VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

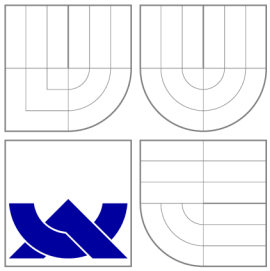
ASL FINGERSPELLING RECOGNITION USING SLOW FEATURE ANALYSIS

BAKALÁRSKÁ PRÁCA
BACHELOR'S THESIS

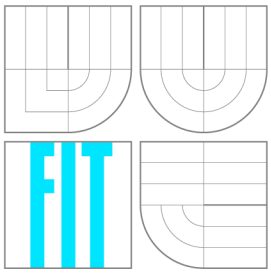
AUTOR PRÁCE
AUTHOR

MARTIN WINKLER

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ROZPOZNÁVANIE ASL HLÁSKOVANIA POUŽITÍM SLOW FEATURE ANALYSIS

ASL FINGERSPELLING RECOGNITION USING SLOW FEATURE ANALYSIS

BAKALÁRSKÁ PRÁCA
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN WINKLER

VEDÚCI PRÁCE
SUPERVISOR

Ing. BURGET LUKÁŠ, Ph.D.

BRNO 2014

Abstrakt

Táto práca popisuje proces testovania slow feature analysis ako metódy, ktorá extrahuje robustné črty z komplexných obrazových dát americkej znakovej reči. Za účelom testovania bol vytvorený systém v programovacom jazyku python, ktorý zjednodušuje testovanie a ponúka bohatú škálu meniteľných parametrov aby umožnil užívateľovi rôzne testy za účelom zistenia nakoľko použiteľná je táto metóda na klasifikáciu a rozpoznávanie gest rúk. Teoretická časť predstaví slow feature analysis, diskutuje o štruktúre systému a popisuje dáta na ktorých bude metóda pozorovaná. V praktickej časti je metóda podrobená analýze úspešnosti na videných a nevidených rečníkoch, jej schopnosť adaptovať sa na vyšší počet gest a zaujímavé formátovanie dát v pokuse vylepšiť jej úspešnosť.

Abstract

This work describes the process of testing slow feature analysis as a method of extracting rhobust features from complex image data of american sign language. For purposes of testing a system in python is created that facilitates test runs and offers rich scale of changable specifications to allow the user run various tests in order to determine how viable the method is for classification and recognition of hand shapes. The theoretical part introduces the slow feature analysis, discusses the structure of the system and describes the dataset on which the method is to be observed. In practical part the method was subjected to performance analysis on seen and unseen speakers, its viability with higher number of gestures and some interesting input data formatting in attempt to improve the performance.

Kľúčové slová

Analýza pomalých komponentú, hierarchická architektra, MDP - Toolkit pro modulárne spracovanie dát, vlastné čísla a vektory, singulárny rozklad, ASL - Americká znaková reč

Keywords

Slow feature analysis, hierarchical architecture, MDP - Modular toolkit for Data Processing, eigenvalues and eigenvectors, singular value decomposition, ASL - American sign language

Citácia

Martin Winkler: ASL Fingerspelling Recognition Using Slow Feature Analysis, bakalárská práca, Brno, FIT VUT v Brně, 2014

ASL Fingerspelling Recognition Using Slow Feature Analysis

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety under the supervision of Ing. Lukáš Burget, Ph.D. and Nicolas Pugeault, Dr. Rer. Nat. . I have faithfully cited all sources used in the thesis.

.....
Martin Winkler
August 7, 2014

Acknowledgments

First and foremost I would like to thank my english supervisor Nicolas Pugeault, Dr. Rer. Nat. for insightful advice,friendly approach that kept me on the right track and his will and genuine interest to consult new ideas and improvements at any time. Next I would like to thank my czech supervisor Ing. Lukáš Burget, Ph.D. not only for his formal correction of this document but mainly for introducing me to machine learning and recognition as without his enthusiasm and will to pass his knowledge onto others I highly doubt that I would have chosen and pursued this thesis.

Contents

1 Preface **3**

2 Slow feature analysis **4**
2.1 The slowness principle . . . 4
2.2 The optimization problem . . . 4
2.3 The algorithm . . . 5

3 Hierarchical network **7**
3.1 Relation to Slow feature analyis . . . 7
3.1.1 Overlapping . . . 8

4 Dataset **9**
4.1 Origin . . . 9
4.2 Data volume . . . 9
4.3 Variance accross letter . . . 9
4.4 Variance accross speakers . . . 9
4.5 Background noise . . . 10

5 System design **12**
5.1 Goal formulation . . . 12
5.2 General outline . . . 12
5.3 Hierarchical network specifications . . . 12
5.4 Image preprocessing . . . 13
5.5 MDP Toolkit . . . 14
5.6 Nodes . . . 14
5.6.1 Input and output formatting . . . 14
5.6.2 SFANode . . . 15
5.7 The core system overview . . . 15
5.8 Cell composition . . . 15

6 System implementation **17**
6.1 Segmentation . . . 17
6.2 Preprocessing with PCA . . . 18
6.3 Hierarchy of SFA units . . . 18
6.4 Generalized eigenvalue problem . . . 19
6.4.1 Removing insignificant dimensions . . . 20
6.4.2 Singular value decomposition . . . 20
6.4.3 Accuracy comparison . . . 22

7 Testing the model **23**
7.1 Trained speaker . . . 23
7.1.1 Trainset size dependency test . . . 23
7.1.2 Performance accross varying number of letters . . . 24
7.1.3 Extracted feature analysis . . . 25
7.2 Unseen speaker . . . 26
7.2.1 Performance changes based on number of training signers . . . 27
7.2.2 Speaker detection . . . 28
7.2.3 Speaker ordered data . . . 28

8 Conclusion **30**

Chapter 1

Preface

The problems of classification and recognition are becoming more and more popular. The aplications ranging from QR code reading, through OCR, face recognition and voice recognition have flooded the market and people are starting to move away from ever so restricting keyboards and are actively searching for alternate and more natural ways to communicate with machines.

Gesture recognition is one of the possible replacements of traditional input as it is easy to perform for everyone and much closer to innate human behavior than typing. Furthermore the gestures are easier to remember than shortcut combinations as they involve full body experience. There are many gesture systems out of which one of the most widely used is unarguably the sign language which however, brings small variations among letters and great variation among speakers therefore being one of the more difficult ones to recognize.

Main goal of this thesis is to observe and evaluate how does one of the perspective methods, slow feature analysis, cope with this problem. In the practical part a system for learning and testing the extracted knowledge oF features using both seen and unseen speakers will be created, implemented and conclusions of the usability of the method will be drawn using variety of tests.

Chapter 2

Slow feature analysis

2.1 The slowness principle

Visual data interpreted by brain are received by cones and rods in the eyes. The inputs alter constantly at very high rate. Color of an object is a manifestation of the light reflected by the object. Any motion of an object causes displacement which in turn alters the angle and intesity of the light that hits the surface of the object. Even minimal change in position causes great variation of the input signal that gets read by our eyes.

How do we then see the objects, their relative positions and movements? The brain bases its representation on the fact that objects have common structures which then lead to hidden similarities in the visual input.[6] Based on those similarities the brain processes the signal and finds patterns that are changing slowly and that we perceive as moving. Those objects change position in orders of seconds but the visual input varies much faster.

The idea behind the slowness principle is to process the fast changing inputs to find slowly varying reasons that alter the visual input signal. For our example this would be trying to find and identify the object that is moving thereby changing the light reflects off of it causing the signal processed by our eyes to change.[6]

2.2 The optimization problem

The goal of the Slow feature analyis, from now on referred to as SFA, is to find a function that when received the visual input signal the output signal generated will vary as slow as possible.[3, 6] However, constant solution, no variation at all, is not sought after and hence the specification is - Try to find a function that alters the input signal in such a way that the output is a signal with slowest possible variation that is not constant and still carries information.

The formulas in this section were taken from [3]. Given the input signal $x(t) = [x_1, x_2...x_t]$, function $g(a)$ new singal $y(t) = [y_1, y_2...y_t]$ is calculated as:

$$y_j(t) = g_j(x(t)) \tag{2.1}$$

We want to assure that the the output signal $y(t)$ is as slowly varying as possible which means that for each $j \in 1...j$ is the following value minimal:

$$\Delta j := \Delta y_j := \langle \dot{y}_j^2 \rangle \tag{2.2}$$

Constraints need to be listed as the trivial constant solution is not what we are looking for.

$$\langle y_j \rangle = 0 \tag{2.3}$$

Describes zero mean that helps to avoid the unwanted constant solution.

$$\langle y_j^2 \rangle = 1 \tag{2.4}$$

Unit variance also helps with not finding constant solution.

$$\forall j' < j \langle y_{j'} y_j \rangle = 0 \tag{2.5}$$

Makes sure there is no correlation and orders them so that the slowest component is first, second slowest second and so on and so forth. The resulting optimization however is a problem of variational calculus that is very difficult to solve. It is possible to simplify the problem by restricting the components of the function $g(v)$ to be only linear combination of a finite set of nonlinear functions. This restriction applies for the algorithm outlined in the next section.

2.3 The algorithm

At start we only have the input signal that $x(t)$ has I dimensions. We are searching for the input-output function $g(x)$ specified in the optimization problem. Due to the restrictions we decided to put in place in order to simplify the problem the input-output function contains g_j components which are a weighted sum over a set of K nonlinear functions.[3]

$$h_k(x) : g_j(x) := \sum k = 1 K w_{jk} h_k x \tag{2.6}$$

Where the w_j are the weights that need to be learnt. Applying each of the $h_1 to h_k$ on the input signal the expanded signal $z(t)$ is created (Step B of figure 2.3:

$$z(t) = h(x(t)) \tag{2.7}$$

The new input output function to optimize then becomes:

$$\Delta(y_j) = \langle \dot{y_j^2} \rangle = w_j^T \langle \dot{z} \dot{z}^T \rangle w_j \tag{2.8}$$

The expanded signal z signal needs to meet the constraints specified by 2.3, 2.4 and 2.5. All of the conditions are automaticaly met if the wieghts w are orthonormal as stated in[3] and proven by following equations.

$$\langle y_j \rangle = w_j^T \langle z \rangle = 0 \tag{2.9}$$

Where $\langle z \rangle$ is 0

$$\langle y_j^2 \rangle = w_j^T \langle z z^T \rangle w_j = 1 \tag{2.10}$$

where $\langle z z^T \rangle$ is an identity matrix

$$\forall j' < j \langle y_{j'} y_j \rangle = w_{j'}^T \langle z z^T \rangle w_j = w_{j'}^T I w_j = 0 \tag{2.11}$$

The signal needs to be normalized which is done in the stage C of Figure 2.3 by getting the unit covariance matrix. This is done by applying the sphering matrix S on matrix

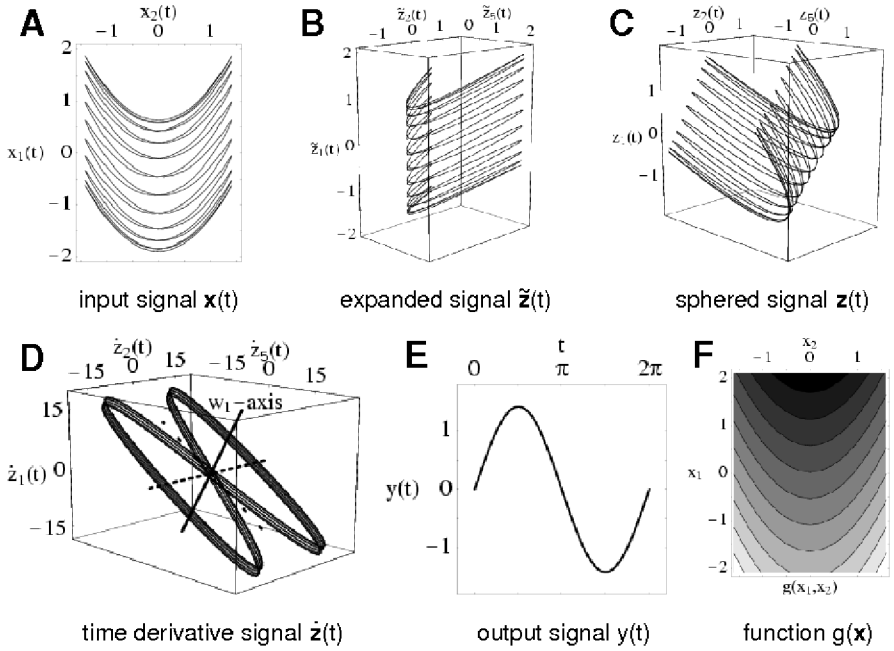


Figure 2.1: Hierarchical architecture demonstrating overlapping on bottom layer

$\langle \tilde{z}^T \rangle - \langle \tilde{z} \rangle$ The first component of the input-output function then is the normed eigenvector of the smallest eigenvalue of $\langle \dot{z}\dot{z}^T \rangle$ as it minimizes the $\Delta y1$ in our optimization equation. The other components are the eigenvectors of eigenvalues of the same matrix ordered from smallest to greatest.

Once the model has been trained and is tested the algorithm is done the same way as written above but the normalization has to occur with the same offsets and factors[3] as the training signal.

Chapter 3

Hierarchical network

Hierarchical network is such system where each but one unit of the system is subordinate to one or more units that are at higher level of the system. Due to the the most common shape of a hierarchical system it tends to be visualised as a pyramid.

Hierarchical structure has found many uses from power distribution to classification of species and nomenclature. In computer science it is used to categorize domains, packages and filesystems to mention a few. It is no surprise that this system also found its place in machine learning as it can describe increasing level of abstraction with respective levels. [7]

3.1 Relation to Slow feature analyis

The forshadowed capability to represent the higher level of abstraction with each layer of the hierarchy fits the needs of the slow feature analysis as more and more robust features are to be extracted by each step. The idea behind the system is to run the SFA in form of layers where each layer corresponds to one level of the hierarchy. The layers depict increasing abstraction of the features extracted from the SFA as the size of the receptive field grows with the layer number starting at zero at the bottom as can be seen in figure 3.1

The zeroth layer represents the image as a set of receptive fields, cells, each covering a certain segment of the image. From layer to layer multiple neighboring receptive fields are joined rendering a receptive field with greater area in respect to the processed image. The final layer containing only one cell is in fact a receptive field covering the entire image.

Thanks to the sequential training of the network's layers the function space that we are looking for the function in for can be much higher thanks to dimensionality reduction in each of the layers. The hierarchical approach therefore allows for expansion of the raw input data into function space of polynomials of degree of $q = a^k$, where a is the degree to which the signal is expanded in each layer and k is the number of layers of the system.

The output of each of the layers is the n of the slowest varying components of the input. Since each of the cells of higher layer joins the outputs of multiple cells of the lower layer the input is a set of outputs of covered cells from the previous layer. Putting the features of multiple receptive fields together this way allows to extract a relationship that cannot be extracted from either of the signals thanks to the polynomial expansion of the signal

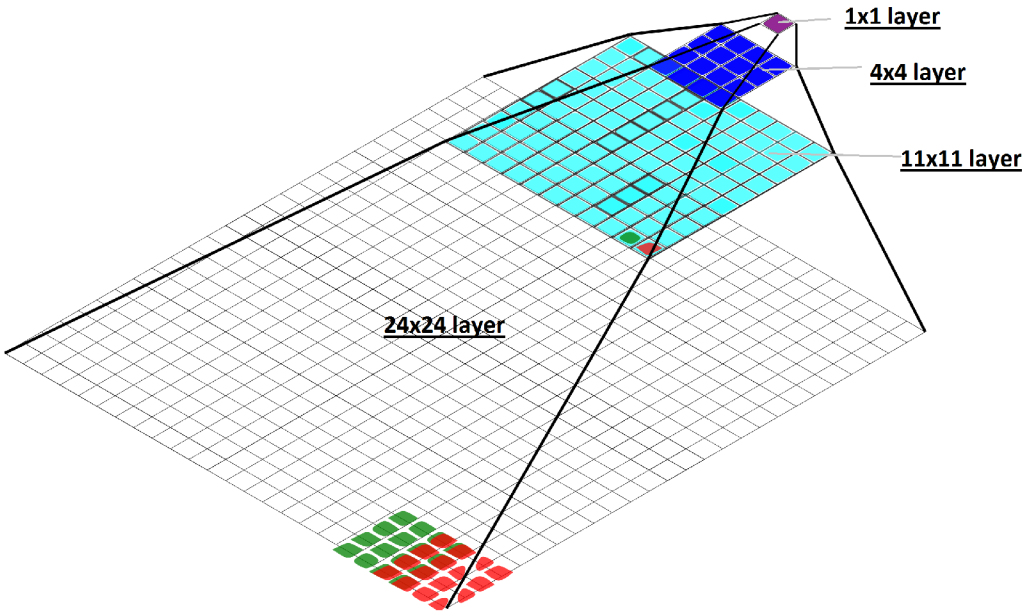


Figure 3.1: Hierarchical architecture demonstrating overlapping on bottom layer

3.1.1 Overlapping

Each of the segments represents a receptive field on the image. Overlapping of the segments provides continuos information flow of the feature from one segment to another. This fact improves the performance as the feature does not appear only in one segment but in all of the segments that have overlap and the feature present within this area.

Chapter 4

Dataset

4.1 Origin

The dataset on which the system was trained and tested contained exclusively images extracted using Kinect where the hands were detected and cropped using automated detection software. The images were not edited or preprocessed in any way before coming to the system and therefore were good representation of data that would be extracted in possible real life applications of the system.

4.2 Data volume

The volume of training and testing data is crucial to just about every classification and recognition method. The dataset consisted of 5 speakers where each speaker recorded 24 letters. The letters excluded from standard english alphabet were j and z as there is movement within the letter representation. The system is based around distinguishing shapes not trajectories of the letters and therefore the above mentioned letters are omitted from the classifiable set. Each speaker's video is represented in frames as recorded by the camera and the length of the video for each letter varies. The average number of images per letter regardless of speaker is 500.

4.3 Variance accross letter

To simulate real life tests the data captured has considerable variance among images even for the same speaker and the same letter. Each of the speakers was instructed not to hold the hand-shape for the entire span of the video but to perform rotations and translations of the hand. This way the system's resistance to translation and rotation would be tested. Furthermore the data recieves more realistic feel as the posture of the letter varies even within one speaker quite significantly on his current state similar to the speech.

4.4 Variance accross speakers

Just as in speach there are also significant differences among speakers in sign language as seen in 4.4. This shows as a problem in sign language recognition as the extent to which one letter's representation can vary is so great that it might well be considered a different letter completely. Alterations of the same letter range from the angle of wrist based on

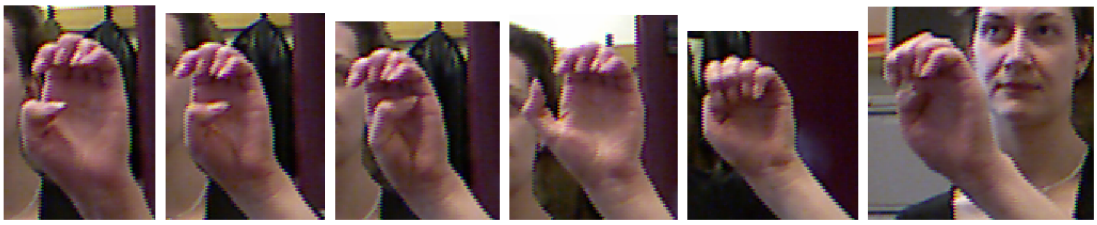


Figure 4.1: Variance accross letter E of speaker B - frames 2,81,101,124,193,230.

where the signer stands all the way to changes in positions of fingers - the very gesture defining features.

The reason why SFA is prone to failing due to big differences among representations of the same letter is because the sign is represented by the entire hand. As the hand holds the shape and moves as a unit the features describing it vary at roughly the same rate. An argument can be made that this leads to conclusion that change in the way the wrist and forearm and the fingers are held can come to a result where SFA fails to recognize the same letter from different speakers due to the difference in form beign too great and has no idea that the letter is the same as the algorithm is unsupervised.

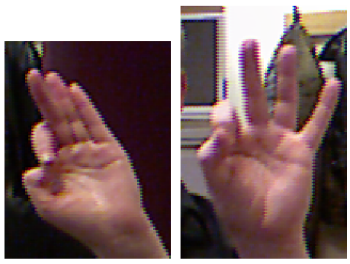


Figure 4.2: Very different representation of letter F by speakers B and E

4.5 Background noise

Even though the background noise is in most cases varying too rapidly for the SFA to be propagated as one of the slow features[7] due to the nature of the algorithm I found that there are cases when this is not true. One of such cases presented in data would be the speaker's head or face that was moving during the signing as well. As those movements are done by the same person and the movement is fluid the rate of change is generally very similar to that of the translation and rotation of the sign. If this is to happen the face must be presented in all of the images of one letter, this will bind the features of the letter with the features of the face as they vary at similar rates. ntly on his current state similar to the speech.

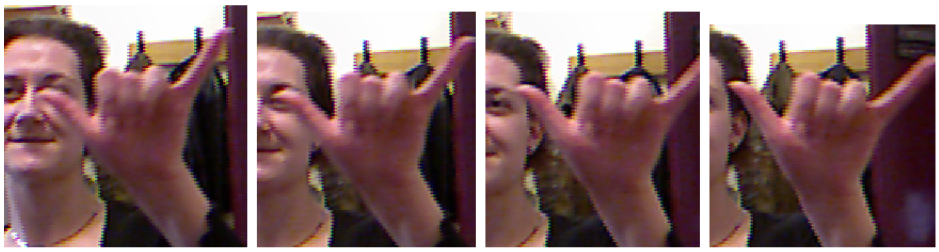


Figure 4.3: Head moving in opposite direction as the hand

Chapter 5

System design

5.1 Goal formulation

The main goal of the final application is to be able to allow to be capable of running various tests accross definable letters and speakers in order to supply enough data to draw conclusions about how reliably is SFA able to extract hand shape dependednt features. The application has to have a high degree of freedom in alteration of inner structure of the system to allow for parameter optimization by extensive user testing if required. Since image preprocessing, including dimensionality reduction on sgements with PCA, takes up significant part of the run time this part is to be separated from the network training and evaluation so that multiple network inputs can be prepared and serialized on the disk ahead of time. Using this separated approach the testing of the system will take shorter.

5.2 General outline

Based on the stated goal the system needs to be split into two distinct parts. First part is the file preprocessing and segmentation into the PCA units that extract given number of components. After the segments have been processed the output of has to be available in order to prepare data for multiple runs if needed as the output of this section is fed straight to the hierarchical network. Given that the system's second part's inputs can be precalculated a way to set the input externally has to be provided.

This system also needs to have two run modes - one for training the system and one for running tests on already trained model. In order to be able to project images onto the responses of the features and therefore be able to deduct roughly what is the feature learnt the system needs to be able to graph and store all the features accross all of the layers as well.

The output of the application is the list of all of the features that is then trained to any classifier if the results are to be compared. The labels for supervised classifiers for the output data are not to be generated due to expected memory issues with greater amounts of data.

5.3 Hierarchical network specifications

The structure of the hierarchical architecture was adopted from a similar experiment as it showed to be successful in the case it was used. [2] That being said the system was not tested

Table 5.1: Network architecture summary

Layer	Grid size	Each segment contains # of previous	Overlap
0	24x24	1x1	A 2 pixels
1	11x11	4x4	1/2 of the segment
2	4x4	5x5	2/5 of the segment
3	1x1	4x4	None

on completely real data but on manufactured ones with added noise. The structure of the architecture is following: The image is sliced to 576 segments, 24x24 already overlapping grid of 10x10 pixels. The layer on top of that consists of 11x11 grid where each of the cells is made of 4x4 cells of lower layer. This however doesn't mean that the receptive field increases 16times due to overlaps. The next layer consists of 4x4 grid where each of the cells covers 5x5 window of the previous layer. In the final layer all of the cells are joined into a single output with a receptive field covering the whole image. This arechitecture is visualised in 3.1.

In majority of the tests the number of features extracted from each of the cells was 10 excluding the top layer where the number was 20. The higher number of features in the highest layer was in order to preserve enough information about the signs as if one feature reacted only to one sign and there were no other features, reacting to more than 1 sign or so sign at all, the maxim number of classifiable signs would be equal to the number of features extracted. As majority of the tests were run with 10 or less letters due to memory restrictions 20 dimensions were considered sufficient. In tests where more dimensions were required information about the change will be provided.

The degrees of freedom mentioned in 5.1 at this stage are the number of segments in each layer and definable overlap.

5.4 Image preprocessing

As the images were automaticaly cut by external software the dimensions vary greatly. The detected hand was cut out and since there are letters that have both horizontal and vertical orientation the amount of dimensionality variation is understandable. It is not possible to simply crop the images to fit the minimal dimensions of one of the images processed. One of the reasons is that both test and train data would have to be provided for this operation, which is managable, but the second and a lot more severe one is that once cropped some of the images could lose too much valuable data as some of the signs are horizontal and other are vertical resulting in having only half a hand to train on if both types were mixed which generally is the case.

In order to preserve as much of the images and meet the needs of the system the images were scaled to 148x148 pixels. This number was not chosen at random but after careful calculation of the image segmentation in order to fit the zeroth layer perfectly so that no parts of the image are lost in the process. The segmenting parameters can be set by the user as far as the splitting and segmentation on the image is mathematicaly possible.

Concerning if the resizing of the images causes drops in performance I have strong reason to believe that the performance should stay almonst untouched. This is caused by

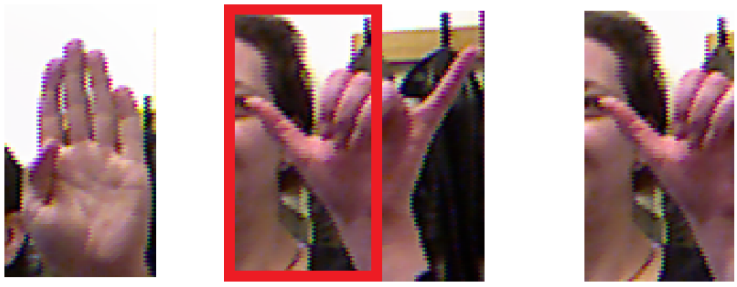


Figure 5.1: Cropping image of letter Y to have same dimensions as B

the fact that one letter will always have similar if not same aspect ratio and after resizing it is altered but it is altered by roughly the same way for all of the images of the given letter. From this it is apparant that the features will still be detectable altered but still detectable nevertheless as the way they were changed is uniform for each image of given letter.

As the features are not color dependent and we hope that the system learns the outlines of the gestures concentrating on edges as the siluette of the hand shape is what defines the letter there is no need to use all of the red, green and blue channels. Not only they are not that important for this test but would also greatly increase the size of the data processed taking toll on both time and memory consumption of the test runs. I therefore decided to convert all of the input images into grayscale before processing. This saves a lot of computational time and memory and allows to run more complex tests on the same hardware.

5.5 MDP Toolkit

MDP Toolkit, Modular toolkit for data processing referred as MDP from here on, is a data processing toolkit written in python.[8] It facilitates the use of data many data processing algorithms from which I notably used `PCANode` for principal component analysis, `SFANode` for slow feature analysis and `PolynomialExpansionNode` for expansion of the input singal.

5.6 Nodes

The power of the MDP is it's ease of use and uniform representation of algorithms and approaches as *Nodes* which share the exact same interface. The entire algorithm training or evaluation can the be done by one simple call. The methods for training the algorithm and testing it are *train* and *execute* respectively. Another great feature worth mentioning is the use of fortran functions for mathemathical operations to ensure the fastest possible calculations and guarantee high precision. The names of Fortran routines are automaticaly generated based on their availability in the host operating system so cross platform compatibility is assured.

5.6.1 Input and output formatting

The input and output format is essential for correct work with the MDP and points out certain restrictions worth mentioning. Both input and output are *mxn* matrices of data that are ordered such that the columns correspond to dimensions and the rows to observations.

In case of our problem that means that each row will contain information about one image and the columns will be the values of features from this image.

The restrictions for both `PCANode` and `SFANode` is the requirement to have the same number of dimensions, not observations, for each of the testing sets and this number has to be the same as the number of dimensions the node was trained on. This is one of the factors why are the images resized as we need to ensure that the dimensionality of the input is the same.

5.6.2 SFANode

The `SFANode` from the MDP I used substitues most of the SFA algorithm as it initializes the covariance and delayed covariance matrices during training and performs sphering and slowest component extraction during execution. It should be noted that for specific cases and purposes there is also a SFA2Node that also includes polynomial expansion in polynomial space of degree two. However, I wanted to give the user the freedom to choose what function space subset, defined by polynomial to used specified degree, is the signal expanded and thus chose to do expansion and slow feature analysis separately.

5.7 The core system overview

Due to the usage of MDP chosen language was python. The application was designed to run in separate stages each corresponding to the parts from the introduction. In order to create a capable and reusable tool the objecct design had to be considered. It had to be clear and each of the parts needed complete separation not only for aestethics but mainly because the data processed is generally of very high dimensionality and very hard to read. If one of the trasnformations was to be wrong it would be borderline impossible to detect it just by looking at extracted data.

The `Segmenter` provided list of files and image ranges is designed to extract the overlapping segments of the image accross all of the images and prepare them for PCA preprocessing part of the zeroth layer. Those data are then used by the `PCAobject` class to be processed by corresponding PCA unit extracting the user specified number of principal components as input for the hierarchical architecture of the SFAs. This architecture is composed of layers represented by the `Layer` object. The cells contained in the layer corresponding directly to receptive fields over the image are described by `Processing unit` object. The system as a whole is wrapped in the `SFA_system` object that provides the interface to configurate and run each of the stages.

5.8 Cell composition

Each of the cells that make up a layer are a system of their own. They are composed of multiple steps that simluate the SFA by the use of resources from MDP. The cell consists of input preprocessor in the form of a `SFANode` to reduce the dimensionality of the input signal, `PolyNomialExpansionNode` that performs polynomial epxansion of the input signal to subset of function space given by the degree of polynomial specified and another `SFANode` that finds the slowest varying components of the expanded signal. The final output of the node is then clipped at ± 10 - a value that I experimentaly found that most of the features never reach. This gets rid of spikes within the output that are generally not useful in feature's behaviour as they represent great response to a stimulus in short amount of time.

Once the features are exported into files the spikes also scale the maximum y coordinate of the graph and make the rest of the features reactions unreadable due to the great difference in values.

The representation of the steps in regards to the SFA algorithm is that the initial `SFANode` merely preprocesses the input, the `PolynomialExpansionNode` expands the input and the final `SFANode` spheres the signal and calculates the output signal.

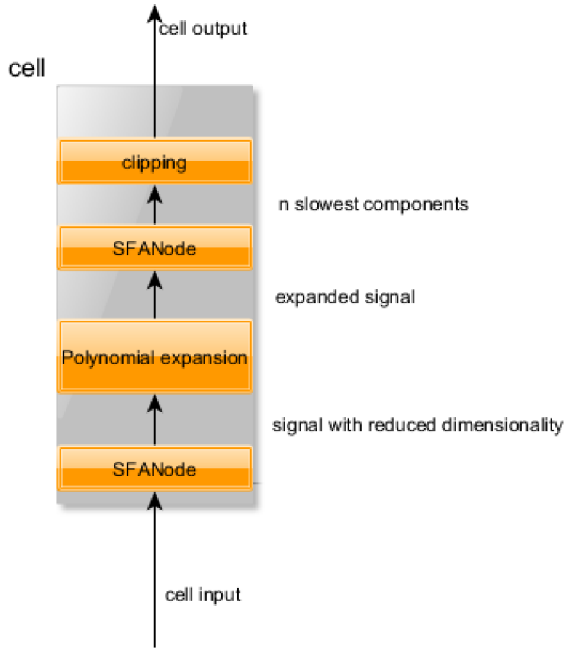


Figure 5.2: Cell structure

Chapter 6

System implementation

6.1 Segmentation

There were two distinct ways to approach the image segmentation regardless of the segment overlaping and segment size. Approach number one segments the entire image in at the same time and stores the output. In spite of great time performance this approach has proven utterly useless in respect to the amount of data used. The critical problem is that entire structure containing segments accross all of the files needs to be stored in memory at the same time. Considering the fact that the system usually runs on more than 5 letters and trains, at least in my tests, on more than 200 images the amount of memory used up by this operation is simply overwhelming.

On the other hand the second approach processes only one sgement at a time and thus decreases the total memory requirements of the network during this stage by

$$drop\_in\_memory\_requirements\_in\_percent = \frac{number\_of\_segments - 1}{number\_of\_segments} * 100 \qquad (6.1)$$

If default values are used there are 576 segments in each image by default so the memory requirement drops roughly by 99.83%. The disadvantage this effect produces is that the run time increases gradually based on the number of segments as well. This can be expressed as $number\_of\_segments * original\_run\_time$. In spite of this huge increase I used this segmentation method due to severe memory restrictions.

The class that takes care of this process is `Segmenter`. Upon reciving information on list of files folders and files to be segmented in format of array of folders, starting image number and ending image number, the segmentation process occurs one of the two above mentioned ways. In spite of rare cases that allow the usage of the first listed approach I still decided to include it as the speed increase is significant. There are no checks that prevent memory error however, so the system is prone to crashing in case of insufficient memory with generic python `Memory error`. The full image scanning and segmentation can be invoked by calling the `segment` method of a `Segmenter` instance. The second approach is equally easy to invoke by calling `get_one_segment` method. Both methods check if the required parameters have been set.

This is the first door into user model customization in order to perform alternate tests. The object contains information about number of segments to be created from the image in variable `segment_count`, segment size in pixels in `dim0` for height and `dim1` for width. Overlapping is introduced by instance variables `x_step` and `y_step` that describe by how many pixels does the shifting window move for each subsequent segment. The behavior is

such that the number of segments represents an *axa* grid. With that in mind I calculate the number of segments to be extracted for each row. Since the number of segments per collumn will be the same as the one per row the variables describing those values are bound to be equal. The reasoning behind the *axa* grid of the zeroth layer is that quantified amount of useful information is roughly the same due to letters being positioned both horizontally and vertically. The amount overlap is then defined by the relationship of the `x_step` and `y_step`, collectively referred to as step in this case, and dimensions of the extracted image. There are several possibile outcomes:

- $step < dimensions$ resulting in overlap
- $step = dimensions$ resulting in no overlap at all and no gaps between segments
- $step > dimensions$ resulting in gaps between consecutive segments
- $step <= 0$ causing an error and no segmentation

6.2 Preprocessing with PCA

Once the segments have been created dimensionality reduction is required before the signal can be passed to the hierarchical network. This dimensionality reduction is done on each segment separately and is approach dependant. If the first approach is chosen the entire segment array is extracted at the same time and the dimensionality reduction in PCA is done afterwards. However, if there is only one segment extracted at a time it is essential to pass it through PCA straight away as when next segment is read the old one is replaced in order to conserve memory.

The class that handles the PCA processing from creation of the units through processing to storing output is `PCAobj`. For easy use entire PCA preprocessing can be done by one method call - `do_0th_layer_PCA` in the wrapper object, `SFA_system`. The parametrization available at this stage is not great as the number of PCA nodes is directly dependent on the number of segments as each receptive field needs just one PCA node to decrease its output's dimensionality. The number of output dimensions of the PCA unit however, can be adjusted to a user defined value. It is important to note that polynomial expansion in the zeroth layer of the hierarchy might cause significant memory issues given high number of dimensions provided by this stage.

This is the first stage of the implementation where the difference between training and testing of the system can be noticed. During training the PCA units are trained and output is stored within the object in order to provide the user with chance to serialize the object to a disk for later analysis of output components accross the images. The training phase ends by testing the trained data so that inputs for the hierarchical network are generated.

During testing however, the data are merely passed through already trained units to generate outputs. Furthermore the output in case the unit is tested is returned outside of the object by the method thus conserving the original output for comparison.

6.3 Hierarchy of SFA units

As the core of the application this is where the slow feature analysis and continuous extraction of more abstract features describing greater receptive fields happens. As such this system is implemented over multiple classes each describing certain part.

The `Processing_unit` class represents a cell within the layer with contents as described in 5.8. The cells with some additional data contruct a layer that is represented by the `Layer` class. The wrapper for the whole system is `SFA_system` class. this is the main point with which the user interacts and on which methods correspoding to each of the main parts of the system are invoked.

The inputs passed from one layer to another are calculated first within units and then added to the layer output as a list of all unit outputs. This step is significant as they are effectively the features extracted at given layer of the system and need to be stored in order to be used as a resource for analysis of extracted features and their relations to the image sequence. Since the system calculates the layers iteratively from bottom up the output of one layer, stored in `output` instance variable, is directly fed to the input of the next layer, stored in `full_layer_input` variable. Upon change of the input for the layer all of the units remap their inputs based on the layer rules. This step is crucial in separating the training from testing as the units store their inputs. As the input to the layer changes during testing new input data has to be mapped on units. The default mapping based on the rules of the layer occurs already on their creation when `full_layer_input` is provided for training.

Mapping rules for the layers can also be provided by the user in order to map n cells from lower layer into one cell of higher level that is currently processed. Furthermore number of slow components can be specified for each cell but has to be the same accross the whole layer as the grid of cells representing a layer is homogenous. If the dimensions were to differ it would cause issues in the upper layers of the hierarchy as the same testing and training dimensions would not have been met (as required by 5.6.1).

6.4 Generalized eigenvalue problem

The slow feature extraction algorithm contains a generalized eigenvector problem that needs to be solved during sphearing part of the algorithm and is formulated as following:

$$Ax = \lambda Bx \tag{6.2}$$

Where A is the delayed covariance matrix, B is the covariance matrix, λ are the eigenvalues and x are the eigenvectors. In order to get the sphered signal we need to assure that the signal of which the eigenvalues and eigenvectors were calculated has unit covariance matrix. Hence we want to get the following equation.

$$A'x = \lambda Ix \tag{6.3}$$

The equation actually represents the specific eigenvalue problem that can be solved by eigenvalue decomposition. To achieve this knowing the fact that $AA^{-1} = I$ it is in theory sufficient to multiply the equation by B^{-1} hence by inverse of B.

$$B^{-1}Ax = \lambda BB^{-1}x \tag{6.4}$$

$$B^{-1}Ax = \lambda Ix \tag{6.5}$$

$$B^{-1}A = A' \tag{6.6}$$

This is the way the MDP `SFANodes` implement this algorithm as well. I found the problem to be that many of the segments of the image contain almost no relevant information and therefore their extracted components are small. After polynomial expansion those values

get even smaller. The result of those incredibly small values is that the values are either rounded down to 0 or even reach negative values due to computational errors that are system dependent. I realized that due to this development certain orders of matrix B became either rounded to 0 or even negative rendering the matrix singular and thus not being able to perform the inversion and the algorithm fails.

6.4.1 Removing insignificant dimensions

I came to conclusion that when the determinant is calculated to find out if eigenvalue of given order is positive the calculation multiplies number of elements of the matrix equal to the length of the diagonal. This means that if the matrix is of order greater than 308 it is enough for the values to be of 10^{-1} to underflow the minimum recognisable float value in python as its magnitude is 10^{-308} [1]. Understanding this fact, having matrices, after the expansion of course, of higher diensionality than 400 and seeing values smaller than 10^{-17} among them I understood that one of the reasons the eigenvalue is not positive is that those values underflow the float precission and are rounded to 0.

I decided to preprocess all of the matrices in such a way that I would remove the insignificant dimensions by either static or dynamic threshold, based on the dimensionality of the matrix. In the end, however its implementation failed to resolve the issue as the problem of singularity of the B matrix persisted. I understood that the reason the eigenvalue comes close enough to 0 to be rounded down is because of the numerical errors resulting from multiplication and subsequent substraction. This problem couldn't be solved by a threshold as the numbers can be of any size and differ only in the 10^{20} range for example.

6.4.2 Singular value decomposition

A great alternative for eigenvalue decomposition that under certain restrictions provides the same output is the singular value decomposition, referred as SVD from here on. Singular value decomposition breaks a matrix down into three matrices UsV^T. Matrix s is diagonal and contains singular values of the original matrix. Matrices U and V^T are orthonormal.

If SVD is applied on a symmetrical matrix its results are equivalent to the ones of eigenvalue decomposition. The eigenvalues are the diagonal of the s matrix and the eigenvectors are contained in both U and V^T matrices. Both covariance and delayed covatiance matrices are symmetric but once we find inverse of the matrix B and calculate the result on the left side we find out that the matrix is no longer symmetric and thus the SVD in its original form cannot be used anymore. However, there are ways to express generalized eigenvalue decomposition in terms of singular value decomposition.[4] The method is called simultaneous diagonalization and one such algorithm has already been developed for SFA for matlab as well.[4]

The algorithm is based on finding a transformation for the delayed covariance matrix B to become an identity matrix transforming the general eigenvalue problem into a specific case when the SVD can be applied.

We start with general eigenvalue problem:

$$Ax = \lambda Bx \tag{6.7}$$

We find and introduce the transformation S and its transpose S^T to both sides so that they can be turned into an identity matrix having no influence on the outcome at all. Given

that dot product is commutative their positioning within each of the sides is irelevant.

$$SAS^T x = \lambda SBS^T x \tag{6.8}$$

In order to be able to use the SVD the matrix decomposed needs to be symmetric and therefore we add the restriction to the S transformation. With the previous mentioned one the full set of conditions for S is as following.

$$SBS^T = I \tag{6.9}$$

$$SAS^T = \mathbb{I} \tag{6.10}$$

Where $\mathbb{I}$ represents diagonal matrix. If all of the above conditions are met we get a regular eigenvalue problem that is solvable by the SVD and the solution is numericaly stable. The goal is to find transformation S so that it meets the conditions. We start from eigenvalue decomposition of B as it yields a diagonal matrix of eigenvalues that can be easily converted to an identity matrix.

$$FBF^T = \Lambda_b \tag{6.11}$$

Multiplying the equation by the inverse of Λ_b results in identity matrix on right side which is just what is required.

$$FBF^T \Lambda_b^{-1} = \Lambda_b \Lambda_b^{-1} \tag{6.12}$$

From this form we can derive the transformation to be $S = F\Lambda_b^{-1/2}$ by substituting into the original equation 6.8 we get the following.

$$F\Lambda_b^{-1/2} BF^T (\Lambda_b^{-1/2})^T = I \tag{6.13}$$

Since $\Lambda_b^{-1/2}$ is a diagonal matrix it is the same as its transform thus by rearranging we get:

$$FBF^T \Lambda_b^{-1} = I \tag{6.14}$$

We can then create the new transformed matrix for the eigenvalue problem as:

$$A' = SAS^T \tag{6.15}$$

We solve the eigenvalue problem by SVD again and get the correct egienvalues yet the eigenvectors are altered due to the transformation matrix. They therefore need to be transformed back:

$$w_j = w'_j S^T \tag{6.16}$$

Where w_j are the eigenvectors of the original matrix A and w'_j are the eigenvectors of the transformed matrix A'. This algorithm then completely substitutes the numericaly unstable one provided by the MDP. The changes were written directly in the MDP's file as the call is nested very deep in the architecture and would require extensive copying of the original code. The file $MDP\_HOME/nodes/sfa\_nodes.py$ needs to be replaced by the file provided on the disc supplied with the thesis. The only modification is the way eigenvalues are calculated and the rest of the file is untouched.

6.4.3 Accuracy comparison

Due to very different approaches to the same problem there are completely different numerical operations involved. This however, has minimal impact on the results as the eigenvectors extracted are exactly the same and eigenvectors are calculated with maximum error of 10^{-10} which has very little effect on the future calculations as they are only used to calculate the output by multiplying them with the data afterwards. It is important to note that some of the eigenvectors have opposite signs. This is not a mistake as eigenvectors are not unique and if each of the components of an eigenvector has its sign inverted the calculation is not affected - which is the case of this system as well.[5] Also the results of each of the SVD runs needs to be rearranged as the SVD provides reults in descending order but ascending one is needed since only the smallest eigenvalues are used in sink with their eigenvectors.

Chapter 7

Testing the model

The following tests are to observe and conclude how effective is slow feature analysis if applied on real data. The problem itself branches into two distinct categories. First is how well does the algorithm cope with the speaker it has been trained on. However, none of the tests will be run on data very similar to the ones the system has already seen in order to see if the system is resistant to translation and rotation of the image. Tests will be run also on different numbers of letters to see how well is the system able to differentiate between very similar letters if put into such position.

Second and more challenging series of tests will observe how well is a nonspecialised and quite generic slow feature analysis model able to abstract the letters across unseen speakers. I will observe if the performance increases with more trained speakers or if the system fails to identify the same letter presented by different people due to their signing styles.

To try and analyse what data were actually extracted and if they represent the gestures well an analysis of the features corresponding to different images will be done with primary goal to identify the stimuli that the features responded to.

For decission making linear support vector machine classifier was used as I focused on the overall performance of the method and if the method was actually extracting useful information.

The architecture for all of the below tests is the one described in 5.3 unless stated otherwise.

7.1 Trained speaker

The following series of tests was trained and tested on the same speaker. The train and test sets were kept the same and usualy at 250 images each.

7.1.1 Trainset size dependency test

This test is to illustrate and evaluate how train-size dependent the slow feature analysis is. Multiple tests were run involving the same letters but different trainset sizes on the the same testset. The testset parameter was kept constant in order to eliminate the posibility of certain parts of the testset altering the results by being removed and being the extremes on either side of the performance spectrum. For testing purposes the second half of the video was used (frames 250-500). To keep the video fluid the decrease of the trainset was done by removing images from the start of the video. Output dimensions are 10 for each but the top layer that has 20. The testset size also varies for trainsets with imagecount

greater than 250 as there are only 500 images available. An argument can be made that this affects the performance but I believe that the testset is still large enough to correctly represent the trend of the performance even if it doesn't provide the exact performance it should given the proper size. The system was trained on first 12 letters of the alphabet.

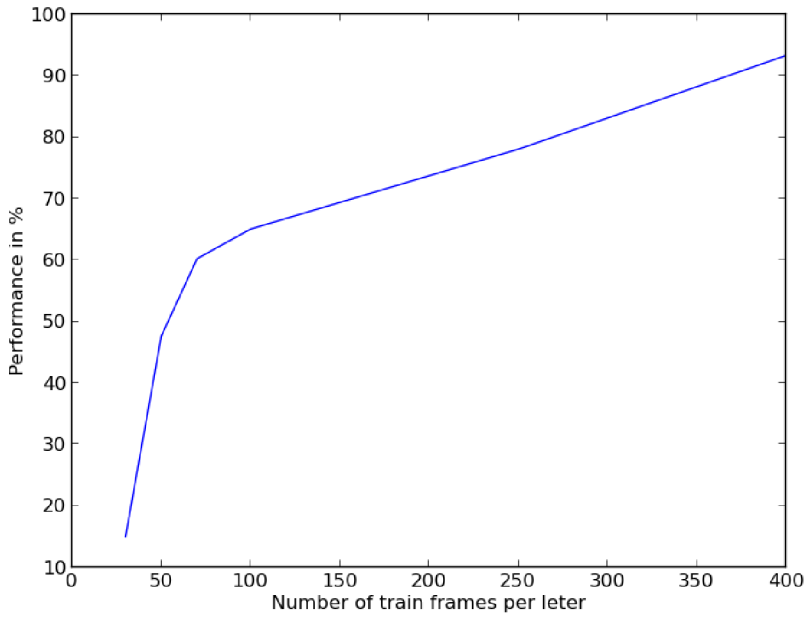


Figure 7.1: Graph of how performance varies based on number of input images per letter

The results are very pleasent for big enough train set and keep dropping at reasonable rate until about 100 images per letter trained. The linear decrease in performance indicates several things. Firstly - that the features extracted are indeed robust as they are resistant to translation and rotation in spite of the testing set being up to 2.5times greater than the training one. Secondly - the important features are extracted rather quickly. If we omit other factors such as change in the actual gesture, as some speakers tend to do so as shown in 4.4, it only takes about 100 frames to extract the most significant features of a sign. The performance peaked at 93.3% for 400 segments trained (and only 100 tested per letter). The important point to note here is that the hierarchical SFA shows good potential to recognise at least seen speakers.

7.1.2 Performance accross varying number of letters

It is quite obvious that the higher the number of supposedly learnt patters the lower the performance should be. The question is how fast will the performance drop sheding light on for what range of gestures is this system viable. When letters *s,t,u* were part of the trainset, being the last 3 tests, the train set were 178 images. For all other tests the sets were 250 images each. The decreased number in the testing set was caused by lower number of frames for the 3 mentioned letters. I decided to keep the training data the same in order to keep the extracted features the same (for letter that have previously been tested that is).

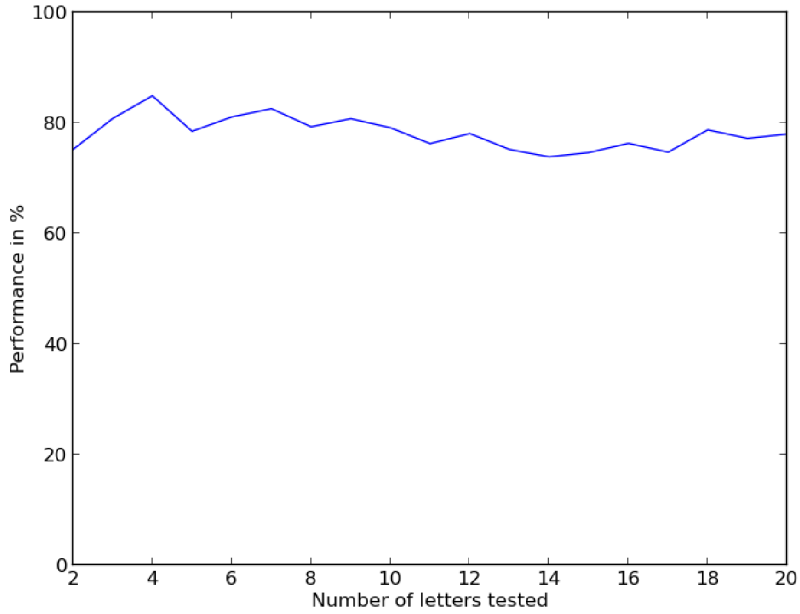


Figure 7.2: Graph of how performance varies with varying numbers of letters classified

The trend of the performance in this test was surprising. Even though the number of output dimensions was set to only 20 the performance stayed rougly constant and dropped only when similar looking letters were introduced as the performance for them is obviously the worst. The graph indicates that the SFA performed about the same no matter what the number of letters meaning that the features extracted are descriptive enough to cover all of the letters that were tested and that the SFA is a sufficient method to recognize all of the letters of the ASL, excluding the ones that were not tested - j and z. Considering the actual performance percentages one would say that they are not all that high. Keep in mind that the system was trained only on 250 images per letter. If we refer to graph 7.1.1 we can estimate that performance over 90% should be reached if trained on 400 images. In order to test if the performance develops the same it does in the figure 7.1.1 I ran a test on 20 letters and 350 train frames. I found the performance to be 85.24% which fits the referred figure perfectly.

7.1.3 Extracted feature analysis

The features correspond to stimuli from the image. Based on patterns of how much the feature reacts to which image conclusions can be drawn concerning what part of the image is the feature actually reacting to. The greater the deviation from zero for presented image the more is the feature present or recognisable.

Figure 7.1.3 shows an exemplary feature. The model was trained on only 2 letters when this feature was exracted. The letters were A and B and each had 350 frames. The receptive field of the feature is noted in the figure as well. From looking at the feature graph we can see that the feature reactions almost exclusively to letter A. In figure **??** images from this session are shown. From the images and especially the content of the receptive field of the feature I deduced that this particular feature reacts to knuckles of a

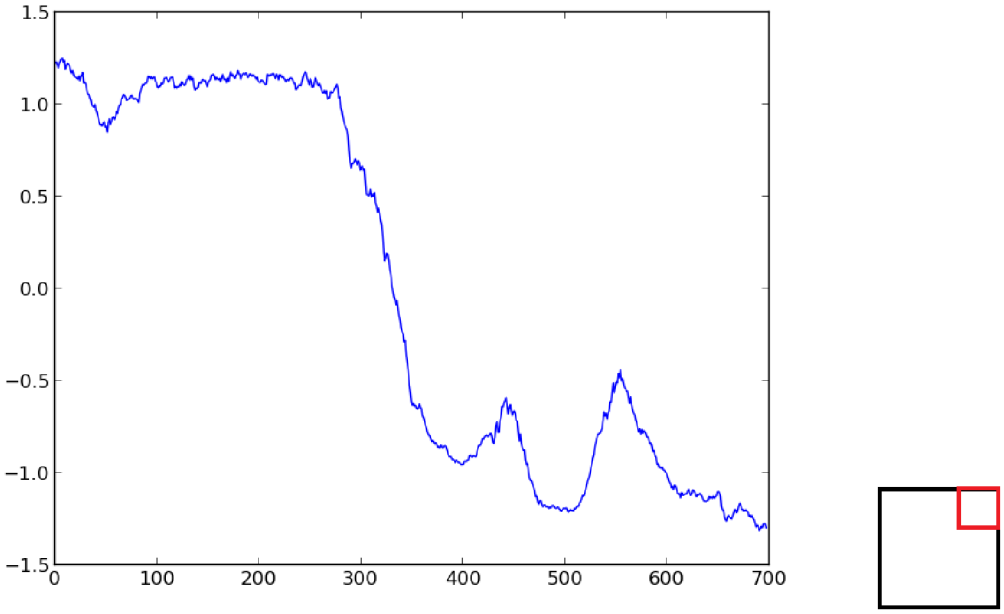


Figure 7.3: Feature response graph

Figure 7.4: Images: A123, A320, A372, A396, B51, B100, B137, B200, B281 and B369

closed hand. Since letter A is such a compact letter the knuckles are almost always visible in the segment except for after picture 300 when they start to move away. As for image B there are no knuckles in corresponding region of the picture. I also included the images in letter B that cause some sort of weak response. In those cases the tilted tips of fingers look somewhat like the knuckes in letter A and therefore result in some response. Their similarity is not too great explaining why the magnitude of the response is low.

7.2 Unseen speaker

This group of tests was based on the question if the method is capable of extracting robust enough features or if it gets beaten by the great variation in speakers' signing styles and habits.

Intresting factor concerning the learning process in this case are the different styles of

organisation of training data. Multiple tests will be analyzed in order to determine the best way to present the data to the system in order to maximize the performance if there is a differnce in performance at all. For each of those tests there are data from two speakers present and first four to eight letters of the alphabet are used to get a better understanding of how does the performance evolve. Testing and training sets of 150 images per letter are used and system architecture as well as output dimensions are set to standard values.

The baseline test that gives us an idea if the method is viable for different speakers with limited number of training subjects, since my dataset has only five signers, is a test that compares the performance based on the number of trained speakers.

7.2.1 Performance changes based on number of training signers

Probably the most important test of the method is to see if increasing the signer pool helps to improve the performance. This test also shows how robust the features extracted are on small sets of data. Only exemplary set of first six letters of the alphabet was used due to memory and time restrictions. The train and test sets are the standard 250 images for both. The input was organised based on speakers - full sequence of images of speaker A was shown before any of the images for speaker B were shown and so on.

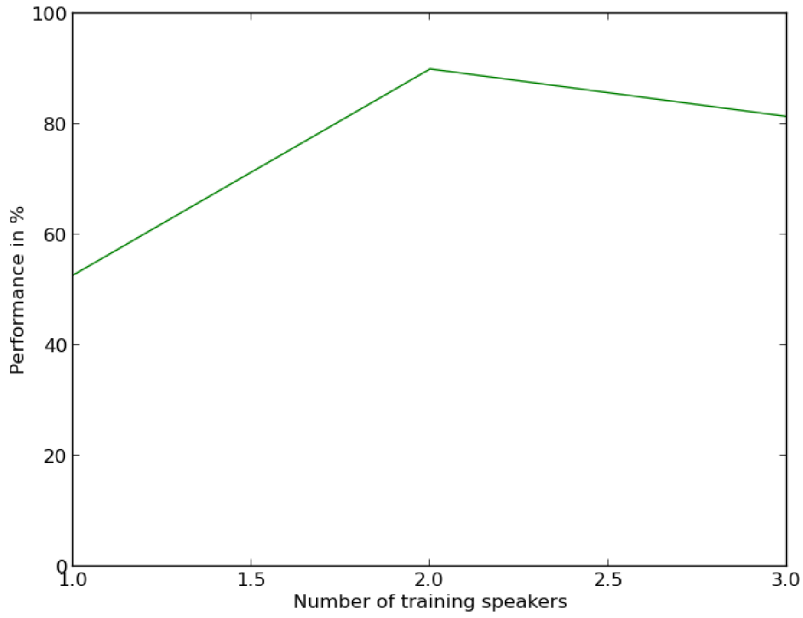


Figure 7.5: Unseen speaker number effect on performance

The performance increases considerably as the number of the speakers increases. Even though the system was trained only on 3 letters for this case based on the trend of the performance the improvement depends on if a similar speaker has been seen before. If not then the improvement is better. The best way would be to train the network on as many different speakers as possible as it increases the chances that the tested speaker will be somewhat similar to one of the training ones. The test prooves that the method has the capacity to improve with more speakers.

7.2.2 Speaker detection

The system might also extract features describing the signing style of the speaker. This test is trying to distinguish two speakers from each other. The test and train sets are set to 250 images per letter and an exemplary set of six letters was used - g,h,i,k,l,m Network architecture and output dimensions are default.

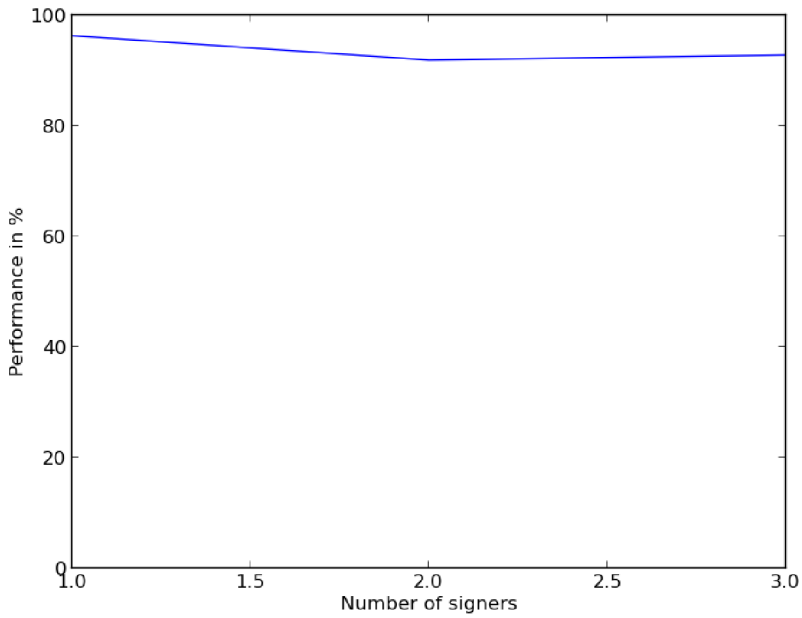


Figure 7.6: Graph of speaker classification

As figure 7.2.2 indicates that the performance is much higher than for letters. This is to be expected the signer being the slowest changing variable in all of the multiple speaker test runs with speaker ordered data. The speaker identification may be done from specific features of his or her hand such as ratio of the length of the fingers to the size of the palm. Other option is that there are behavioral patterns of signers extracted as features.

7.2.3 Speaker ordered data

One of the possible input data ordering for training is the natural way the video would be presented to the system. One speaker signing all of the letters, leaving the camera and another speaker presenting his data. This method of training is the most intuitive and was used in all other tests. As such this test is used as a baseline in order to estimate how much worse the performance gets if the speaker has not been seen before.

As seen in figure 7.2.3 for unseen speakers the performance drops rapidly with each added letter in spite of being trained on 350 images per letter (corresponding to 85% performance in single speaker scenario). There are several reasons behind this. Firstly the same letter signed by a different speaker sometimes look completely unrecognizable, as mentioned in 4.4. Another problem is that when only one speaker is trained the variation in each of the letters doesn't matter too much because the letter is at least somewhat specific. Once we add another speaker that has different signing style the error from malformed

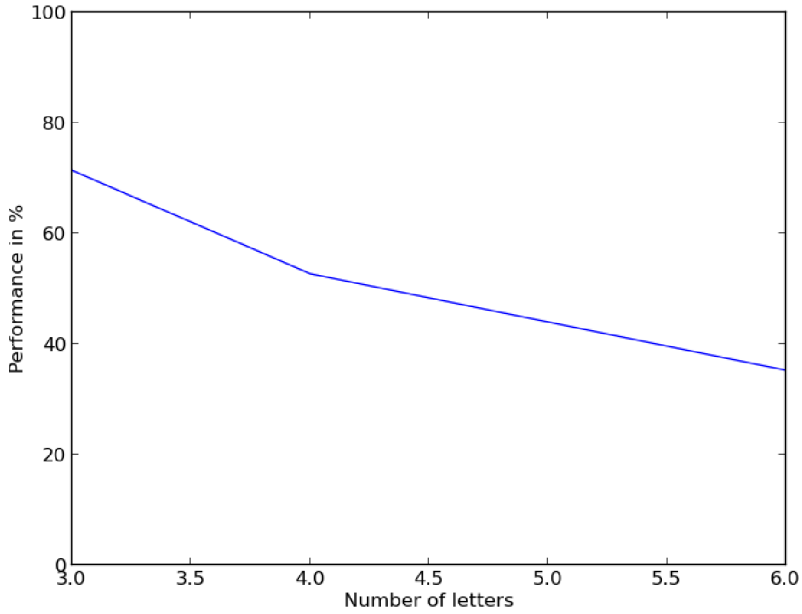


Figure 7.7: Unseen speaker development with more letters

letters starts to add up and the performance drops fast.

Chapter 8

Conclusion

The goal of the thesis was to observe and evaluate how well does the slow feature analysis cope with sign language data with high dimensionality and complexity. The goal was achieved by creating functional systems for various tests that inspected the performance in different scenarios that will most likely be requested in real life implementations. The method showed great promise in case of seen speakers where in spite of great differences in provided data caused by translation, rotation and sign deformation results over 90% were achieved. Adding the patterns, at least to maximum extent of the problem solved in the thesis - the ASL alphabet, the method managed to perform at generaly the same level for all of the letters in spite of some of the signs being very similar. The method has shown that it is fully capable to extract features that are robust enough to describe 20 signs in just 20 features. Some of the features extracted were also analyzed and a general idea what was the pattern that exemplary features were describing was obtained. Concerning the unseen speaker reasults I am not too confident about this method. Because of different signing styles and malformed letters that even a human would clasify incorrectly the method's performance drops quickly. The solution might be adding more speakers to the training as features extracted from more training data might be more descriptive. I see this having significant impact on performance once the system has been trained with every style of signing so that the unseen speaker can be generalized more easily.

In the first part of this text I got introduced to the slow feature analysis, the aspects of the data and the possible the possible system to apply the method on high dimensional data. The way of obtaining more robust features that can describe complex patterns was achieved by extracting locac features on smaller portion of the image and then using them to create new features by binding more parts of the image into one receptive field.

During the second part a testing application was created as a biproduct in order to facilitate testing on alternate letters and frames allow quick dimension changes and segment sizes. A part of the MDP framework had to be reworked as well due to numericly unstable algorithm to calculate the eigenvalues. This algorithm was replaced and everything else was untouched.

As for the future development I would definitely advise to use the slow feature analysis in case of high dimensional and complex visual data if high enough trainig set is provided. The method is able to abstract virtually any patterns that describe objects or actions varying slowly in time. I see the possibility to use the method to detect and learn emotions, more difficult gestures that also involve movement along a trajectory. As the performance of the computers increases so will the possibilities of this method as the data processing and training is resource intensive. On the other hand I see a big problem with the method.

As the process of learning is unsupervised if there is more than one object present in the image as the method has not been tested for this yet. The solution might be preprocessing algorithms that cut out only the parts of input data that contain the information to be learnt. This prohibits the method to learn from full visual input to find more abstract patterns.

Bibliography

[1] Python Software Foundation. System-specific parameters and functions, August 2014.

[2] Mathias Franzius, Niko Wilbert, and Laurenz Wiskott. Invariant object recognition and pose estimation with slow feature analysis. *Neural Computation*, 23(9):2289–2323, 2011.

[3] Laurenz Wiskott Mathias Franzius, Niko Wilbert. *Artificial Neural Networks - ICANN 2008*. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-87535-2.

[4] Thomas Melzer. Svd and its application to generalized eigenvalue problems, June 2004.

[5] Jarkko Isotalo Simo Puntanen, George P. H. Styan. *Matrix Tricks for Linear Statistical Models*. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-10472-5.

[6] L. Wiskott, P. Berkes, M. Franzius, H. Sprekeler, and N. Wilbert. Slow feature analysis. 6(4):5282, 2011. revision 137965.

[7] Laurenz Wiskott and Terrence Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, 2002.

[8] T. Zito, N. Wilbert, L. Wiskott, and P. Berkes. Modular toolkit for data processing, April 2012.