

University of West Bohemia  
Západočeská univerzita v Plzni

Master Thesis  
Diplomová práce

---

Large Scale Video Sequence Matching  
Vyhledávání duplicitních videí ve velkých databázích

Author:

Bc. Adam Fara

Supervisor:

Ing. Pavel Campr, Ph.D.

# Declaration

I confirm that I wrote this master thesis independently and have not used any sources that are not declared in the text. This thesis was not previously published or presented to another examination board.

In Pilsen, May 17th, 2016

Signature: \_\_\_\_\_

# Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni, 17. května 2016

Podpis: \_\_\_\_\_

# Acknowledgement

I would like to thank my supervisor Ing. Pavel Campr, Ph.D. for his priceless support and useful advice on the way to finish this work.

Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures".

# Poděkování

Děkuji vedoucímu svojí diplomové práce Ing. Pavlu Camprovi, Ph.D. za jeho pomoc a cenné rady při tvorbě této práce.

Děkuji CESNET LM2015042 a CERIT Scientific Cloud LM2015085 za přístup k výpočetnímu prostředí, ve kterém proběhla většina výpočtů. Výpočetní prostředí je poskytnuté v rámci programu "Projects of Large Research, Development, and Innovations Infrastructures".

# Zásady pro vypracování

1. Seznamte se s problematikou indexace a vyhledávání videí.
2. Prostudujte doporučenou literaturu.
3. Navrhněte a implementujte algoritmus, který umožní vyhledávat duplicitní videa nebo jejich části ve velkých databázích.
4. Úspěšnost algoritmu vyhodnoťte na zvolené testovací sadě videí.
5. Příslušné programové kódy vytvořte ve vhodných programovacích jazycích s využitím dostupných knihoven a nástrojů.

# Abstract

This work is focused on video copy detection in large databases. I use two different approaches of fingerprinting media, one of them based on SURF, the other one on siamese neural networks. Fingerprinting is a process of describing an image by n-dimensional numerical feature vector. Simplest feature vector of an image can be for example histogram of brightness values. In this work I use terms fingerprint, feature vector and descriptor as synonyms. Fingerprints are small files representing videos. They can be searched fast and storing them takes just a fraction of computer memory compared to storing video files.

The goal of this work is to determine the ability of proposed approaches of fingerprinting media and their transformed copies. Transformed copy of any media is changed in some way from its original source. It is also called duplicate media. It can be affected by compression, geometric transformations, etc. Another part of this work focuses on matching algorithm used to match fingerprints of two samples of visually similar media.

I also compare ability of both proposed descriptors to generalize feature vectors of original and transformed media. General feature vector should be similar for original media and its transformed copies. In other words, feature vectors retrieved from original and transformed media should be close to each other in feature space and be separated from feature vectors computed from another media.

Entire program created in this work consists of part computing image fingerprints and part matching similar fingerprints. Usage of this program might be in automatic media searching in large databases or online sources (YouTube, etc.), e.g. for finding unauthorized video copies. For the purpose of this work I created dataset of thousands of videos and their transformed copies.

I managed to create a system that is quite reliably able to find duplicate video if another copy affected with similar transformation is already in my precomputed dataset.

**Keywords:** video copy detection, video matching, siamese neural network for image description

# Abstrakt

Práce je zaměřena na vyhledávání duplicitních videí ve velkých databázích. V oblasti zpracování obrazu jsou často zažité anglické termíny, které se používají i v češtině. Překládání takových termínů způsobuje zmatek. Budu proto používat zažité anglické termíny. Používám dva různé přístupy k popisu (fingerprinting) médií. První z nich je založený na SURF deskriptoru, druhý využívá siamské neuronové sítě. Fingerprinting je proces popisu obrazu pomocí n-dimenzionálního číselného příznakového vektoru (feature vector). Nejjednodušší feature vector obrazu může být například histogram jasových hodnot. Termíny fingerprint, feature vector a descriptor mají v této práci stejný význam. Fingerprinty jsou velikostí malé soubory reprezentující video. Mohou být rychle prohledávány a jejich uchovávání zabírá jenom zlomek paměti počítače ve srovnání s uchováváním původních video souborů.

Cílem práce je určit schopnost představených přístupů popsat původní média a jejich upravené kopie. Upravená verze videa je nějakým způsobem změněná proti originálu. Upravenému videu se také říká duplicitní video. Může být změněné například kompresí, použitím geometrických transformací a podobně. Další část práce se zaměřuje na algoritmus sloužící k párování fingerprintů dvou vzorků vizuálně podobných videí.

Porovnávám také schopnost představených deskriptorů popsat obecně originální a upravenou verzi videí. Obecný příznakový vektor by měl být podobný pro originální a duplicitní video. Jinými slovy, příznakové vektory získané z původního a transformovaného videa by měly mít malou vzdálenost v prostoru příznakových vektorů a zároveň by měly být vzdálené od příznakových vektorů spočtených pro jiná videa.

Celý program se skládá z části, která počítá příznakové vektory videí, a části, která páruje příznakové vektory podobných videí. Použití programu může být k automatickému vyhledávání duplicitních videí v rozsáhlých databázích nebo online zdrojích (YouTube a podobně) například kvůli ochraně autorských práv. Pro účely práce byl vytvořen dataset čítající tisíce videí a jejich transformovaných kopií.

Podarilo se mi vytvořit systém, který je schopný celkem spolehlivě nalézt duplicitní video, pokud už v databázi předpočítaných videí existuje podobným způsobem poškozená verze stejného videa.



**Klíčová slova:** vyhledávání duplicitních videí, matchování videí, siamské neuronové sítě pro popis obrazu

# Table of Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Summary</b>   | <b>1</b>  |
| <b>2</b> | <b>Introduction</b>  | <b>2</b>  |
| 2.1      | My approach . . . . .  | 2         |
| <b>3</b> | <b>State of the art</b>                                      | <b>3</b>  |
| 3.1      | Spatial methods . . . . .                                    | 4         |
| 3.1.1    | SURF algorithm . . . . .                                     | 4         |
| 3.1.2    | Yang hashing algorithm . . . . .                             | 5         |
| 3.2      | Spatio-temporal methods . . . . .                            | 5         |
| 3.3      | Neural networks . . . . .                                    | 5         |
| 3.3.1    | Convolutional neural networks . . . . .                      | 6         |
| 3.3.2    | Applications of convolutional neural networks . . . . .      | 7         |
| 3.3.3    | Siamese neural networks . . . . .                            | 8         |
| <b>4</b> | <b>My contribution</b>                                       | <b>11</b> |
| 4.1      | Hashing algorithm based on method proposed by Yang . . . . . | 11        |
| 4.2      | Siamese NN . . . . .   | 16        |
| 4.2.1    | Implemented models . . . . .                                 | 16        |
| 4.2.2    | Model modification . . . . .                                 | 17        |
| 4.2.3    | Model verification . . . . .                                 | 18        |
| 4.2.4    | Training dataset . . . . .                                   | 19        |
| 4.2.5    | Experimental results . . . . .                               | 20        |
| 4.2.6    | Time consumption . . . . .                                   | 24        |
| 4.3      | Video matching . . . . .                                     | 25        |
| 4.3.1    | Video matching algorithm . . . . .                           | 25        |
| 4.3.2    | Sequence verification . . . . .                              | 25        |
| 4.3.3    | Results for Yang algorithm . . . . .                         | 26        |
| 4.4      | Preprocessing . . . . .                                      | 28        |
| 4.5      | Dataset . . . . .  | 28        |
| 4.5.1    | Flowchart . . . . .  | 30        |

|          |                       |           |
|----------|-----------------------|-----------|
| <b>5</b> | <b>Conclusion</b>     | <b>33</b> |
| 5.1      | Summary . . . . .     | 33        |
| 5.2      | Future work . . . . . | 33        |
| <b>6</b> | <b>Appendix</b>       | <b>35</b> |

# 1 Summary

In this work I introduce ways of media copy detection in large video datasets, specifically video copy detection. Large video dataset is a database of precomputed fingerprints. Storing only fingerprints instead of their source video files brings two advantages. Files with fingerprints are small compared to video size and this easier to store. In the case of this work fingerprint files were about thousand times smaller than corresponding video files. Fingerprints can be also effectively searched in relatively short time.

Media copy can be found as an entire or partial copy of original media, or along with other copies as a subpart of newly created media. Newly created media can for example be a movie review. It contains short parts of original media reedited and saved as new video. In general, there are two approaches in matching videos, content based matching and watermarking[1]. Unlike content based matching, watermarking can be only used for those media which were marked before their original distribution. Marking video can be done by inserting a logo or another pattern.

In this work I focus on spatial media matching that belong to content based matching. Spatial feature vectors are extracted from each single image. They describe in some way what is happening in the image and extraction of such feature vector has no connection with other frames in the video. Spatial approach is only based on image similarity, and is a basic approach that one could use for steady images matching. Video, being a process that evolves in time, allows us to augment its description by connection between individual frames. I compare two different ways of computing features and matching them. First, I present simple hashing algorithm based on SURF features [2]. Second approach is based on siamese neural networks. I assume the first approach to be robust on identical video sequences, however, sensitive to any kind of image transformation. Neural network should overcome this disadvantage. If the network is trained correctly it should return more similar feature vectors for original and duplicate media.

## 2 Introduction

Video matching problem is interesting field with no clear path of correct approach. There are different attempts [1, 3, 4, 5] of video matching, however, none of them has been proved to be the right one. There are many companies and research teams trying to solve this problem more or less effectively. Modern approaches of video copy detection are content based, so they can be applied on any existing video with or without previous marking. There are several ways of fingerprinting videos and I will introduce two of them including video matching algorithm. Content based methods do not require any additional information but the media itself. All information that is needed for video matching is retrieved from the video itself.

### 2.1 My approach

The main goal of this work is to summarize some of existing solutions for video matching and try and build own implementation upon them. Core structure of my algorithm is based on brute force matching algorithm which will definitely need deep revision in future work. As an input for the matching algorithm I introduce two approaches. First of them is based on spatial method [2] using SURF features as an image descriptor, second one is based on siamese neural network. Siamese neural network is non-trivial approach with many variables, therefore, I will try to find satisfactory model for my task.

### 3 State of the art

Unauthorized video usage and sharing has become a massive problem for authors and distributors. There is already a neologism "freebooting" [6] associated with using others people media. Freebooting means downloading a video and reuploading it again (in same version or edited) to another server or channel without the creator's permission.

Unlike images, video is much more complex media making it more difficult for video search engines to find the copies. Video copy is not an exact duplicate of originally distributed media. It can be edited in several ways; for instance geometric transformation (resizing, rotating, etc.), non-geometric transformations (blur, contrast, etc.), cutting and using short parts of original video, reversing video, changing audio track, picture-in-picture and many others. Making a robust search system that would be capable of covering all possible transformations and their combination is probably impossible task at the current time. Many approaches have been introduced, each of them more or less precise in different cases.

There are two ways of protecting video content. First and older of them is watermarking, the other one is content based copy detection [3, 4]. Watermarking relies on inserting a pattern into frames. Only those media whose original version was watermarked can be found using watermark based methods. On the contrary, content based methods are using fingerprints independent on any primary pattern or modification and do not require any additional information but the media itself. Fingerprint means the same as feature vector or descriptor. It is a numerical code retrieved from an image which describes some characteristic of the image (e.g. color histogram).

Content based copy detection can be divided into two main groups; spatial and spatio-temporal methods. Spatial methods rely on local descriptors of a video frame, while spatio-temporal methods combine local image descriptors with temporal trails of the partitions [1]. All mentioned methods are often being reinforced by other descriptors, e.g. sequence of certain video can be described by combination of visual and audio features.

## 3.1 Spatial methods

Spatial methods are using image descriptors. Each frame can be fingerprinted by one or by combination of several descriptors. The fingerprints thus obtained should describe a frame in a general way, however, still preserving enough information. Therefore, two fingerprints that are close to each other (e.g. in Euclidean space) should describe two visually similar images. Simplest copy detection method can decide on the basis of number of visually similar frames. Spatial methods are mostly based on feature vectors computed from basic descriptors such as SURF or local color histograms.

Algorithm implemented in this work is based on paper by Yang et al. [2]. This algorithm uses SURF-based features. To avoid confusion of SURF algorithm and hashing algorithm based on SURF [7], I will refer to Yang’s hashing algorithm [2] as Yang algorithm. Proposed method is fast and reliable. Its weakness is hidden in the necessity of keeping enormous amount of precomputed features. Matching algorithm compares query video’s feature vector with each feature vector in the dataset. Query video can be transformed in different ways. Feature vector of each video and for all its considered transformations has to be stored in the dataset. Clearly, cardinality of dataset is a huge disadvantage of this approach.

### 3.1.1 SURF algorithm

SURF algorithm, that is used in Yang algorithm, was first introduced for the task of finding corresponding points between two cameras in stereo vision. Matching images from two different sources is non-trivial task. Effective image descriptor must be robust to changing light conditions, geometric transformations, noise, etc. Descriptor must also ensure that corresponding points of interest will be the same in both images. In SURF descriptor this was achieved by splitting descriptive function into two parts.

Detector part is responsible for locating distinctive locations, such as blob, T-junctions, and corners. On the other hand, descriptor part represents neighborhood of point of interest found by detector. Combination of mutual positions of points of interest from both images together with their similarity computed from corresponding descriptors gives strong performance of image matching. Term SURF descriptor is sometimes used for entire algorithm (or returned feature vector) and does not only refer to point’s discriminative part.

SURF detector is based on Fast-Hessian detector. Given a point  $\mathbf{x} = (x, y)$  in an image  $I$ , the Hessian matrix  $H(\mathbf{x}, \sigma)$  in  $\mathbf{x}$  at scale  $\sigma$  is defined as follows

$$H(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix} \quad (3.1)$$

where  $L_{xx}(\mathbf{x}, \sigma)$  is the convolution of the Gaussian second order derivative  $\frac{\partial^2}{\partial x^2} g(\sigma)$  with the image  $I$  in point  $\mathbf{x}$ , and similarly for  $L_{xy}(\mathbf{x}, \sigma)$  and  $L_{yy}(\mathbf{x}, \sigma)$ .

### 3.1.2 Yang hashing algorithm

Another term usually associated with image processing is hash code. For the purpose of this work it will mean just the same as feature vector or fingerprint. The hash code algorithm implementation works as follows; each frame of a video is resized to 400 by 400 pixels and subdivided into 100 by 100 pixels cells, giving 16 non-overlapping subparts in the frame. SURF [7] descriptor is computed over each cell. Number of key points found by SURF is used as descriptor for given cell. Neighborhood description given by SURF is not used at all. Hash code for each frame consists of 16 integers, one for each subpart of the frame. Hash code can be also used in a meaning of descriptor of an entire video. Then it is simply an array of feature vectors of each frame.

## 3.2 Spatio-temporal methods

Changick and Bhaskaran introduced innovative approach [1] of video sequence matching which adds "action" information to the descriptors. They claim that traditional content based approach using only spatial information about image often returns false matches or fails at matching duplicate media. Their algorithm computes temporal fingerprint of media. Basic idea lies in dividing each frame into four non-overlapping parts and computing pixel intensity histogram over each part. Splitting image into four parts gives an algorithm robustness against changing aspect ratio from 4:3 to 16:9 and vice versa, and similar transformations (e.g. letterbox). Beside that they compute difference between adjacent frames which is used as feature vector instead of fingerprinting each single frame.

This idea is similar to differential Freeman chain code as it also makes the hash code invariant to initial conditions. For example media affected by brightness adjustment should have almost the same hash code as its original source. Hash code based on brightness histogram is also insensitive to many transformations I had to deal with, see chapter 4.1.

## 3.3 Neural networks

Neural networks have become widely used tool in many applications of machine learning and data mining. Origin of first artificial neural networks dates back to 1943 when model of McCulloch-Pitts (MCP) neuron was introduced. Artificial neurons were initially designed to mimic function of neurons in human brain. Single neuron can only be used for simplest classification task. Connecting neurons into multilayer nets gives them power of high dimensional non-linear classifiers.

In process of learning, network adapts weights connecting individual neurons to give the best possible response for given input, for instance to simply classify positive and



negative numbers into two classes. Process of updating weights based on classification error is called backpropagation [8].

### 3.3.1 Convolutional neural networks

Convolutional neural networks [9] have been successfully applied to many tasks. Difference between convolutional neural networks and fully connected neural networks is in the way each layer processes data. While fully connected layers receives and processes data as an one dimensional array, convolutional layer deals with two dimensional structure. Image data can be also represented as an one dimensional structure, however, it is much more convenient to keep them in their natural two dimensional structure. Another advantage of convolutional layers is that they are usually much smaller than fully connected layers. Size of each layer is affects training time of the network and using convolutional layers makes training much faster and less memory consuming.

Convolutional neural networks typically use layers that work as local filters; edge detectors, blurring mask, line detector, etc. Single convolutional layer usually consists of multiple convolutional kernels, each of them can be trained to work as different type of detector. Resulting effect is that entire neural network can gain more significant information from input data than one would ever be able to suggest and compute by classical engineering approaches. Process of abstraction high-level features using multi-layer neural network is also called deep learning [10].

Main advantage of using convolutional neural network is that it is not clear what kind of preprocessing would be most convenient for given data and doing such a thing would be guessing. Process of backpropagation used while training neural network naturally forces the layers to form convolutional kernels into most effective image filters and detectors. Figure 3.1 shows typical structure of convolutional neural network. Convolutional layers are usually followed by pooling layers (sub-sampling layer) which reduces output dimension of previous layer. Pooling layer is traditionally a non-overlapping averaging or max value filter.

After several pairs of convolutional and pooling layers it is common to use some kind of non-linear layer or normalization layer. Popular non-linear layer used in convolutional neural networks is rectified linear units (ReLU) layer. Using ReLU allows faster training [11] compared to standard activation function 3.2 or 3.3. Particular structure always differs for individual application.

$$f(x) = \tanh(x) \tag{3.2}$$

$$f(x) = (1 + e^{-x})^{-1} \tag{3.3}$$

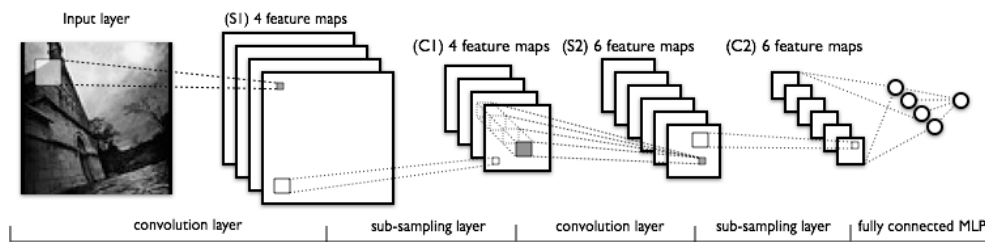


Figure 3.1: A typical structure of convolutional neural network [12].

### 3.3.2 Applications of convolutional neural networks

#### Face recognition

Convolutional neural networks have been successfully used for both face detection and recognition. Face detection is simply a task of deciding whether there is face in an image and locating face. Task of face recognition on the other hand is to match face template with face stored in the database. There are basically two usages of neural networks for face recognition depending on the application of the face recognition system [13].

First of them is finding face in large databases. Often there are only few images of each person in the database. This application usually returns list of possible candidates. Searching in this case does not need to run in real time and does not have huge requirements on speed performance. Typical situation is finding a person in police database.

Second typical application is real time face recognition. Such an application is usually used for security reasons, for instance verifying identity of person trying to access secured system. Typical usage can be computer login verification. In this case there is quite limited database of possible candidates and there are also several images of each person in the database. These systems require high reliability in face detection. Another important requirement is time consumption for real time usage.

#### Image classification

Another interesting field of using convolutional neural networks is image classification. With growing usage of computer vision there is also growing need of having reliable system capable of object recognition. Training such a network requires enormous number of annotated training images. There are already several databases that can be accessed. From interesting datasets I can mention ImageNet [14] which consists of roughly 15 million high-resolution images in over 22 000 categories, or LabelMe [15] database which consists of thousands fully segmented images and is still growing.

Image classification problem is similar to the application introduced in this work. Image

classification tries to train a network capable of deciding whether several images belong to the same class, in other words to decide if they are in some way similar enough and differ from other visually less similar images. In my case I do not have thousands of classes that I need to classify images into. Task of my network is only to decide if two images are visually similar or not. For this reason, similar structures of neural networks that are usually used for image classification can be used.

### 3.3.3 Siamese neural networks

Siamese neural networks were first introduced by Bromley et al. [16] in 1993 as a tool of verifying hand-written signatures. Learning similarity metric with siamese neural networks is done by presenting pairs of similar and non-similar images to the network. Main idea lies in ability of network to find a subspace in feature vectors that describes similar inputs with low distance and non-similar inputs with high distance.

Siamese network has two branches; each branch is one neural network. In my case these networks are forward convolutional networks. Both networks share their weights. Networks are fed at the same time by two images which can be either similar or not. Similarity of images is given by labeled dataset. Purpose [16] of shared weight is to find subspace in feature space that will lead to similar (ideally same) outputs (feature vectors) on both branches for similar images and different outputs for non-similar images. Loss function on the top of siamese network is used while training the network. It computes similarity of both outputs from their distance in feature space. Loss function is designed to map outputs of neural network (n-dimensional feature vectors) closed to each other for similar inputs (images) and distant for non similar images. Similarity of images is defined by training dataset. Similar pairs of images are transformed in one way and labeled with 1, non similar pairs of images are transformed in another way (or derived from different images) and labeled with 0. By defining transformations used in training dataset is also defined metric which neural network is forced to learn. Distance of non similar images is given by parameter *margin* in 3.4.

Siamese neural networks are ideal for the task of two images comparison. Figure 3.2 shows general structure of siamese neural network. Whole structure consists of two identical neural networks sharing all weights. Neural network is trained by sending pairs of images simultaneously to both inputs. Pairs are binary labeled; 1 for similar images and 0 for non-similar images. Composition of neural network can contain different layers. Usually there are several convolutional, pooling, and non-linear layers. Last layer is usually fully connected.

On the top of the layers is placed loss function layer. It compares output from both fully connected layers and its output is used for backpropagation.

There are several commonly used loss functions comparing outputs from both networks. In my case I used contrastive loss [17] function 3.4. Along with other loss functions,

contrastive loss function is used in the case when two inputs are fed into the network and can compute their distances.

$$E = \frac{1}{2N} \sum_{n=1}^N (y)d^2 + (1 - y)\max(\text{margin} - d, 0)^2 \quad (3.4)$$

where:

$d = \|a_n - b_n\|_2$  is distance if two feature vectors  $y$  is label of training pair; 0 for non similar and 1 for similar images

$\text{margin}$  is desired distance between two not similar images

$N$  is number of input pairs in one training iteration

$d$  is distance between current pair of feature vectors fed into the network

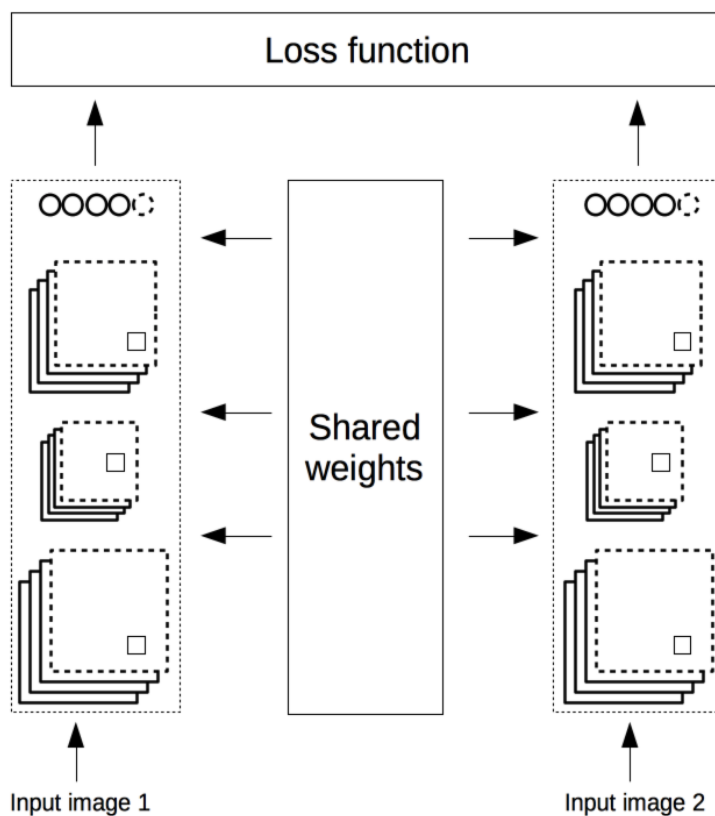


Figure 3.2: General structure of siamese neural network

Implementing neural networks in source code of a program is not a hard task. But training large networks is enormously time consuming and very slow. Luckily, there are

several open source deep learning frameworks available and optimized for fast computing. Noteworthy frameworks are Caffe [18], Cuda-convnet [19], DeCAF [20], TensorFlow [21] or Theano [22]. Some of them can compute on both CPU and GPU. In chapter 4.2.6 time requirements for CPU and GPU are consulted. In this work I used Caffe framework with CUDA acceleration.

## 4 My contribution

The aim of this work was to create matching system that would be able to find duplicate video (if it exists) in video database. To achieve the goal I split the task into four smaller independent parts.

- Dataset creation 4.5
- Video descriptor based on Yang algorithm 4.1
- Video descriptor based on Siamese neural networks 4.2
- Video matching algorithm 4.3

Structure of created source code allows one to use each part separately or exchange one part by another implementation. Matching algorithm can use hash codes retrieved from any other descriptor; another video descriptors can be computed from the dataset. Complete system can be used for finding duplicate videos in large databases. Importance of such system is explained in section 3. Idea behind both implemented video descriptors is that feature vector computed from original and transformed media should be the same (or very similar) for both of them. Video matching algorithm searches for duplicate media by finding feature vectors with small distance in feature space. Small distance in feature space means that feature vectors are similar.

### 4.1 Hashing algorithm based on method proposed by Yang

Following chapter summarizes implementation of Yang algorithm [2] introduced in section 3.1.1. Yang algorithm returns SURF-based features. I refer to this algorithm as Yang algorithm or simply Yang. Image 4.1 shows six different transformations and original frame. Tested transformations are Gaussian blur, overlapping borders, brightness adjustment, gamma correction, rotation and scale. Each frame is divided into 16 non-overlapping cells (blue lines) and SURF-based features (red circles) are computed over each cell. Hash coded retrieved by Yang is 16 dimensional vector consisting of number of Yang features found in each cell. Number of red circles in corresponding cells should ideally be the same for each transformation and original image. In other words, we would like to get same hash codes for all (a) - (g) images in figure 4.1.

Every transformation has impact on resulting feature vector. Applying transformation on a video moves pixels in frames or changes their values. Yang features retrieved from transformed image will be different from features retrieved from original video. To make the matching system robust, dataset must contain descriptors of several variations of each transformation and their combinations. Robustness of video matching system relies on its ability to find duplicate video in existing dataset (if duplicate video exists). Robust system should not return too many false matches. False match occurs when the system indicates a video as duplicate, although the corresponding video in database is different. Matches indicated by system are usually checked by a person and the purpose of video matching system is to report as few false matches as possible.

In my dataset I generated several variations of each used transformation. For example Gaussian blur transformation was computed with kernel sizes 3, 5, 7, 9, 13, and 17. Complete list of used transformations and their results for Yang algorithm are in section 6. Tables 6.1 - 6.6 document sensitivity of Yang features to all used transformations. Representative range of parameters for each transformation was chosen to show response of algorithm to increasing damage made to original video. These are the main results Yang implementation. Results show ability of Yang features to retain similarity of original and transformed media. Meaning of results is to show how many videos were correctly matched if they were affected by used transformations. Ideal result would be values close to 1 on main diagonal and values close to 0 on antidiagonal. The farther these values are from desired values the less is matching system capable of matching original video with transformed video.

Table 4.1 shows results for selected parameters of each implemented transformation. Table contains accuracy,  $F_1$  score and normalized confusion matrix.

Idea behind creating wide range of transformation for each video assumes that media that has to be matched will be affected in similar way to one or few modifications stored in prepared dataset. Matching algorithm can decide based on video with highest response.

Interesting data to watch in confusion matrix are in the first row. First element is *true positive* value, second element is *false negative* value. True positive says how many of frames classified as similar were actually similar. False negative number says how many of frames classified as non-similar should have been classified as similar. Changes between these two values are dependent on transformation performed on video. For each type of transformation it was measured how much the media can be transformed in order to still be correctly matched.

Second row of confusion matrix contains *false positive* and *true negative* values. False positive number says how many of non-similar frames were classified as similar. True negative says how many of non-similar frames were classified as actually non-similar.

Introducing false matches after transforming video is just a coincidence and therefore this value keeps very low. Most of false matches are later filtered by matching algorithm explained in section 4.3.2.

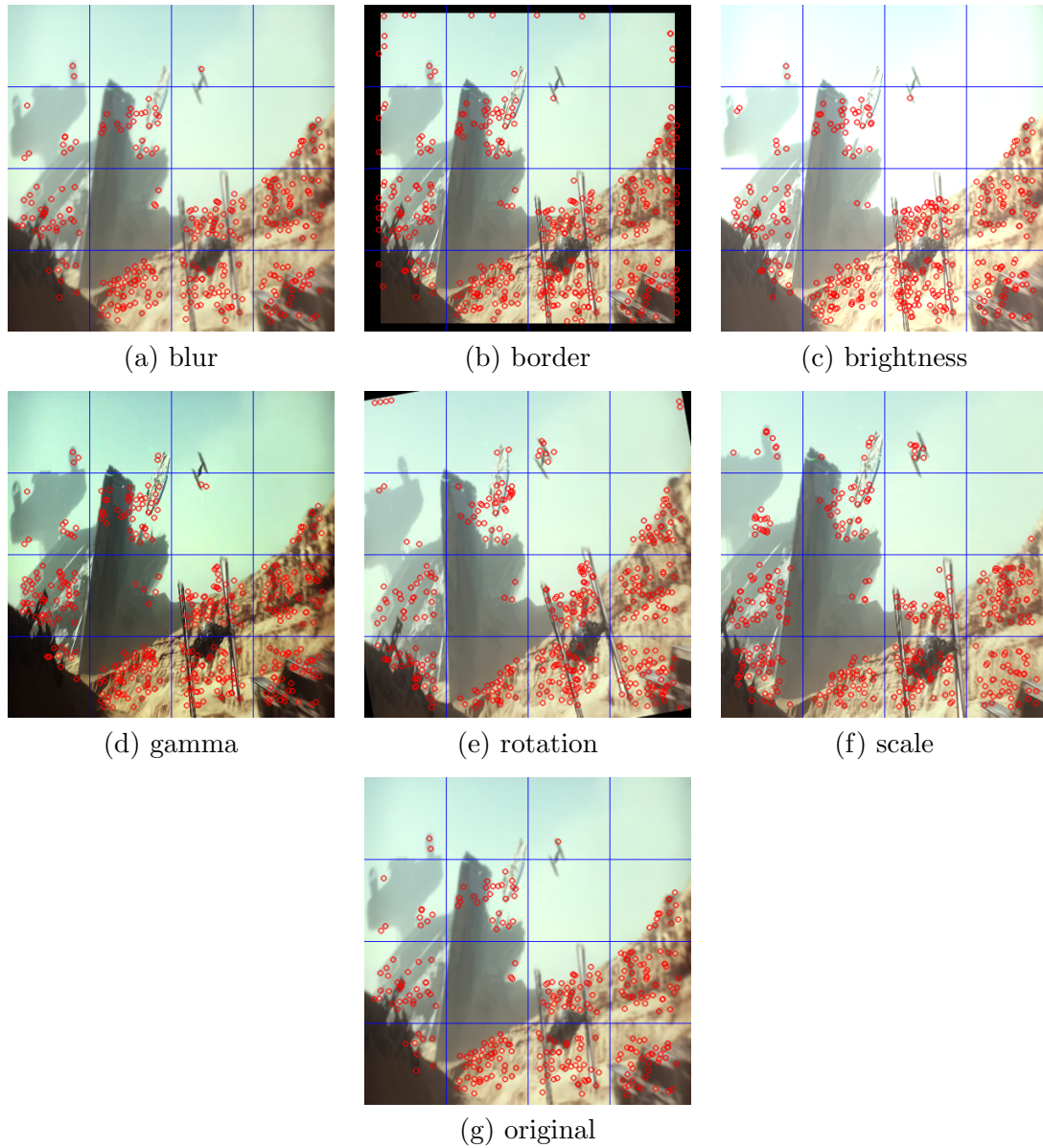


Figure 4.1: Transformations performed on videos stored in the dataset; (a) Gaussian blur, (b) overlapping black border, (c) brightness adjustment, (d) gamma correction, (e) rotation, (f) scale, (g) original image. Displayed image is result of Yang algorithm.



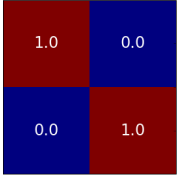
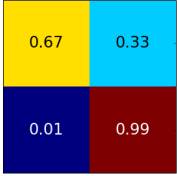
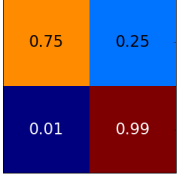
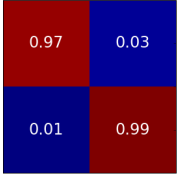
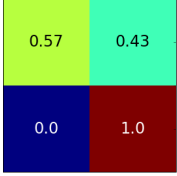
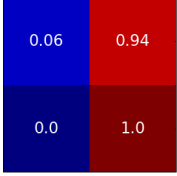
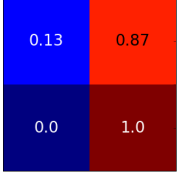
| Transformation                      | Accuracy | F <sub>1</sub> score | Confusion matrix  |
|-------------------------------------|----------|----------------------|---|
| No transformation (identical files) | 0.996    | 0.441                |    |
| Blur, kernel size 3 px              | 0.994    | 0.228                |    |
| Overlapping border, 2 percent       | 0.994    | 0.257                |    |
| Brightness + 30                     | 0.995    | 0.331                |   |
| Gamma correction 1.1                | 0.997    | 0.357                |  |
| Rotation 3°                         | 0.996    | 0.044                |  |
| Scale 1.05                          | 0.996    | 0.074                |  |

Table 4.1: Sensitivity of Yang algorithm to selected transformation. Complete results of YANG algorithm are in appendix 6.

## Accuracy

Accuracy 4.1 is statistical measure giving information about closeness of measured values to real values. In this case accuracy does not really have distinctive meaning. When comparing two videos, great number of non-similar frames is retrieved. These frames appear in *true negative* value. Since the number of *true negative* is usually about thousand times greater and other responses, accuracy is always pushed close to one. For this reason, it is important to watch other measurements as well.

$$accuracy = \frac{tp + tn}{tp + tn + fp + fn} \quad (4.1)$$

## Precision, recall

*Precision*, also known as *positive predictive value*, and *recall* 4.3, also known as *sensitivity*, gives better picture about classification. Precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned [23]. In other words, precision gives probability that predicted class will be correct and recall gives probability that correct class will be chosen by the classifier. Values of precision and recall should be similar. With better classification both values should grow.

$$precision = \frac{tp}{tp + fp} \quad (4.2)$$

$$recall = \frac{tp}{tp + fn} \quad (4.3)$$

## F<sub>1</sub> score

$F_1$  4.4 is computed from *precision* and *recall* and therefore is more descriptive than *accuracy*.  $F_1$  is good measurement to use for comparing two different classifiers on the same data.

$$F_1 = \frac{precision \cdot recall}{precision + recall} \quad (4.4)$$

## 4.2 Siamese NN

### 4.2.1 Implemented models

My goal with neural networks was to train a net that would be able to distinguish similar and non-similar images. Similarity of two videos is decided based on the similarity threshold which is later described in section 4.2.4. Two images are similar when they both are derived from the same image and they both are transformed (each of them in different way) within bounds set for similar images. If the transformation is within bounds for non-similar images or if the images are derived from two different images, they are considered non similar.

After learning this metric, neural network can be used to compute feature vector of an image or sequence of frames in video. Such feature vectors can be used for video matching.

Creating new structure of neural network for image matching is tricky task, since there are too many parameters that can be modified. One can use different types of layers, sizes of kernels, change the order of layers and so on. In a way, huge neural networks are kind of a black box. For instance, evolution of convolutional layers' kernels can be observed, however, it is not really clear what would be desired state of convolutional kernels. Some networks are for example trained to find hand-written digits. In this case one can expect to see some kind of edge detectors. On the other hand, siamese network trained to compare similarity of two input images does not necessarily have to detect edges. In fact, I observed that kernels of trained network still look rather noisy.

I implemented three different network structures from previous research. Implemented structures were previously for similar tasks of image classification and therefore I consider them being suitable for solving my task. Out of these, two networks were implemented including their pretrained weights. These neural networks consist of several convolutional layers which are usually followed by pooling layers and non-linear (ReLU) layers. Last two layers are fully connected; output from the very last layer can be stored as a feature vector for an image.

I implemented these models using neural network surgery. Such a process comprises part of original network including pretrained weights and new layers, usually initiated with random weights. Idea of this method is that pretrained part of the network helps to better and faster train the network, while newly added part with randomly initiated weights allows the network to adapt to new type of input images than the original neural network was trained for.

In my case I kept all layers but the very last fully connected one. I tried initializing last fully connected layer with different number of neurons to measure impact of size of feature network on training. Experimental results, however, did not show big difference.

Overall accuracy of network was not satisfactory.

### 4.2.2 Model modification

Coming with brand new structure of neural network without deep previous research is not a good way of finding successful network model. Therefore, I came with network structures derived from models introduced in related work. Table 4.3 shows different modifications of networks which I tried to train. Structure of neural network is based on model introduced in Caffe example [24]. Table 4.2 shows structure of neural network including parameter settings for network modification ID 1. ID 1 refers to models in 4.3. All other networks in table 4.3 have the same structure as network ID 1, but different parameters. Interesting thing about this structure are three fully connected layers which are gradually lowering dimension of feature vector describing the image. Experiments showed that increasing of dimension of feature output feature vector can slightly improve network performance.

Implemented Caffe framework allows to switch between several predefined solvers. One of the parameters that can be changed is learning rate policy which controls adjustment of learning rate during training the network. Original learning policy used in this model was "inv" 4.5 which controls learning rate as a function

$$lr = base\_lr \cdot (1 + gamma \cdot iter)^{-power} \quad (4.5)$$

where:

*base\_lr* is initial learning rate

*gamma* and *power* are parameters controlling speed of decrease of the function

*iter* is training iteration

Second used learning policy was "multistep" 4.6 defined as

$$lr = base\_lr \cdot gamma^{(floor(iter/step))} \quad (4.6)$$

where:

*base\_lr* is initial learning rate

*gamma* and *power* are parameters controlling speed of decrease of the function

*iter* is training iteration

Using "multistep" policy gives one full control over when and how the learning rate is changed.

More networks were trained based on models from [9] and [25]. Those networks were not successful for my task and are not included in the table.

| layer number | layer type    | kernel size | number of outputs | stride |
|--------------|---------------|-------------|-------------------|--------|
| 1            | convolutional | 3           | 20                | 1      |
| 2            | pooling       | 2           |                   | 2      |
| 3            | convolutional | 3           | 50                | 1      |
| 4            | pooling       | 2           |                   | 2      |
| 5            | inner product |             | 500               |        |
| 6            | ReLU          |             |                   |        |
| 7            | inner product |             | 10                |        |
| 8            | inner product |             | 2                 |        |

Table 4.2: Structure of most successful neural network.

| Id | Parent | Ker 1 size | Ker 1 output dim | Ker 2 size | Ker 2 output dim | Input image size | Fully con. layer size | Acc.        | Other changes          |
|----|--------|------------|------------------|------------|------------------|------------------|-----------------------|-------------|------------------------|
| 1  | -      | 3          | 20               | 3          | 50               | 20x20            | 2                     | <b>0.85</b> |                        |
| 2  | -      | 5          | 20               | 5          | 50               | 40x40            | 2                     | <b>0.8</b>  |                        |
| 3  | -      | 5          | 20               | 5          | 50               | 64x64            | 2                     | <b>0.7</b>  |                        |
| 4  | -      | 5          | 20               | 5          | 50               | 96x96            | 2                     | <b>0.75</b> |                        |
| 5  | 4      | -          | -                | -          | -                | -                | -                     | <b>0.7</b>  | multistep <sup>1</sup> |
| 6  | 5      | -          | -                | -          | -                | -                | 32                    | <b>0.8</b>  |                        |
| 15 | 6      | -          | -                | -          | -                | -                | 64                    | <b>0.75</b> | LR <sup>2</sup>        |
| 16 | 7      | -          | -                | -          | -                | -                | 128                   | <b>0.8</b>  | LR                     |
| 17 | 8      | -          | -                | -          | -                | -                | -                     | <b>0.9</b>  | LR                     |
| 18 | 4      | -          | -                | -          | -                | -                | 32                    | <b>0.9</b>  |                        |

Table 4.3: Overview of some modifications of trained network. Table is structured as derivation tree with parents and children. If a cell is dashed no change was done in that parameter. Many networks were not successfully trained, those are excluded from the table.

### 4.2.3 Model verification

Implemented neural network model was verified on Mnist dataset [26]. It has been shown before that neural networks can be successfully trained on this dataset. It is reasonable to verify my model on Mnist dataset before trying other inputs.

<sup>1</sup>multistep = Learning rate policy changed from "inv" to "multistep" with  $\gamma = 0.5$  and step changes in iterations 1000 and 1500.

<sup>2</sup>LR = Initial learning rate decreased from 0.01 to 0.001.

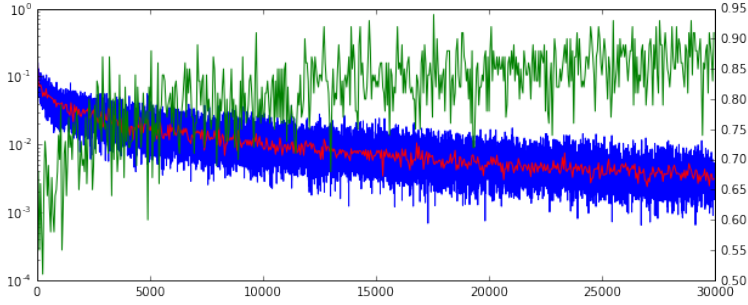


Figure 4.2: Training of model verification network. Red line shows test loss, blue line shows train loss, green line shows train accuracy. X axis represents training iterations. Y axis is in logarithmic scale for loss values and linear scale for accuracy.

#### 4.2.4 Training dataset

Training of neural network requires enormous number of training images to prevent overfitting and to make the network robust for unknown inputs. Since training of the network needs tens of thousands of training cycles, and labeling such an amount of unique was far beyond my capacity, I decided to generate training images automatically. Table 4.4 shows limits used for generating random data for training and testing.

| transformation type | T1 - soft transformation | T2 - hard transformation   |
|---------------------|--------------------------|----------------------------|
| rotation            | $\pm 0 - 10^\circ$       | $\pm 10^\circ - 170^\circ$ |
| borders             | 0 - 10%                  | 0 - 10%                    |
| blur                | 0 or 3                   | 0 or 3                     |
| brightness          | $\pm 0 - 50$             | $\pm 50 - 100$             |
| crop                | 0 - 20%                  | 0 - 20%                    |

Table 4.4: Transformation limits used for generating samples. Values of transformation were randomly chosen within T1 and T2 limits of each image when generating test data. *Rotation* transformation rotates image in either direction. *Borders* adds overlapping border over image, width of the border is given in percentage of width (height) of the image. *Blur* is Gaussian blur with kernel size 0 or 3. *Brightness* adds brightness value to each pixel. *Crop* cuts out sides of an image; as a result transformed image will be scaled.

Figure 4.3 shows random samples of similar and non-similar pairs of images generated for training. Similar images are generated from same image using transformation T1. Non-similar images are generated from same or different images using transformation T2. This approach was used to set a metric which implemented network should learn.

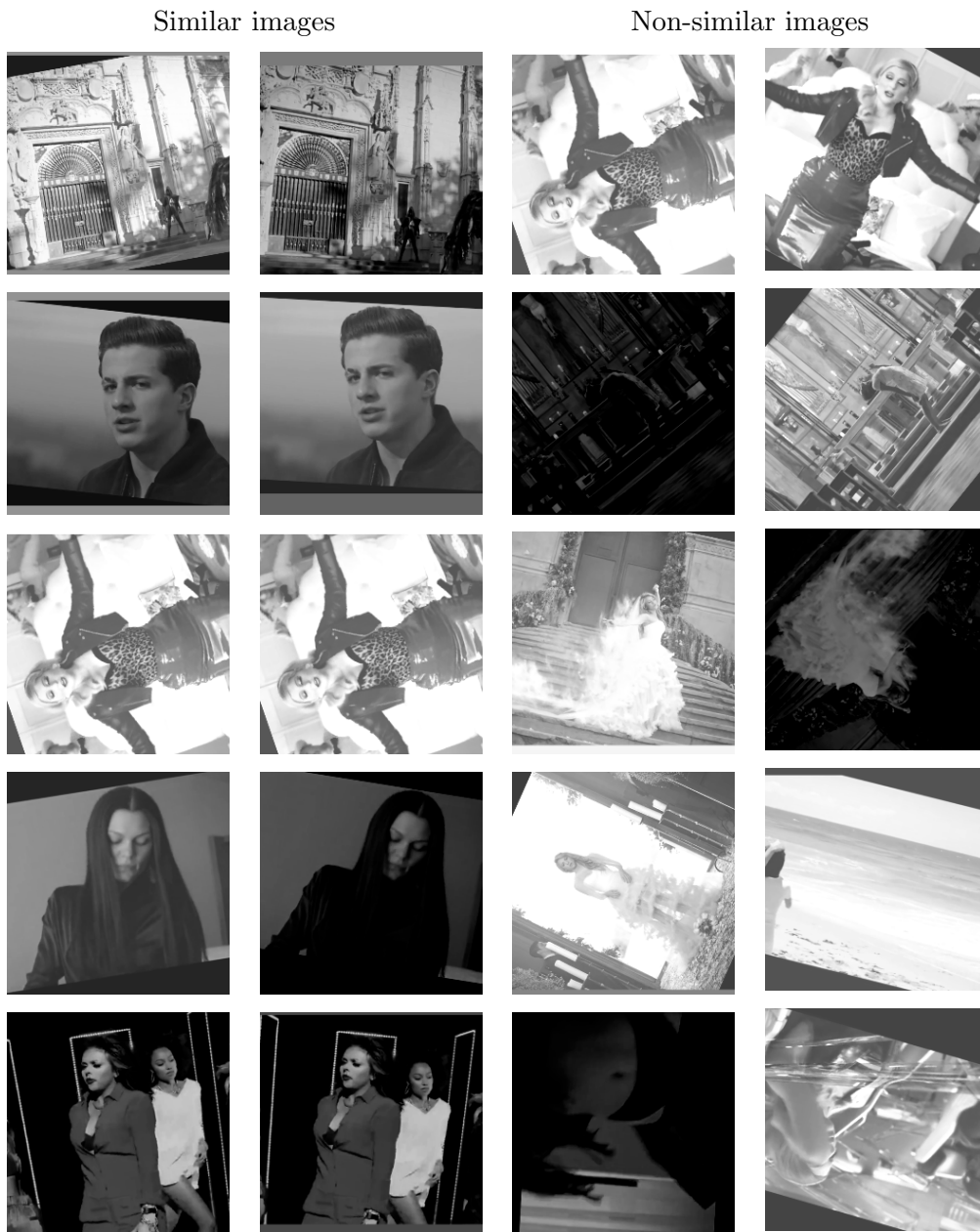


Figure 4.3: Examples of automatically generated similar and non-similar images.

#### 4.2.5 Experimental results

Following chapter summarize results of the best found model of neural network. Structure of network is described in table 4.5. The training of network started with learning rate 0.1. Learning rate was decreased by multiple of  $\gamma = 0.5$  in iterations

3000, 4000, and 4500. This method of controlled learning rate decrease is called "multi-step" learning rate.

For experimental purposes transformations applied to training dataset was also changed so that similar and non-similar transformations are more distant. Only rotation was adjusted in this case; similar images were rotated in range from 0 to 10 degrees, non-similar images were rotated in range from 40 to 80 degrees. Other transformations were not used. Main difference from previously defined transformations is that 30 degrees gap of uncertainty was allowed where the network is not forced to make strict decision about image similarity.

Figures 4.4 and 4.5 show train and test loss. In test loss one can see small drop of values in iterations where learning rate changes.

Figure 4.6 shows test accuracy for  $margin = 1$ .  $Acc0$  represents similar images; they are derived from the same source image and transformed with transformation T1. For better idea of the network capabilities non-similar images were divided into two classes. Accuracies  $acc1$  and  $acc2$  both represent non-similar images. Class  $acc1$  is derived from same images transformed with transformation T2, class  $acc2$  contains pair of different images.  $Acc$  is mean accuracy.

Network learns itself in only few hundreds iterations to correctly distinguish similar images. Biggest issue has the network with recognizing non-similar images derived from same images.

Figure 4.7 shows test accuracy of trained network for different values of margin. Margin represents interesting factor here. Margin is desired distance between non similar images used while computing contrastive loss function 3.4. Although the network was trained to make distance between non-similar images equal to one, highest accuracy can be achieved with margin close to 0.4. With this knowledge one can get even better performance from the network.



| layer number | layer type       | kernel size | number of outputs | stride |
|--------------|------------------|-------------|-------------------|--------|
| 1            | convolutional    | 3           | 16                | 1      |
| 2            | ReLU             |             |                   |        |
| 3            | pooling          | 3           |                   | 2      |
| 4            | convolutional    | 3           | 16                | 2      |
| 5            | pooling          | 3           |                   | 2      |
| 6            | LRN <sup>3</sup> |             |                   |        |
| 7            | inner product    |             | 128               |        |
| 8            | inner product    |             | 32                |        |

Table 4.5: Structure of most successful neural network.

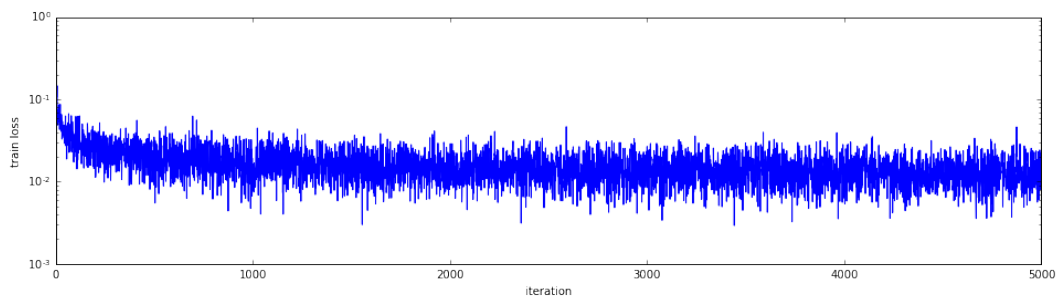


Figure 4.4: Train loss dependent on training iterations.



Figure 4.5: Test loss dependent on training iterations.

---

<sup>3</sup>LRN = Local Response Normalization

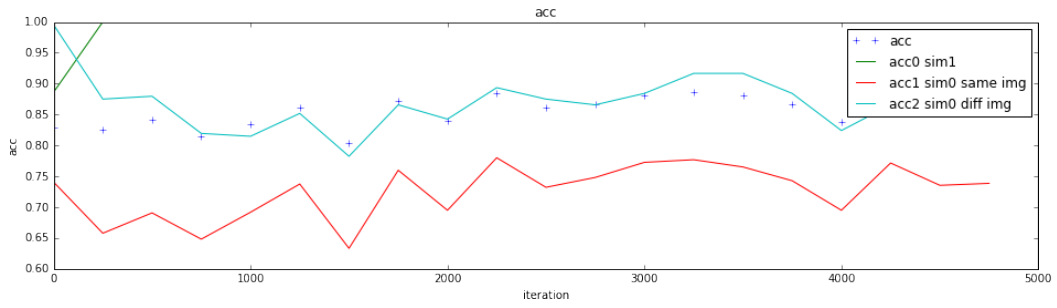


Figure 4.6: Test accuracy per class dependent on training iterations.

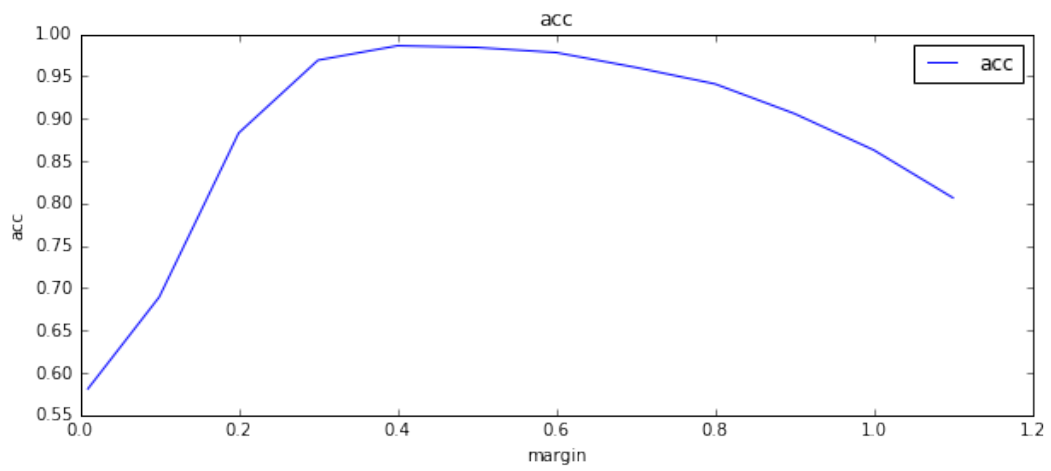


Figure 4.7: Accuracy dependent on margin. Meaning of margin was previously explained in 3.4.

## 4.2.6 Time consumption

Time consumption is crucial parameter while training large neural networks. It can easily take up to several days or weeks to train one network on CPU and by using optimized frameworks this time can be reduced to the range of minutes. Following tables show time consumption of training neural network with 10 000 iterations and technical specifications of used clusters. Caffe can use cuDNN [27] library high-performance GPU acceleration. *GPU with CUDA* in table 4.6 was measured with using cuDNN library.

| Cluster | CPU           | GPU           | GPU with CUDA |
|---------|---------------|---------------|---------------|
| doom    | 25 min 56 sec | 16 min 54 sec | 8 min 50 sec  |
| zubat   | 31 min 18 sec | 16 min 5 sec  | 8 min 5 sec   |

Table 4.6: Time consumption of training neural network with 10 000 iterations.

|         |  |
|---------|--|
| cluster | <b>doom</b>                            |
| CPU     | 2x 8-core Intel Xeon E5-2650v2 2.60GHz |
| GPU     | 2x nVidia Tesla K20 5GB                |
| RAM     | 64 GB                                  |
| disk    | 2x 1TB 10k SATA III, 2x480GB SSD       |
| cluster | <b>zubat</b>                           |
| CPU     | 2x 8-core Intel Xeon E5-2630v3 2.40GHz |
| GPU     | 2x nVidia Tesla K20Xm 6GB              |
| RAM     | 128 GB                                 |
| disk    | 2x 1TB 10k SATA III, 2x 480GB SSD      |

Table 4.7: Technical specification of clusters used for training neural networks.

## 4.3 Video matching

### 4.3.1 Video matching algorithm

Video matching algorithm is based on distance between two images. Distance of two images is computed from their feature vectors as an Euclidean distance in feature space. Similarity and non-similarity of two frames can be decided by thresholding their distance.

Matching algorithm returns cost matrix for two videos (in their hash codes). Cost matrix contains distance in feature space for each two frames in compared videos. Figure 4.9 shows cost matrices after thresholding.

### 4.3.2 Sequence verification

Video matching algorithm takes feature vectors computer either by Yang 4.1 or neural network 4.2. Feature vectors precomputed from videos in dataset are called reference. When new query video comes and has to be matched with a reference video, its hash code is computed in the same way and compared with each reference feature vector. Matching algorithm is based on euclidean distance between hash codes representing two frames. Two frames are considered the same if their Euclidean distance does not exceed certain threshold. Furthermore, to eliminate false matches, bigger number of adjacent frame matches is required to affirm a sequence match. Result of this process is similarity score of query video with each one reference video.

Image 4.8 shows result of matching algorithm applied on two identical videos. Values on axis represent frame number. Matched frames should ideally only be located on diagonal, however, there are some other false matches farther from diagonal and some false matches adjacent to diagonal. Distant matches are caused by stand-alone frames with same hash code. Matches close to diagonal are caused by small changes in video during short time where following frames are almost identical.

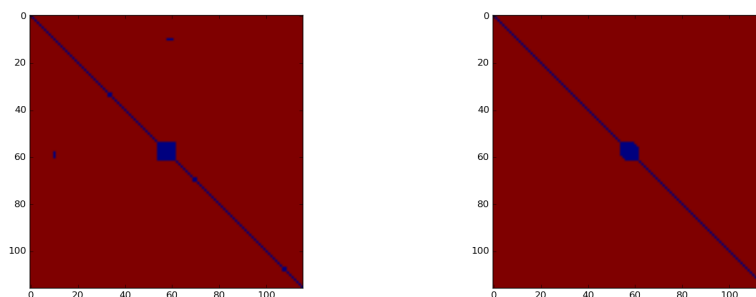


Figure 4.8: False matches pruning can be done by requiring minimal number of adjacent matches.

### 4.3.3 Results for Yang algorithm

Figure 4.9 shows detailed results of matching algorithm for different values of thresholding distance  $\epsilon$  and required length of adjacent similar frames. These results are sample results computed from only one pair of videos. However, very similar results were get from other videos too.  $\epsilon$  is distance between two frames in feature space; higher value of  $\epsilon$  will accept more false matches as similar frames. False matches can be filtered by required minimal length of adjacent similar frames. It can seem that setting large requirement on number of adjacent frames would solve the problem. But it is also important to consider that not always there will be this long video sequences in real life application.

Blue dots represent similar frames. Compared videos are original video and video with adjusted brightness. Videos have the same length. Ideally, blue dots should only be located on diagonal. Rows a - e vary thresholding distance  $\epsilon = \{12, 20, 30, 50, 70\}$ . These values define how close must features in feature space be to be considered similar. Columns I - IV vary minimal number of adjacent frames  $\{3, 7, 20, 40\}$  which must be similar in order to verify similarity. Value under each image is matching accuracy.

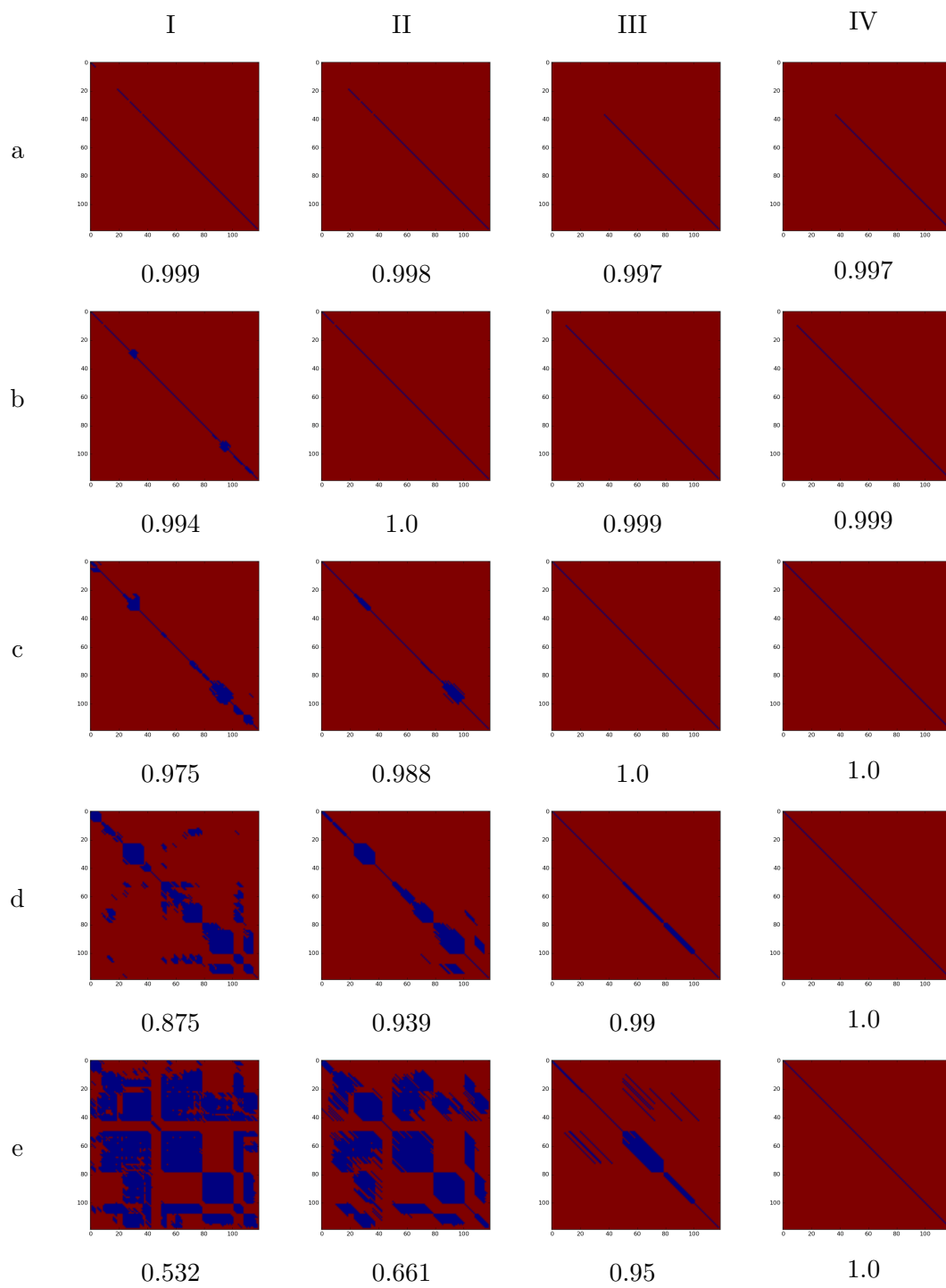


Figure 4.9: Results of video matching algorithm for SURF features.

## 4.4 Preprocessing

Video copy detection is computationally and timewise highly demanding process. Video sequence detection requires computing descriptors on user side and storing feature images on server side. It is assumed that each video sequence contains redundant information and can be compressed without any significant impact on matching result. Firstly, bitrate of each video in the dataset is limited to 800 kbps. Secondary, frame rate is reduced to 4 fps. Finally, while computing features, each frame is resized to 400 by 400 pixels.

Also it has to be considered that two copies of the same video sequence can differ in such things as frame resolution, frame rate, bitrate etc.; therefore, preprocessing is absolutely essential for such an algorithm to reduce video file size and unify videos from different sources.

Another reason for media preprocessing is real life application of the system. I assume that preprocessing can often be done locally on the side where data is stored. This gives a possibility of having a separated server which only receives a reduced amount of data and performs media matching. Delivering data in full resolution to the matching machine is not necessary.

## 4.5 Dataset

Commonly used databases for image processing such as Mnist [26], ImageNet [14] or Cifar10 [28] are not suitable for my task, however, they can be used for testing purposes and model verification.

Video sequence matching requires robustness against video transformation such as scale, mirroring, contrast and brightness adjustment, rotation, adding borders etc. Since I did not find a suitable dataset, one had to be made. As an input videos for the dataset I used 912 unique videos from four different playlists downloaded from YouTube. Each video was compressed to 800 mbps, split into 30 seconds long parts and frame rate was reduced to 4 frames per second. This returned 5006 short unique videos in total.

Each short video was kept in its original version and also was adjusted with several transformations. Figure 4.10 shows transformations used in the prepared dataset. Name of adjusted video fully describes its original source and all the transformations used to create the final video. Therefore, based only on names, one would be able to recreate the dataset. Complete dataset consists of 160 039 short videos.

For testing purposes I also created unique videos and long videos. Unique videos are only used once in the entire dataset and do not have any kind of transformation. Long video is made as a combination of one unique video followed by four randomly chosen reference videos.

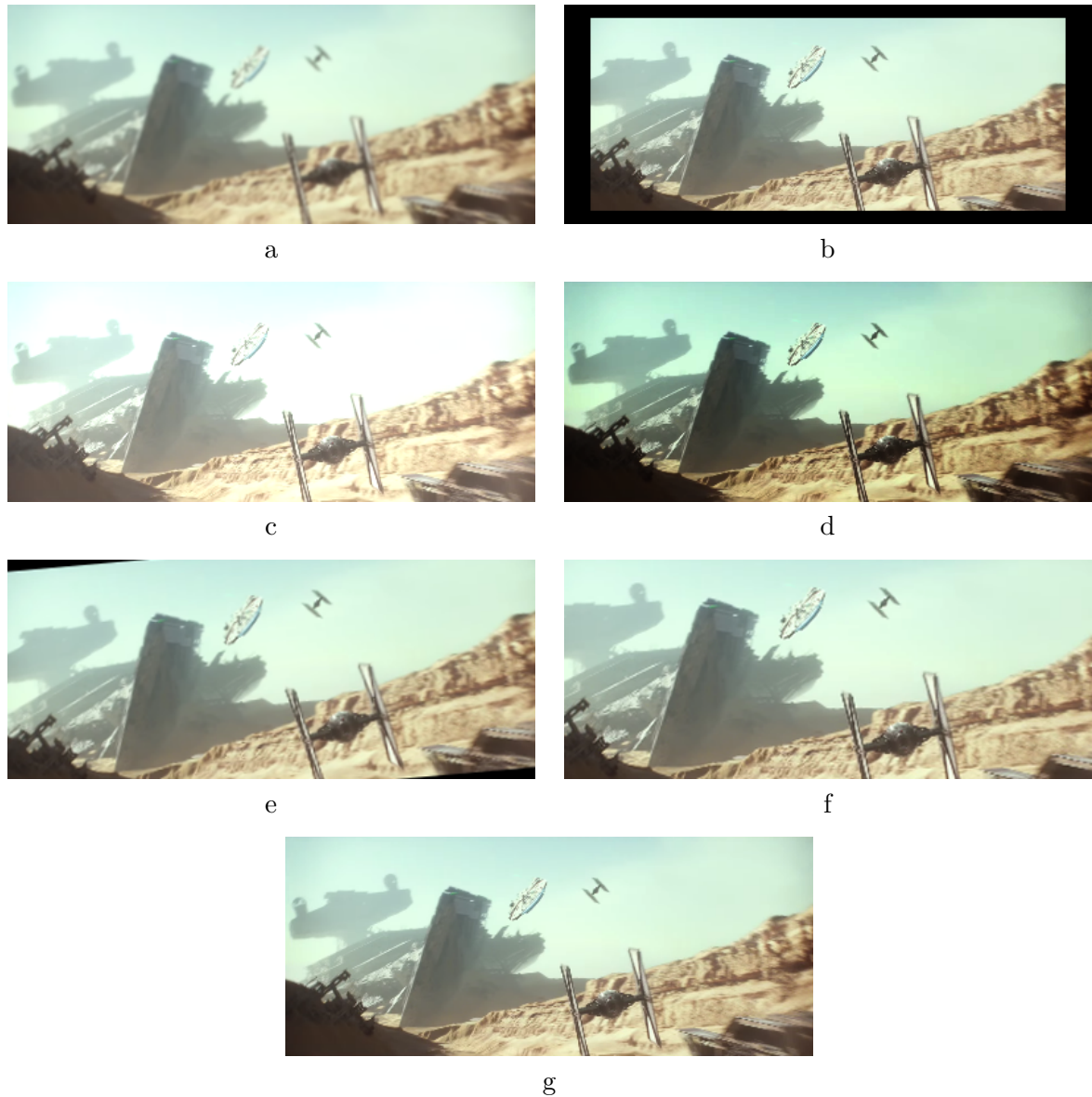


Figure 4.10: Transformations performed on videos stored in the dataset; (a) Gaussian blur, (b) overlapping black border, (c) brightness adjustment, (d) gamma correction, (e) rotation, (f) scale, (g) original image

Image 4.11 shows the process of creating reference and query videos of the dataset. Input and output files are shown in blue boxes. While developing the algorithm I had to store both videos and hash files. Storing of all the video files is highly memory consuming. Once the algorithm is complete, only the hash files can be kept and large



video files can be deleted.

### 4.5.1 Flowchart

Entire process of video matching can be divided into three steps. First step is creating short videos and computing their hash files. Dataset contains several transformed versions of each video (reference videos) and videos which are only used once (unique videos) while creating long videos. Unique videos were created from movies and no transformation was used while creating them.

- Video database - Database of media that will be included in final dataset and will be prepared for video matching. I used about 1 000 videos downloaded from YouTube. Downloaded videos are movie reviews and music videos.
- Video compression - Process of reducing video size by compressing bit-rate to 800 kbps.
- Video time change - Cutting videos into 30 seconds long parts and changing frame rate to 4 fps.
- Video transformation - Applying transformations described in sections 4.1 and 6.
- Reference videos - Output videos that are temporarily stored until hash codes are computed. Reference videos are large files. They can be deleted after all hash files are computed.
- Reference hash files - Hash codes computed either by Yang 4.1 or neural network 4.2. Hash codes are saved in small text files (or .h5py structure). Files with hash codes complete the dataset.

Second step is merging short videos into longer sequences and computing their hash files.

- Reference videos - Video files taken from previous step.
- Unique videos - Video files taken from previous step.
- Merge video - Combining one unique video and four randomly chosen reference video. Each unique video is only used once. in entire dataset.
- Query videos - Combined video files used for testing purposes.
- Query hash files - Hash codes computed either by Yang 4.1 or neural network 4.2.

Third step is actual matching of videos.

- Query hash files - Hash codes from previous step.
- Reference hash files - Hash codes from previous step.

- Cost matrix - Applying matching algorithm described in section 4.3.2.

In real life application query video would be the one that needs to be checked for matches against precomputed dataset. Its hash code would be directly computed and checked against database as described in step three.

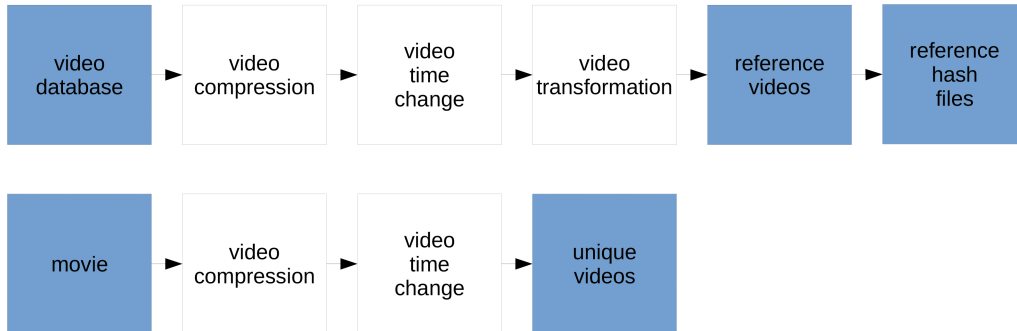


Figure 4.11: Creating short reference videos from input video database and short unique videos from movies. For reference videos hash files were computed and stored. Blue boxes represent data, white boxes represent processes.

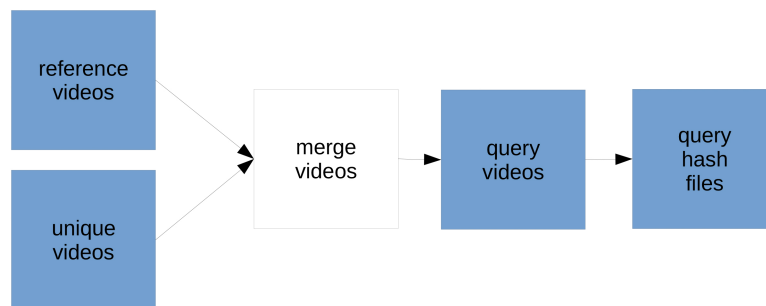


Figure 4.12: Combining reference and unique videos. Merged videos are stored as query videos and hash files are computed for them too. Blue boxes represent data, white boxes represent processes.

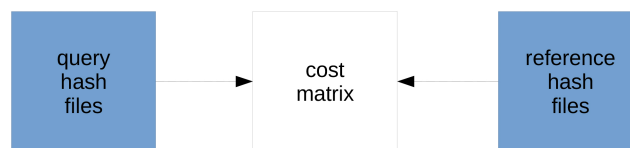


Figure 4.13: Finding matches between hash files. In this step no video files are needed, all data can be precomputed and saved as hash files.

Results of matching algorithm are summarized in 4.3.3 and 6. Thanks to pruning, matching algorithm does not return false matches. In section 6 this can be seen in

every confusion matrix on second row. False matches would appear as false positive values. Sensitivity of matching algorithm can be controlled by thresholding distance  $\epsilon$  and minimal number of adjacent similar frames. These values are explained in 4.3.3.

# 5 Conclusion

## 5.1 Summary

Performed experiments show that proposed algorithms are fully functional and capable of matching duplicate videos. As expected, Yang algorithm was capable of matching media that were not strongly damaged. I have also found that some kinds of transformations (brightness adjustment, soft blurriness) are easier to be covered by hashing and make system quite reliable, while others (rotation, contrast adjustment) make it impossible for the system to match similar media. Only way to bypass this nature of Yang based matching is storing enormous amount of possible transformations for each media.

Using siamese neural networks brought greater ability of generalizing fingerprints of transformed media. Although I did not perform test on as many transformations as with SURF algorithm from the final network, interim results gained for less accurate networks proved that siamese neural networks can be trained to retrieve similar fingerprints from transformed media.

## 5.2 Future work

For the future work there is a potential of further development in all parts of proposed algorithm. Common possible improvement for both proposed hashing algorithms can be done by upgrading them to spatio-temporal method. Idea of using fingerprints including evolution of media in time is briefly described in chapter 3.2. Beside this method, there are other possible ways of including time evolution that could be used.

### Yang algorithm

Yang based descriptors seem not to be generalizing enough. For this approach it is crucial to have rich scale of precomputed transformations stored in dataset. Collected results show that duplicate video can be successfully found if transformations made on it are very close to those precomputed and stored in the dataset. For creating really reliable dataset one would need to store much more possible transformations of each media

that has to be to matched. This is very inconvenient for time and data consumption requirements.

### **Siamese neural networks**

Neural networks seem to be promising solution for computing media fingerprints. Although, in feature work it would be necessary to train neural network on much bigger dataset and make sure that feature vector provided by neural network is general for frequent media transformations. In this work I did not have chance to connect neural network with proposed matching algorithm leaving it open for next experiments.

### **Matching algorithm**

Great deal of possible improvement lies in matching algorithm. The one proposed in this work is purely brute force algorithm. It gives some freedom of accepting more or less similar feature vectors and filtering matches by the length of found sequence, but the core of the algorithm still relies on comparing each single frame with each frame stored in the database. There are two main improvements that need to be done. First of them is clustering stored feature vectors in some way. For instance, by using spatio-temporal features one would be able to group short parts of videos that have similar changes in time. Once the clusters are separated, matching of newly coming media would only be performed against small part of entire dataset.

Second important part of improving matching algorithm is adding dynamic time warping. Current algorithm needs input media that have exactly the same frame rate. But due to existence of different video codecs it is very common that one video saved in different file formats can differ in length. These difference cannot be distinguished by a person while watching the video, however, for matching algorithm comparing frames one by one it makes a big problem which leads in not finding actual matches.

Matching algorithm was only tested with Yang based features. Siamese neural network based features have the same structure as Yang based features and easily be implemented into matching algorithm in future work.

# 6 Appendix

Following figures show detailed results of implemented YANG algorithm. Each table contains several parameters for every implemented transformation. These results are the main conclusion of YANG algorithm.

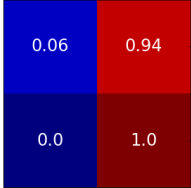
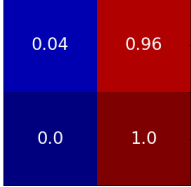

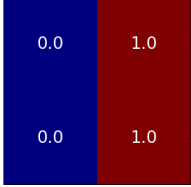
| Transformation | Accuracy | F <sub>1</sub> score | Confusion matrix   |
|----------------|----------|----------------------|--|
| Rotation 3°    | 0.996    | 0.044                |   |
| Rotation 5°    | 0.998    | 0.051                |  |
| Rotation 7°    | 0.998    | 0.043                |  |
| Rotation 9°    | 0.998    | 0.002                |  |

Table 6.1: Sensitivity for rotation.

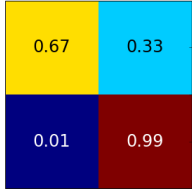
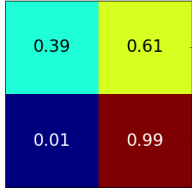
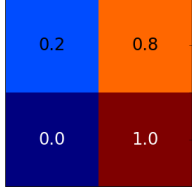
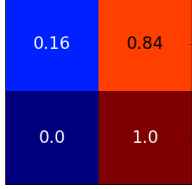
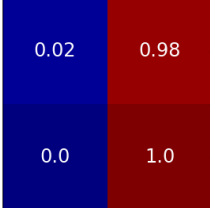
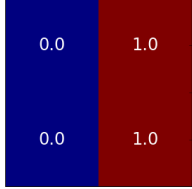
| Transformation          | Accuracy | F <sub>1</sub> score | Confusion matrix  |
|-------------------------|----------|----------------------|---|
| Blur, kernel size 3 px  | 0.994    | 0.228                |    |
| Blur, kernel size 5 px  | 0.994    | 0.148                |    |
| Blur, kernel size 7 px  | 0.995    | 0.111                |   |
| Blur, kernel size 9 px  | 0.996    | 0.105                |  |
| Blur, kernel size 13 px | 0.998    | 0.032                |   |
| Blur, kernel size 17 px | 0.999    | 0.003                |  |

Table 6.2: Sensitivity for Gaussian blur.

| Transformation                 | Accuracy | F <sub>1</sub> score | Confusion matrix |
|--------------------------------|----------|----------------------|------------------|
| Overlapping border, 1 percent  | 0.994    | 0.289                |                  |
| Overlapping border, 2 percent  | 0.994    | 0.257                |                  |
| Overlapping border, 5 percent  | 0.994    | 0.055                |                  |
| Overlapping border, 10 percent | 0.999    | 0.022                |                  |
| Overlapping border, 15 percent | 0.999    | 0.003                |                  |

Table 6.3: Sensitivity for border.



| Transformation  | Accuracy | F <sub>1</sub> score | Confusion matrix |
|-----------------|----------|----------------------|------------------|
| Brightness - 30 | 0.999    | 0.071                |                  |
| Brightness - 20 | 0.997    | 0.112                |                  |
| Brightness - 10 | 0.995    | 0.246                |                  |
| Brightness + 10 | 0.994    | 0.324                |                  |
| Brightness + 10 | 0.994    | 0.309                |                  |
| Brightness + 30 | 0.995    | 0.331                |                  |

Table 6.4: Sensitivity for brightness.

| Transformation       | Accuracy | F <sub>1</sub> score | Confusion matrix |
|----------------------|----------|----------------------|------------------|
| Gamma correction 0.7 | 0.999    | 0.003                |                  |
| Gamma correction 0.8 | 0.998    | 0.124                |                  |
| Gamma correction 0.9 | 0.995    | 0.177                |                  |
| Gamma correction 1.1 | 0.997    | 0.357                |                  |
| Gamma correction 1.2 | 0.999    | 0.116                |                  |
| Gamma correction 1.3 | 0.999    | 0.033                |                  |

Table 6.5: Sensitivity for gamma correction.

| Transformation | Accuracy | F <sub>1</sub> score | Confusion matrix |
|----------------|----------|----------------------|------------------|
| Scale 0.8      | 0.999    | 0.003                |                  |
| Scale 0.9      | 0.998    | 0.041                |                  |
| Scale 0.95     | 0.995    | 0.061                |                  |
| Scale 1.05     | 0.996    | 0.074                |                  |
| Scale 1.1      | 0.998    | 0.098                |                  |
| Scale 1.2      | 0.999    | 0.003                |                  |

Table 6.6: Sensitivity for scale.

# Bibliography

- [1] Changick Kim Changick Kim and Bhaskaran Vasudev. Spatiotemporal sequence matching for efficient video copy detection. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(1):127–132, 2005. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1377368>, doi: 10.1109/TCSVT.2004.836751.
- [2] Gaobo Yang, Ning Chen, and Qin Jiang. A robust hashing algorithm based on SURF for video copy detection. *Computers and Security*, 31(1):33–39, 2012. URL: <http://dx.doi.org/10.1016/j.cose.2011.11.004>, doi:10.1016/j.cose.2011.11.004.
- [3] Alexis Joly, C Frélicot, and Olivier Buisson. Content-based video copy detection in large databases: A local fingerprints statistical similarity search approach. *Image Processing, 2005. ICIP ...*, (2):8–11, 2005. URL: [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=1529798](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1529798), doi:10.1109/ICIP.2005.1529798.
- [4] Arun Hampapur and Ki-ho Hyun. Comparison of Sequence Matching Techniques for Video Copy Detection.
- [5] Mei-chen (University of California) Yeh and Kwang-Ting (University of California) Cheng. Video copy detection by fast sequence matching. *Proceeding of the ACM International Conference on Image and Video Retrieval - CIVR '09*, page 1, 2009. URL: <http://portal.acm.org/citation.cfm?doid=1646396.1646449>, doi:10.1145/1646396.1646449.
- [6] L. Scott Harrell. What is Video Freebooting and Can It Be Prevented? URL: <http://vtrep.com/what-is-video-freebooting-can-i-prevent-it/>.
- [7] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF : Speeded Up Robust Features.
- [8] Andrew Ng. Coursera - Machine Learning Course. URL: <https://www.coursera.org/learn/machine-learning>.
- [9] Edgar Simo-serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-noguer. Discriminative Learning of Deep Convolutional Feature Point Descriptors. 2014.

- [10] Ian Bengio, Goodfellow Yoshua, and Aaron Courville. *Deep Learning*. 2016. URL: <http://www.deeplearningbook.org>.
- [11] Alex Krizhevsky and Geoffrey E Hinton. 4824-Imagenet-Classification-With-Deep-Convolutional-Neural-Networks. pages 1–9. doi:<http://dx.doi.org/10.1016/j.protcy.2014.09.007>.
- [12] Deeplearning.net. URL: <http://deeplearning.net/>.
- [13] S Lawrence, C.L. Giles, Ah Chung Tsoi, and A.D. Back. Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=554195>, doi:10.1109/72.554195.
- [14] Prof. Li Fei-Fei, Prof. Kai Li, Olga Russakovsky, Jonathan Krause, Prof. Jia Deng, and Prof. Alex Berg. ImageNet. URL: <http://image-net.org/index>.
- [15] Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. LabelMe: a database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1-3):157–173, 2008. URL: <http://people.csail.mit.edu/brussell/research/AIM-2005-025-new.pdf>.
- [16] Jane Bromley, James W. Bentz, Léon Bottou, Isabelle Guyon, Yann Lecun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature Verification Using a “Siamese” Time Delay Neural Network. *International Journal of Pattern Recognition and Artificial Intelligence*, 07(04):669–688, 1993. doi:10.1142/S0218001493000339.
- [17] Caffe Contrastive Loss Layer. URL: [http://caffe.berkeleyvision.org/doxygen/classcaffe\\_1\\_1ContrastiveLossLayer.html](http://caffe.berkeleyvision.org/doxygen/classcaffe_1_1ContrastiveLossLayer.html).
- [18] Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093*, 2014. URL: <http://caffe.berkeleyvision.org/>.
- [19] Cuda-convnet. URL: <https://code.google.com/p/cuda-convnet/>.
- [20] Jeff Donahue, Yangqing Jia, Eric Tzeng, Trevor Darrell, Oriol Vinyals, Judy Hoffman, and Ning Zhang. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. URL: <http://arxiv.org/pdf/1310.1531.pdf>.
- [21] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal L. Jozefowicz, and Xiaoqiang Zheng. TensorFlow. URL: <http://download.tensorflow.org/paper/whitepaper2015.pdf><https://www.tensorflow.org/>.

- [22] Yoshua Bastien Frederic and Lamblin, Pascal and Pascanu, Razvan and Bergstra, James and Goodfellow, Ian J. and Bergeron, Arnaud and Bouchard, Nicolas and Bengio. Theano: new features and speed improvements, 2012. URL: <http://arxiv.org/pdf/1211.5590.pdf>.
- [23] Fabian Pedregosa, Olivier Grisel, Ron Weiss, Alexandre Passos, and Matthieu Brucher. Scikit-learn : Machine Learning in Python. 12:2825–2830, 2011. URL: [http://scikit-learn.org/stable/auto\\_examples/model\\_selection/plot\\_precision\\_recall.html](http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html), arXiv:arXiv:1201.0490v2.
- [24] Caffe Siamese Network. URL: <http://caffe.berkeleyvision.org/gathered/examples/siamese.html>.
- [25] Caffe ImageNet. URL: <http://caffe.berkeleyvision.org/gathered/examples/imagenet.html>.
- [26] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. THE MNIST DATABASE.
- [27] NVIDIA CUDA. URL: <https://developer.nvidia.com/cudnn>.
- [28] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images, 2009. URL: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>.