network just memorises all images. Therefore a training and validation set is generated, each containing 100 images. Figure 1 shows the mean squared error of the learning process. In the left diagram the network is trained with the raw pixel images as input. The algorithm does not converge which shows that the data needs to be preprocessed for successful training. In the right diagram the invariant features were used for

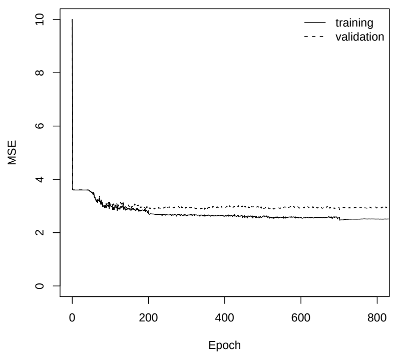


(a) raw pixels (b) invariant features

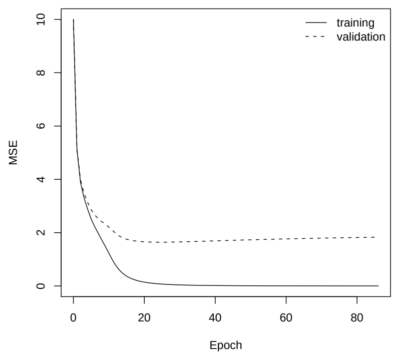
**Figure 1:** Mean squared error of backpropagation learning from (a) raw pixels and (b) invariant features. Network topologies: (a) $4096 \times 10 \times 2$, (b) $1024 \times 5 \times 2$

the training and validation set. The network is able to classify both sets with 100% accuracy after very few learning epochs. It consists of 1024 input neurons ($64^2$ pixels where negative frequencies of the Fourier transform can be dropped), one hidden layer of five neurons (convex problem) and one output layer of two neurons. The output of the network is encoded such that each neuron represents a single character. For instance if an image containing an $A$ is presented then the first neuron gets activated and all others (in this case only the second one) are deactivated. This coding scheme is also used for the extended data set of 10 characters. Instead one could also think of a binary encoding which would decrease the number of required output neurons.

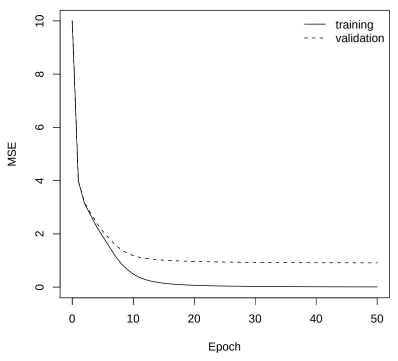
Results of training on the 10 character data set is presented in Figure 2. The set contains 1000 images where 20% are used for validation. In the first trial the algorithm does not converge well (see Figure 2(a)) where the accuracy drops to



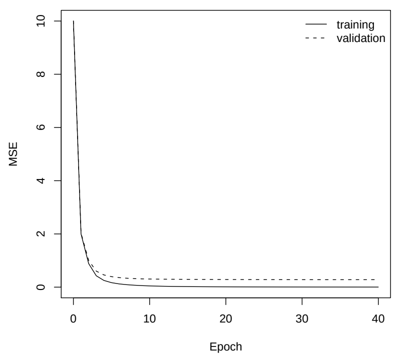
(a) invariant features (1000)



(b) invariant features (1000)



(c) invariant features (2000)



(d) invariant features (10000)

**Figure 2:** Mean squared error of backpropagation learning from invariant features. In figure (a) the features were not scaled whereas in (b), (c), and (d) they were. Also the size of the training set was gradually increased. Network topology: $1024 \times 15 \times 2$

1/3. Training a support vector machine (SVM) however reveals that the data is separable. With a linear kernel the SVM reaches an accuracy of 76% and with RBF kernel an accuracy of 80%. This shows that the data is almost linearly separable, which is because of the high dimensional feature space (as discussed in class). In the second trial (see Figure 2(b)) the data is scaled such that every point lies in the intervall $[0,1]^{1024}$. Scaling the data ensures that the learning rate has the same effect (relative step size) in all dimensions. The diagram shows that the learning algorithm converges much better on the training set but on the validation set the accuracy is still at only 60% (bad generalisation). Reasons for overfitting are either too many neurons, not enough data, or loss of information. With many dimensions it is important to also have many training samples so the boundaries of the classes are well established. Therefore the size of the training and validation set is doubled in the next trial shown in Figure 2(c). The accuracy increases to 79% on the validation set. Figure 2(d) shows results of learning with 10000 images with an accuracy of 93.6%.

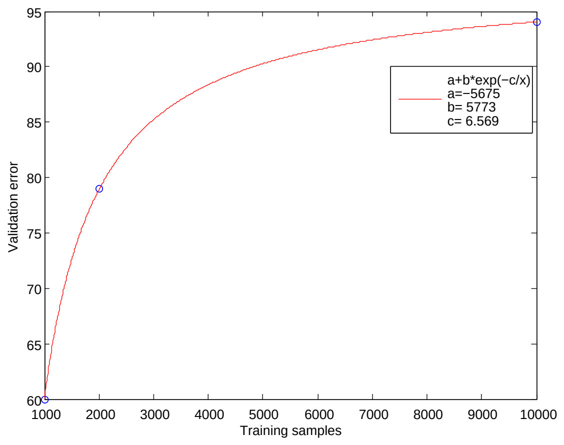
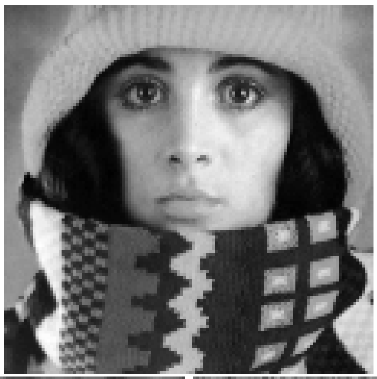


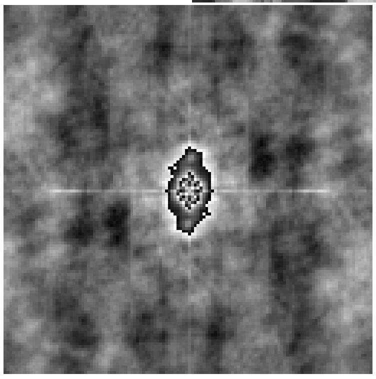
**Figure 3:** Exponential model fitted to the resulting accuracies vs. training samples.

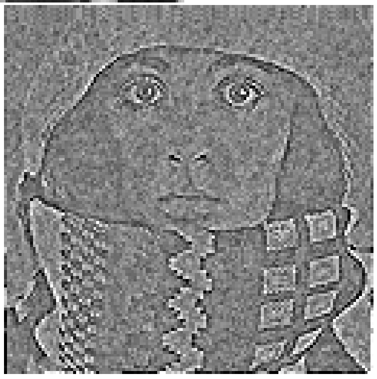
An exponential model is fitted to the results to get an idea of what the maximum accuracy might be as the number of images is further increased (see Figure 3). The model suggests that if number of samples goes to infinity than an accuracy of 98% might be reached (of course this model should be tested by for instance training a network with 1500 and 6000 images).

## How much information is lost?

The following three images give an intuition of what information is encoded in the magnitude and phase of the Fourier coefficients. The original image (upper one) is transformed into Fourier space and afterwards transformed back where in the lower left image only the magnitude was used and in the lower right one only the phase.







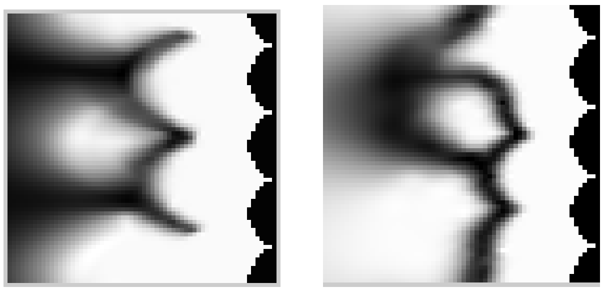
Neither image is sufficient to reconstruct the original one. The phase seems to encode contours of the image (this is used in computer graphics for edge detection) whereas hardly any resemblance is given in the reconstruction from the magnitude. This however does not prove that the magnitude contains less information.

## Implementation of the log-polar mapping

The implementation of the log-polar mapping is complicated because discrete images are used. In a log-polar coordinate system the size of the pixels increase the further away the pixel is located from the image center. The implementation is sketched in Algorithm 1. Two log-polar mapped images are shown below:

**Algorithm 1** Computing the log-polar mapping

```
 1: Given an image $f(x_1, x_2)$ of size $N_1 \times N_2$
 2: Let $r_{max} = \sqrt{(N_1+1)^2/4 + (N_2+1)^2/4}$          ▷ Maximum radius
 3: Let $\Delta_\Theta = \frac{2\pi}{N_2}$                       ▷ Θ step size
 4: Let $\Delta_r = \frac{\log r_{max}}{N_1}$                    ▷ r step size
 5: Create an array C of size $N_1 \times N_2$
 6: for $i \leftarrow 1, ..., N_1$ do
 7:     for $j \leftarrow 1, ..., N_2$ do                        ▷ For all pixels of the image
 8:         Let $r = e^{i\Delta_r}$
 9:         Let $\Theta = j\Delta_\Theta$                        ▷ Compute log-polar coordinates
10:         Let $x = r\cos\theta + \frac{N_1+1}{2}$
11:         Let $y = r\sin\theta + \frac{N_2+1}{2}$              ▷ Compute corresponding Cartesian coordinates
12:         $C(i,j) \leftarrow (x,y)$
13:     end for
14: end for
15: $C$ is a log-polar map containing Cartesian coordinates
16: which can be used to create an interpolated image
```



## The backpropagation algorithm

In the backpropagation algorithm a model (the neural network) is fitted to a set of training samples by adjusting the weights of the network. The network is given by a set of neurons

$$a_i = \sigma(\sum_{j \in Pred(i)} w_{ij} a_j), \tag{11}$$

where $w_{ij}$ is the weight from neuron $j$ to neuron $i$, $x_j$ the $j$th input to neuron $i$, and $\sigma \in \{x \mapsto \frac{1}{1+e^{-x}}, x \mapsto \frac{e^x - e^{-x}}{e^x + e^{-x}}, \dots\}$ a nonlinear activation function. $a_1$

is usually set to 1 to account for a bias term. The network is ordered in layers where connections are purely feed-forward (to avoid complex dynamics).

Given a set of training samples $\mathcal{D}$. An error function can be defined as

$$E_{SSE}(\boldsymbol{w};\mathcal{D}) = \frac{1}{2}\sum_{i=1}^{N} ||\boldsymbol{y}(\boldsymbol{x}_i;\boldsymbol{w}) - \boldsymbol{d}_i||^2 \tag{12}$$

where $N = |\mathcal{D}|$ denotes the number of training patterns, $\boldsymbol{y}(\boldsymbol{x}_i;\boldsymbol{w})$ the network output of input pattern $\boldsymbol{x}_i$ with weight vector $\boldsymbol{w}$, and $\boldsymbol{d}_i$ denotes the desired output value.

Training the network is equivalent to minimising the error $E_{SSE}(\boldsymbol{w};\mathcal{D})$ with respect to the weights. A global minimum can in general not be computed, but a gradient descent approach can be used to find a local optimum. Each weight is updated according to

$$\Delta w_{ij} = -\eta \frac{\partial E_{SSE}(\boldsymbol{w};\mathcal{D})}{\partial w_{ij}}, \tag{13}$$

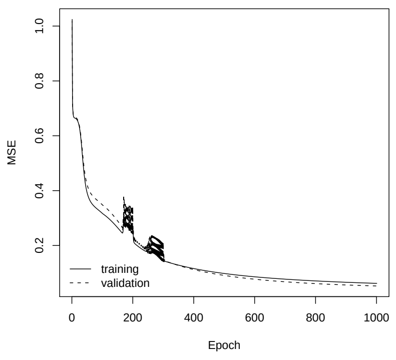
i.e. the negative gradient of the error function is used where $\eta$ denotes the learning rate. The chain rule is used to propagate the error back from the output layer towards the input layer.

The artificial neural network and backpropagation algorithm are both implemented in Lisp. The learning error is always stated as mean squared error (MSE):

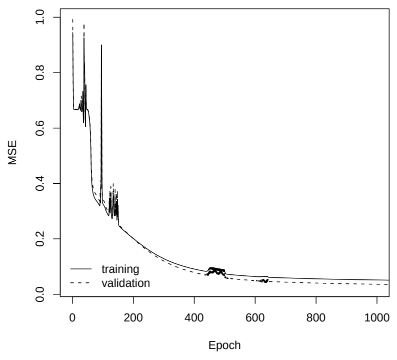
$$E_{MSE}(\boldsymbol{w};\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} ||\boldsymbol{y}(\boldsymbol{x}_i;\boldsymbol{w}) - \boldsymbol{d}_i||^2. \tag{14}$$

The implementation was first tested on the Pima Indians data set. The task is to classify iris plants according to sepal width/length and petal width/length. There are three classes namely *Iris Setosa*, *Iris Versicolour*, and *Iris Virginica* which were encoded as $(1,0,0)^T$, $(0,1,0)^T$, and $(0,0,1)^T$, respectively. One of the three classes is linearly separable, the other two are not. Results of training are plottet in Figure 4. To let the algorithm converge, the learning rate has been manually decreased over time. The results show a good generalisation in the first three trials. The last test run shows overfitting caused by too many layers and neurons.

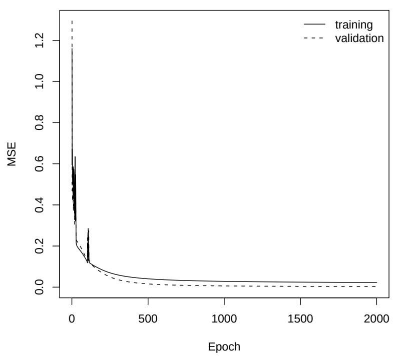
Standard backpropagation is a simple Greedy search algorithm which has a lot of drawbacks. The most obvious is, that it gets easily stuck in local optima,



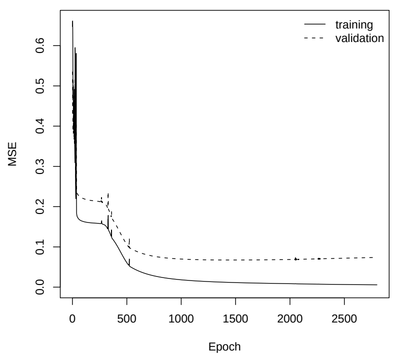
(a)



(b)



(c)



(d)

**Figure 4:** Iris dataset. (a) ANN with one hidden layer of three neurons. (b) ANN with two hidden layers, six neurons in the first and four in the second. (c) ANN with one hidden layer of 40 neurons. (d) ANN with two hidden layers of 20 each.

other drawbacks are illustrated in Figure 5.

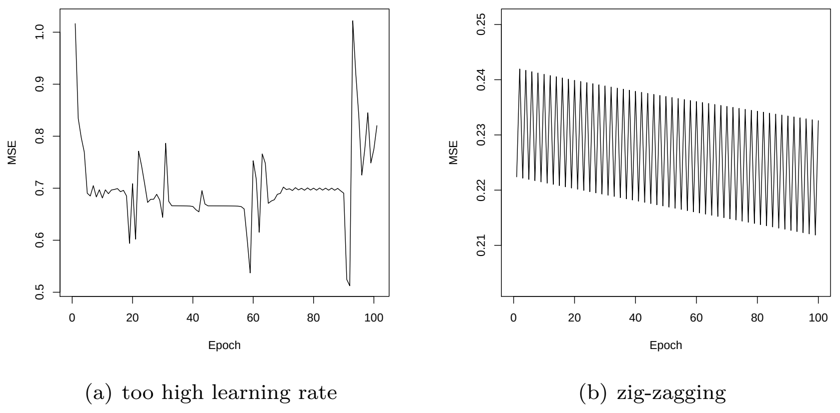


**Figure 5:** Drawbacks of vanilla backpropagation. (a) Firstly the learning rate is suitable and the error decreases, but after very few epochs the error begins to jump because the learning rate is too high. (b) The algorithm starts to jump back and forth over a local minimum (zig-zagging).

## Discussion

The feature extraction method presented in this project was successfully used for the classification of characters. With only two characters an accuracy of 100% is easily reached. For ten characters the classification task becomes more difficult. It is necessary to scale the data such that the learning rate of the backpropagation algorithm has the same effect on all dimensions of the feature space. Also the size of the training set needs to be sufficient because of the high dimensionality of the feature space and the noise that is introduced by the transformation of images and the log-polar mapping. An accuracy of 93.6% was reached on a set of 10000 images. A model fitted to the training results however suggests that more images would further improve the classification accuracy to a maximum of 98%.

A drawback of this feature extraction method is the high dimensional feature space. The neural network needs as many input neurons as there are dimensions in feature space. Learning therefore takes a lot of time and resources, also because many training patterns are required. On the other hand, the more dimensions the feature space has, the more probable it is that the task is linearly separable. It was tested as to whether it is possible to reduce the number of dimensions with a PCA or ICA. It turned out that all dimensions are important for an accurate classification as the accuracy went gradually down the more dimensions were removed.

With 10 characters a perfect classification was not achieved but there is still room for improvement. For instance the log-polar mapping has a minimum radius and all frequencies below this radius are neglected. Also the size of pixels increases with distance to the center meaning that the quality of the log-polar mapping is low in the periphery. A mapping where all pixels are equally sized might significantly improve performance. Besides, it would be interesting to compare this method to others that use phase information instead of magnitudes.

# Appendix

**Listing 1:** Backpropagation Algorithm

```lisp
(defun update-delta-weights (neuron prev-layer learning-rate)
  "Update all delta weights of a neuron."
  (let ((prev-layer-output (layer-output prev-layer))
        (part-error (part-error neuron)))
    (loop for i upto (1- (length prev-layer-output)) do
          (setf (aref (delta-weights neuron) i)
                (+ (aref (delta-weights neuron) i)
                   (* learning-rate part-error (aref prev-layer-output i)))))))

(defun update-part-error (neuron index next-layer)
  "Update partial error of a neuron (for all neurons in hidden layers)."
  (let ((sum (loop for i-neuron from 1 to (size next-layer)
                   for neuron = (get-neuron next-layer i-neuron)
                   for part-error = (part-error neuron)
                   for weight = (aref (weights neuron) index)
                   summing (* weight part-error)))
        (neuron-output (output neuron))
        (transf (transfer-function neuron)))
    (setf (part-error neuron) (* (call-deriv transf neuron-output) sum))))

(defun backprop-error (network target output learning-rate)
  "Propagate error back through the network."
  ;; calculate partial error on output layer
  (let ((output-layer (get-output-layer network)))
    (loop for i from 1 to (size output-layer)
          for neuron     = (get-neuron output-layer i)
          for derivation = (call-deriv (transfer-function neuron) (aref output (1- i)))
          for deviation  = (- (aref target (1- i)) (aref output (1- i)))
          do (setf (part-error neuron) (* deviation derivation))))
  ;; propagate error back through the network
  (loop for i-cur-layer from (1- (length (layers network))) downto 1
        for i-prev-layer = (1- i-cur-layer)
        for cur-layer    = (get-layer network i-cur-layer)
        for prev-layer   = (get-layer network i-prev-layer) do
        ;; update delta weights on current layer
        (loop for i-neuron from 1 to (size cur-layer)
              for neuron = (get-neuron cur-layer i-neuron)
              do (update-delta-weights neuron prev-layer learning-rate))
        ;; update partial errors on previous layer (when not input layer)
        (when (> i-prev-layer 0)
          (loop for i-neuron from 1 to (size prev-layer)
                for neuron = (get-neuron prev-layer i-neuron)
                do (update-part-error neuron i-neuron cur-layer)))))

(defun update-weights (network)
  "Copy all delta weights to real weights on all neurons in the network, after-
   wards reset all delta weights to zero."
  (loop for neuron across (neurons network) do
        (loop for i upto (1- (length (weights neuron))) do
             (setf (aref (weights neuron) i) (+ (aref (weights neuron) i)
                                                (aref (delta-weights neuron) i)))
             (setf (aref (delta-weights neuron) i) 0))))

(defmethod backprop-batch ((algo learning-algorithm) &key
                           (learning-rate *learning-rate*)
                           (temp          *tempcontrol*))
  (loop for pattern = (next-training-pattern (data algo)) while pattern do
       (let* ((input  (first  pattern))
              (target (second pattern))
              (output (propagate (network algo) input)))
         (backprop-error (network algo) target output learning-rate temp)))
  ;; adjust weights here, that is, copy partial weight changes to real weights
  (update-weights (network algo))
  (reset-training-ptr (data algo))
  (incf-epoch algo))
```

```

(defmethod backprop-incremental ((algo learning-algorithm) &key
                                 (learning-rate *learning-rate*)
                                 (temp          *tempcontrol*))
  (loop for pattern = (next-training-pattern (data algo)) while pattern do
       (let* ((input  (first  pattern))
              (target (second pattern))
              (output (propagate (network algo) input)))
         (backprop-error (network algo) target output learning-rate temp)
         (update-weights (network algo))))
  (reset-training-ptr (data algo))
  (incf-epoch algo))
```