FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Lip Reading using Deep Neural Networks
Student:	Jan Horák
Supervisor:	doc. Ing. Pavel Kordík, Ph.D.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2018/19

Instructions

Survey state of the art in the area of image and video recognition. Design and implement a real-time system capable of lip reading from video. Evaluate the accuracy on simplified word classification task and explore the possibility to recognize sentences in real-time. Build a prototype that present capabilities of deep learning algorithms in the browser.

References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague December 21, 2017

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Lip Reading using Deep Neural Networks

Jan Horák

Department of Theoretical Computer Science
Supervisor: doc. Ing. Pavel Kordík, Ph.D.

May 13, 2018

Acknowledgements

I would first like to thank my thesis advisor doc. Ing. Pavel Kordík, Ph.D. for his expert advice and encouragement throughout this thesis.

I am also very grateful to Rob Cooper at BBC Research for help in obtaining the dataset.

Finally, I must express my very profound gratitude to my parents, sister and to my girlfriend for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them. Thank you.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 13, 2018 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Horák, Jan. *Lip Reading using Deep Neural Networks.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018. Also available from: ⟨`http://janhorak.info/lip`⟩.

Abstrakt

Problém odezírání ze rtů, tedy umění odhadu vysloveného slova, popř. celé věty, pouze z vizuální informace, je vzhledem k vysoké různorodosti artikulace lidí, počtu jazyků a slov v každém z nich, velmi složitá, ale zajímavá úloha.

V této Bakalářské práci analyzuji doposud známé způsoby odezírání ze rtů, zjišťuji jejich přesnosti a mou snahou je ověření zda je použití metod umělé inteligence, konkrétně hlubokých neuronových sítí, vhodným kandidátem pro řešení tohoto problému. V praktické části se zaměřuji na prezentaci výsledků a to jednak v podobě přesnosti mnou vytrénované neuronové sítě na testovacích datech, jednak vytvořením webové aplikace pro zjištění, jak náročné by bylo takový nástroj využít v praxi pro rozpoznávání řeči v reálném čase metodou odezírání ze rtů.

Klíčová slova odezírání ze rtů, počítačové vidění, neuronové sítě, hluboké neuronové sítě, 3D konvoluce, detekce objektů, předzpracování dat, Python, Keras

Abstract

The problem of lip reading, which means a skill of guessing one's uttered word or whole sentence only out of a visual information, is a very hard - yet interesting task, due to variety of people, their languages and articulations.

In this bachelor thesis I analyze the known methods of lip reading, I find their accuracy and my aim is to verify whether the use of artificial intelligence methods, namely Deep Neural Network, is a suitable candidate for solving this problem. In the practical part, I focus on presenting the results both in terms of the accuracy of the trained neural network on test data and by creating and publishing a web application to find out how difficult it would really be to use such a tool for a real-time speech recognition using the lip reading method.

Keywords lip reading, computer vision, neural networks, deep neural networks, 3D convolutions, object detection, data preprocessing, Python, Keras

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

1.1 Motivation

The problem of lip reading is a very present topic that has not yet been fully resolved and is a great challenge for solving using artificial intelligence and machine learning methods. The art of lip reading could be used to help hearing people with disabilities by enhancing speech recognition in noisy areas, or possibly by security forces in situations where it is necessary to identify a person's speech when the audio record is not available. Given the number of languages, the vocabulary of each and very diverse articulation across people, it is impossible to manually create a computer algorithm that can be reading from the lips. Even human professionals in this field are able to correctly estimate just about every second word [1, p. 3] and only under ideal conditions. Therefore, the problem of lip reading is a perfect candidate for solution using Artificial Intelligence (AI).

I chose this topic, because I am very interested into Machine Learning (ML) and AI in general and I wanted to further extend my skills and knowledge by working on a practical and interesting project in this field. I took the advantage to work on this project by signing into Datalab Summercamp 2017[1] last year, where I could research this topic by proposing to follow up and replicate the results of Lip Reading in Wild (LRW) [2] achieved by J. S. Chung and A. Zisserman at University of Oxford.

[1] `https://datalab.fit.cvut.cz/events/95-summer-camp-2017`

1.2 Goals

The goals of my thesis are following:

1. Survey state of the art in the area of image and video recognition.
2. Design and implement a real-time system capable of lip reading from video.
3. Evaluate the accuracy on simplified word classification task.
4. Explore the possibility to recognize sentences in real-time.
5. Build a prototype that present capabilities of deep learning algorithms in the browser.

1.2.1 Structure of thesis

In this thesis I go through and describe a complete way how to preprocess the LRW dataset, design a Neural Network architecture capable of lip-reading uttered words and deploying this system on a website.

In the following chapter (**Chapter 2**), I research and describe building methods and algorithms suitable for Automated Lip Reading (ALR) systems, state of the art in image and video recognition in general, and the previous approaches in the field of computer vision.

Next, in **Chapter 3**, I describe some of the applicable tools and libraries, which implements previously described methods and are therefore suitable candidates to be used for the implementation of ALR.

Finally, in **Chapter 5**, I write down the process of preprocessing the LRW dataset, building and training a Deep Neural Network capable of recognizing a lip-articulated words using the tools mentioned in **Chapter 3**. I show the resulting performance and accuracy of the trained model by comparing various trained models of different architectures and parameters, and I design and build a website application where the final model is deployed to and bound to work with a web-camera using JavaScript, so that users can try the lip reading on their own computers and devices in real-time.

Chapter 2

Research

In the past years, researchers have made great advances in a field of computer vision. At the beginning, I highlight the three most common task in computer vision. These are:

- Image Classification.
- Object Detection.
- Instance Segmentation.

Differences are illustrated in Figure 2.1.

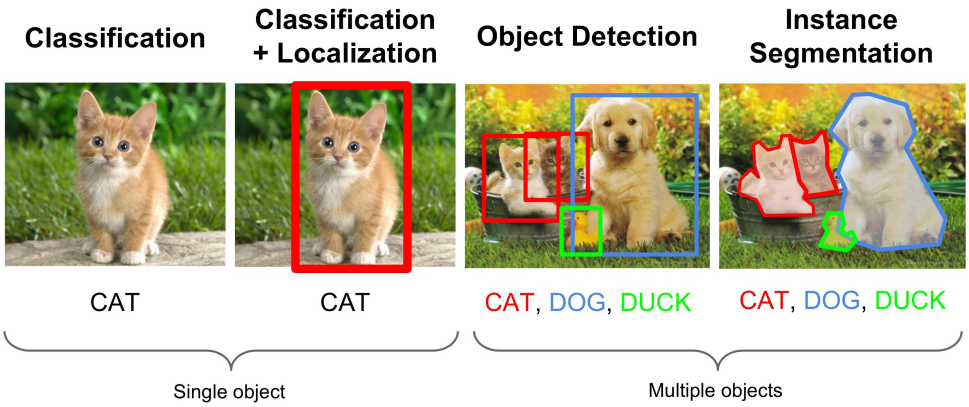


Figure 2.1: Comparison between image classification, object detection and instance segmentation. Source: [3]

In the case of *Object Classification*, sometimes also called *Object Recognition*, we handle the task of determining what kind of object is most likely in the image.

In the past, features such as HOG, Haar (Section 2.1), histograms, etc. combined with machine learning algorithms like Support Vector Machines or Neural Network have been used to address this problem [4, 5].

However, in 2012 team of University of Toronto scientist created an algorithm and won the ImageNet Large Scale Visual Recognition Challenge by using deep Convolutional Neural Networks to classify 1.2 million high-resolution images into 1,000 distinct classes, reducing the top-5 error rate by 11% [6]. Every year since then, deep learning models have dominated the challenges and even surpassed human performance in the field of image recognition [7].

2.1 Detection using Haar Cascades

In 2001 Paul Viola and Michael Jones published a paper *"Rapid Object Detection using a Boosted Cascade of Simple Features"* [8] describing an effective and robust object detection method using Haar feature-based classifiers.

They provided three key contributions:

- Integral image
- AdaBoost-inspired learning algorithm
- Attentional Cascade method

Their proposed solution made possible to run object detection with very good precision in real-time (15 FPS on a 700 MHz processor)

The algorithm is needs a lot of positive images (images of object to be detected) and negative images (e.g. background) to train the classifier. From these images *Haar features* (see Figure 2.2) are extracted. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle. [9]

2.1.1 Integral Image

Integral image is a an image we get by cumulative addition of values of neighbor pixels above and to the left for every pixel in the image.

$$I(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{2.1}$$

where $I(x,y)$ is the integral image and $i(x,y)$ is the original image. This could be computed efficiently in single pass over the image by using following recurrence:

$$I(x,y) = i(x,y) + I(x,y-1) + I(x-1,y) - I(x-1,y-1) \tag{2.2}$$

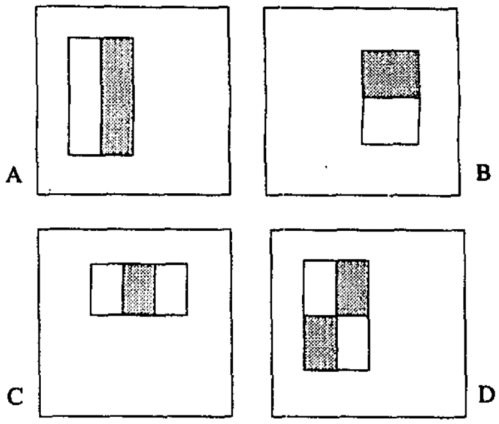


Figure 2.2: Example rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature. [8]

This helps to greatly reduce the computational time, and hence speed up the detection, because by calculating integral image first we can obtain the sum of the pixels lying within the rectangle only by referring the rectangle corner points. To get the sum of $i(x, y)$ over the rectangle spanned by A, B, C and D, we calculate:

$$\sum_{\substack{x_0 \leq x \leq x_1 \\ y_0 \leq y \leq y_1}} i(x, y) = I(A) + I(D) - I(C) - I(B), \tag{2.3}$$

where $A = (x_0, y_0)$, $B = (x_1, y_0)$, $C = (x_0, y_1)$ and $D = (x_1, y_1)$.

2.1.2 Attentional Cascade

When detecting object in images an *Attentional Cascade method* is applied to each area of the image, which provides an order of evaluating and checking for features, to prevent computing all features for each area in the image, since we expect most of the parts to not be the object. The importance of them is trained using an adaptive boosting algorithm (known as *AdaBoost*). This cascading method ensures that redundancy computing is kept to minimum.

2.2 Neural Networks

Neural Network is a computer system heavily inspired by the human brain and nervous system. It consists of series of algorithms that attempts to identify underlying relations in set of data and are able to adapt and learn from provided data to produce best possible result. By adding a so called hidden layer/s between input and output layer, we can, with sufficient and hidden layer/s width, represent any function. Due to this reason Multilayer Feedforward Networks are often called as *Universal Approximators* [10].

2.2.1 Convolutional Neural Networks

A Convolutional Neural Network (CNN or ConvNet) is a class of ordinary Neural Network, designed with assumption that the inputs are images, allowing us to encode certain properties into the architecture. The problem with regular Neural Network is that they don't scale well to images, because even for a low resolution images, when using fully-connected structure of layers, we would have a very big numbers of weights/parameters, which would quickly lead to *overfitting* (explained in following Section 2.2.4). [11]

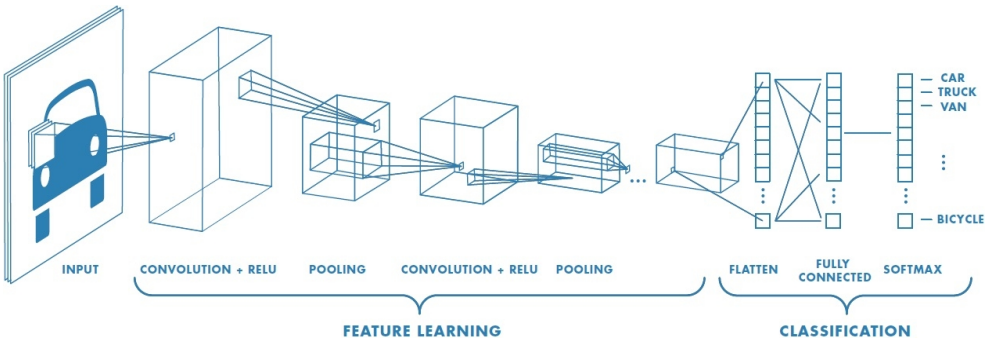


Figure 2.3: Example of a network with many convolutional layers. Filters are applied to each training image at different resolutions, and the output of each convolved image is used as the input to the next layer. Source: [12]

As can be seen on Figure 2.3 there are four main operations in the CNN:

- Convolution
- Non Linearity (ReLU)
- Pooling
- Classification (Fully Connected Layers)

These operations are the basic building blocks of every Convolutional Neural Network.

2.2.1.1 Convolution

The primary purpose of Convolution in case of a CNN is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

Every image is considered by a computer as an matrix of numbers (where numbers specify pixel color intensity). We create a Convolution Layers by convolving or *"sliding"* a filter (sometimes referred to as a kernel) by N pixels (also called stride) across the input image, where the current region bellow the filter is called receptive field, and multiplying the the values in the filter with the original pixel values of the image, thus computing element wise multiplications. We add the multiplication outputs to get the final integer which forms a single element of the output matrix. The final output matrix is called Convolved Image, Activation Map or Feature Map. [13]

As an example, considering an I as an input image, matrix K as a filter/kernel of size $h \times w$, we can compute the Convolved Image $I * K$ as

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} * I_{x+i-1,y+j-1} \tag{2.4}$$

which can be illustrated with Figure 2.4.

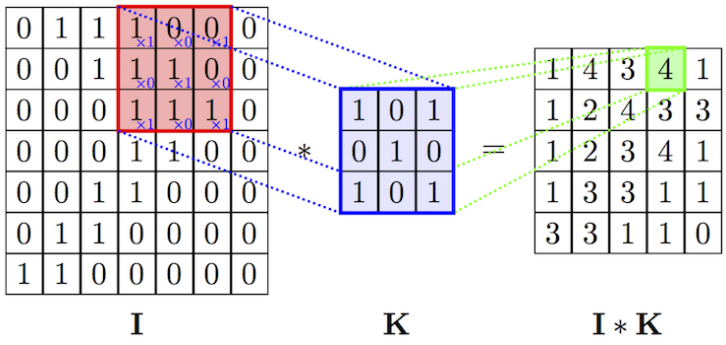


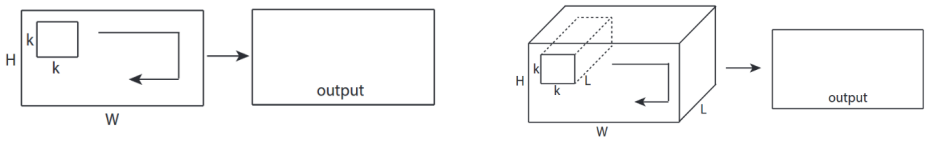
Figure 2.4: Diagrammatical overview of the above Formula 2.4 and the result of applying convolution over an image. [14]

2.2.1.2 2D vs 3D Convolutions

When applying convolutions filter the input shape and filter shapes are important. In case of images we generally have either a 2D matrix of pixel values with dimensions $height \times width$, representing a grayscale image, or a 3D matrix of shape $height \times width \times 3$, in case of colour image with red, green and blue channels.

When handling a grayscale image with input shape of $W \times H$, a kernel of shape $k \times k$ is used convolving in two directions (x, y) across the image, and 2D output matrix is obtained - Figure 2.5a.

With colour image, the input is $W \times H \times L$, ($L = 3$ channels), a kernel of shape $k \times k \times L$ is used and the convolving is still performed in two directions (x, y), thus we again get a 2D matrix as an output - Figure 2.5b.



(a) 2D Convolution with 2D input

(b) 2D Convolution with 3D input

Figure 2.5: Image convolving. Source: [15]

When building a ConvNet for classifying 3D objects, represented as a point cloud or 3D mesh, or when classifying a video, which can be imagined as an image sequence that can be stacked, we can represent the input layer by a 3D matrix. We can then use 3D convolutions (input of shape $W \times H \times L$, and a kernel of shape $k \times k \times d$, where $d < L$). Since kernel depth is smaller than the depth of the input volume, we convolve in three directions (x, y, z) and therefore the output will also be a 3D volume. See Figure 2.6 bellow. [15]

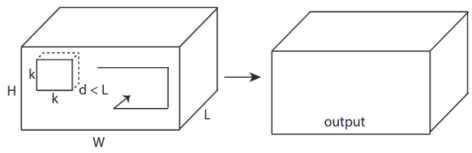


Figure 2.6: Video, 3D object convolving. Source: [15]

2.2.2 Pooling Layer

Pooling layer is another essential block of well-performing CNN architecture. Pooling is also referred to as a downsampling. Several options exists (*Average Pooling*, *L2-Norm Pooling*), but the one being the most used and popular is *Max-Pooling*. Max-Pooling is an operation, where a filter convolves around each subregion (again with specified stride), and outputs the maximum number of this subregion to the output matrix. To better illustrate this process, see Figure 2.7.

Pooling layers are used in part to reduce *overfitting* by providing an abstracted form of the representation. It also reduces the computational cost by reducing number of parameters to learn. Lastly, it provides basic translation

invariance to the internal interpretation, which is the reason some networks, where object specific positions are needed, don't use them at all. [14]

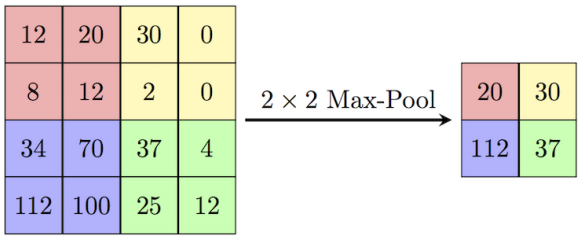


Figure 2.7: Max-Pooling with filter of size 2×2. Source: [14]

2.2.3 CNN Architecture and Training

As seen on Figure 2.3, Convolution Neural Network is commonly made up of only three layer types: *Convolution* (CONV), *Pooling* (POOL), and *Fully Connected* (FC) layers, where CONV and POOL layers are ordinarily repeated several times to create a Deep Neural Network and extract high-level features.

A Fully Connected layer is a normal Multi Layer Perceptron that uses a *softmax* activation function in the output layer.

The whole architecture is then trained by updating and adjusting filters/weights in the Neural Network though a training process called *back-propagation* in the similar way as with a normal NN. [13]

2.2.4 Underfitting and Overfitting

Not only when using Neural Networks, but when training any model with machine learning methods, it is important to make sure that the resulting model is not underfit or overfit to the data.

Variance refers to how much a model changes in response to the training data.

Bias refers to how much a model ignore the data.

Finding a good balance of *bias* vs *variance* is a critical concept in any Machine Learning modeling. *Underfitted* model has low variance and high bias, and thus the model is generalizing too much and fails to learn the underlying relationship between inputs and outputs.

In case of *overffiting*, model having high variance and low bias, results the model is too much relying on the training data and may therefore fail to fit additional data or predict future observations reliably.

Typically the dataset is being split into three parts:

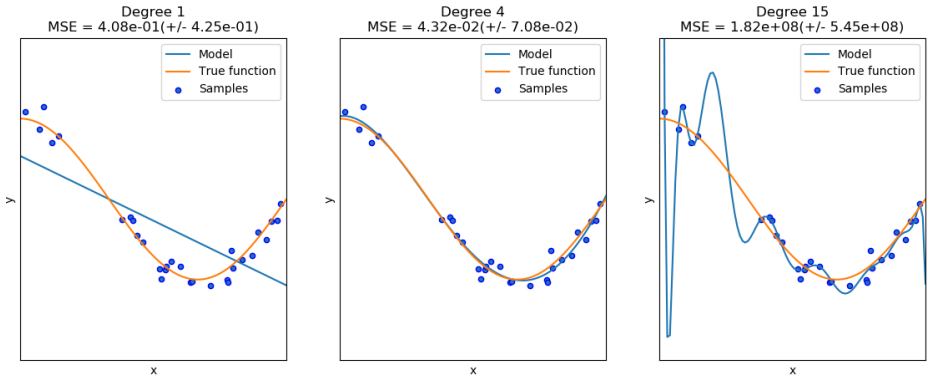


Figure 2.8: Underfitting (**left**), Good fit (**middle**), Overfitting (**right**). Source: [16]

- Training set.
- Validation set.
- Testing set.

When training the Neural Network, or any other Machine Learning model, we feed the model with data from training set, to update the parameters (weights in case of NN) and preform a *cross-validation*[2] with data from *validation set* to compare the performances of the prediction what were created on the training set. After we finish training we perform a prediction on our *test set* in order to see the accuracy on unseen data. Both overfitting and underfitting cause poor generalization on the *test set*. [17]

2.2.5 Normalization

Normalization, known as feature scaling can be an important preprocessing step for many machine learning algorithms. By normalizing our data we put them on the same scale, which can significantly improve the ability for model to learn if the scales for different features are very different.

2.2.5.1 Standardization

An effective normalization technique is *standardization*, involving scaling the values of each feature in the data to have zero-mean and unit-variance:

$$x' = \frac{x - \bar{x}}{\sigma} \tag{2.5}$$

[2] `https://www.cs.cmu.edu/~schneide/tut5/node42.html`

where x is the original feature vector, $\bar{x}$ is the mean of that feature vector, and σ is its standard deviation.

2.2.5.2 Batch Normalization

Batch Normalization of Neural Network is one way to solve or minimize the problem of covariance shift, that is, the change in the distribution of inputs to a unit as training progresses, and therefore improve the training. By normalizing these inputs to have zero mean and unit variance, training can be drastically sped up. If we consider a single unit in a Neural Network, then its output is given by

$$y_N(X; \mathbf{w}, b) = g(X\mathbf{w} + b), \tag{2.6}$$

where g is a nonlinearity, such as the rectified linear function (ReLU) or a sigmoid, X is the input, and w and b are the learned weights and bias. In a a Convolutional Neural Network, the weights can be shared with other units.

With Batch Normalization, instead the input to the nonlinearity is normalized,

$$y_B(X; \mathbf{w}, \gamma, \beta) = g(\frac{X\mathbf{w} - \mu(X\mathbf{w})}{\sigma(X\mathbf{w})}\gamma + \beta), \tag{2.7}$$

where the mean μ and standard deviation σ are computed given a batch X of training data. At test time, the values of μ and σ are fixed. The extra parameters γ and β are needed to still be able to represent all possible ranges of inputs to g. [18]

2.2.6 Regularization

Deep Neural Network are involved with a large number of parameters, and tends to overfit easily. *Regularizing* Neural Network is an important task to reduce overfitting and many algorithms have been presented to handle this problem.

2.2.6.1 Data Augmentation

Training a good model requires a lot of data. Lack of the labeled training data leads to poor generalization (see Section 2.2.4) and in general the more data we have the better the model will be. Obtaining them, however, is often not an easy task and every data collection process is associated with a cost. This cost can be in terms of dollars, human effort, computational resources and off course time consumed in the process. The augmentation can help, by extending the current training dataset with modified (augmented) samples. These new samples are created by various transformations of current labeled sample, leaving label the same. Type of augmentation depends on the type of the data. In case of images, we can rotate the original image, blur it, add noise, change lighting conditions (brightness, contrast), scale, shift or crop it

differently, so for one image we can generate different sub-samples. See Figure 2.9 for an image augmentation example. In audio, we can stretch or narrow the sample, change speed, pitch, volume or add noise.

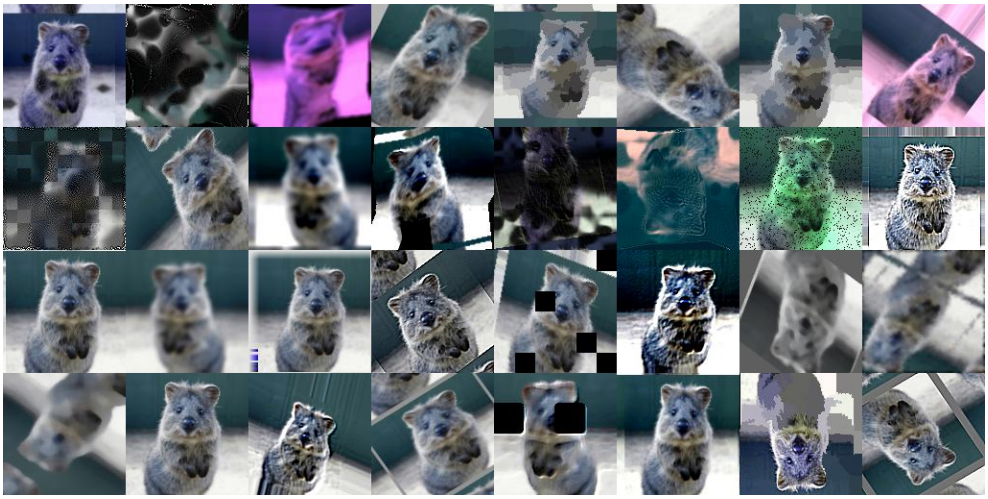
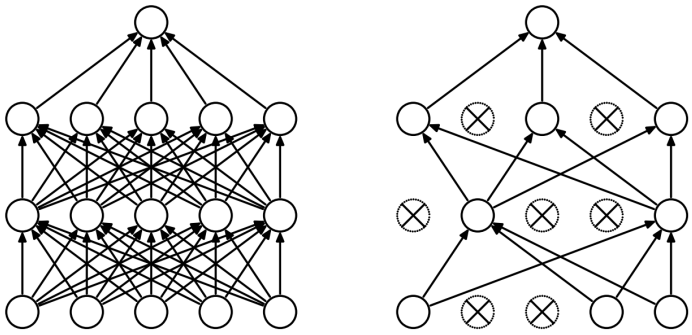


Figure 2.9: Example of heavily augmented samples of an image generated with Imgaug tool. Source: [19]

2.2.6.2 Dropout

Dropout is a very common technique to reduce overfitting. It consists of setting to zero the output of each hidden neuron with a certain probability. This way these *"dropped out"* neurons do not contribute to the forward pass and the Neural Network is forced to learn more robust features by finding an other activation path with conjunction of other neurons. Dropout can be illustrated by Figure 2.10. [20, 6]



(a) Standard Neural Net (b) After applying dropout.

Figure 2.10: Dropout in Neural Network. Source: [20]

2.3 State of the Art

There are lots of new proposed architectures of Convolutional Neural Networks being published nowadays, with slight adjustments, or with improvements over specific domains, or bringing completely new ideas, shifting the performance higher above its ancestors.

2.3.1 Classification and Detection

Image Classification. Some of the best preforming architectures for image classification are: VGG Net (2014) [21], with straightforward deep architecture being used as a basis of many other ConvNets today, GoogLeNet (2015) [22] introducing a non-sequentially stacked up layers, with very good performance and computational efficiency, and Microsoft ResNet (2015) [23], exceeding human performance in accuracy of image recognition by introducing an idea of *"Residual Blocks"*.

Object Detection. Three most known approaches to the object detection task based on CNN are:

1. R-CNN (Fast R-CNN, Faster R-CNN)
2. YOLO
3. SSD

R-CNN (Regions with CNN features) was the first object detection method using CNN. Training R-CNN consist of training a normal CNN model from classification, with additional one class referring to the background (no object of interest). Since CNNs were too slow and computationally very expensive, it was impossible to run on all patches generated by sliding window detector. *Selective Search* object proposal algorithm [24] was used to solve this issue reducing number of bounding boxes to close to 2000 region proposals. These regions are called *"Regions of Interest"* (RoI).

R-CNN feeds RoI patches to CNN, which compute CNN features, and uses SVM for the region classification. In final step, R-CNN runs a simple linear regression on the region proposal to generate tighter bounding box coordinates, to accurately get the final result.

Two years later (2015), Girshick introduced an improved version called *Fast R-CNN* [26], which uses a *"RoI Pooling layer"*. It shares the forward pass of a CNN for an image across its subregions, eliminating the main bottlenecks of R-CNN, in a way of sharing computational across the proposals. Fast R-CNN also combines the training of the CNN, classifier and bounding box regressor in a single model.

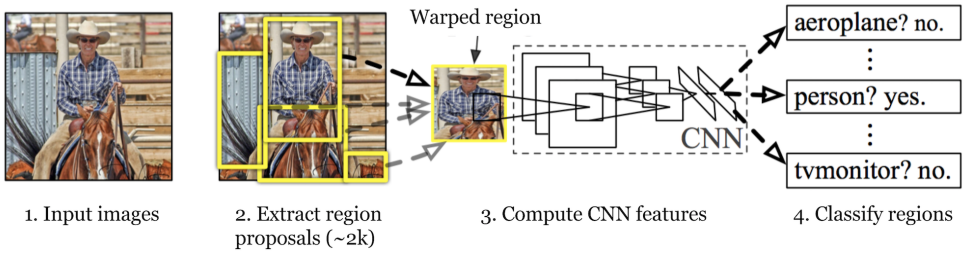


Figure 2.11: Regions with CNN features. Source: [25]

Because of using Selective Search for region proposals, fairly slowing down the whole process, this object detection method still struggled of being used in a real-time.

This was solved effectively with the *Faster R-CNN* architecture [27], introduced by research a team at Microsoft Research (Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun). The insight of Faster R-CNN was reusing already calculated CNN features to generate region proposals, instead of running a separate Selective Search algorithm. This is done via adding a Fully Convolutional Network on top of the features of the CNN, creating a so called *"Region Proposal Network"*.

Faster R-CNN is capable of running in real-time (using GPU) with a very good accuracy (Figure 2.12). [27]

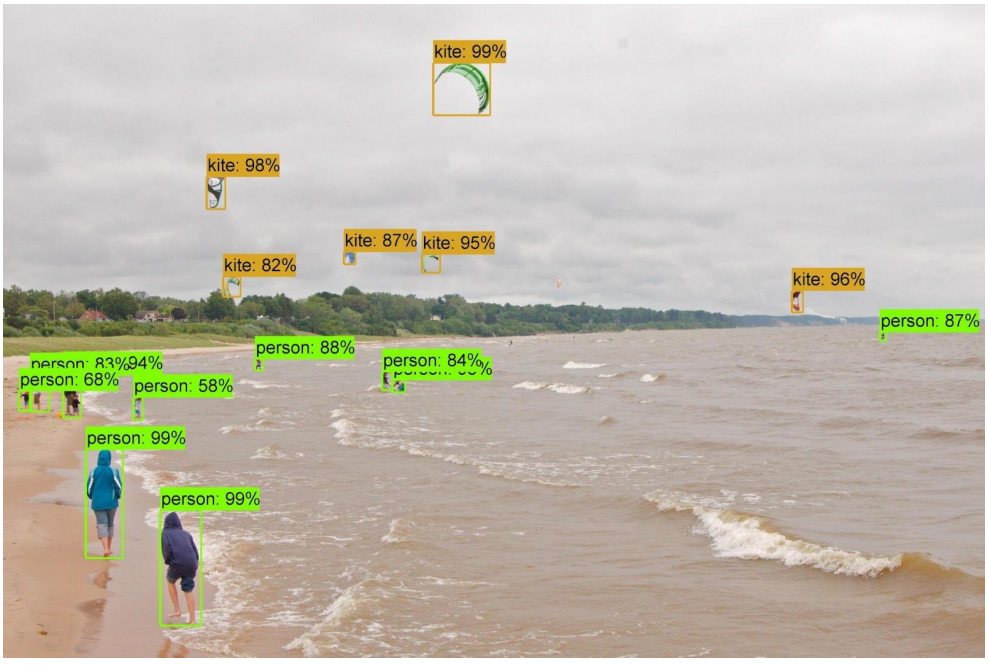


Figure 2.12: Faster R-CNN detection example. Source: [28]

2.3.2 Lip Reading

In lip reading there is a fundamental limitation on performance due to *homophemes*. These are sets of words that sound different, but involve identical movements of the speaker's lips. (e.g 'p', 'b' and 'm') [2]. Using the NNs to perform a prediction on whole sentences can distinguish between visually similar word candidates.

To address this issue Chung *et al.* introduced a new network model which they called *"Watch, Listen, Attend and Spell"* [29]. This model incorporates both audio and video training in a modules, and is then able to predict unseen sentences with lip reading only, or combined with audio to further helps with original sentence decoding. Decoding speed is only about 0.5 second for a 5-second sentence, which makes it significantly faster than real-time.

Another advance progress in a domain of lip reading brings an insight of paper *Lip2AudSpec: Speech reconstruction from silent lip movements video* [30] which aims to reconstruct the lip movements directly into spectrogram, which can be further used for a natural sounding reconstructed speed.

Chapter 3

Used Data and Tools

3.1 Python

Python[3] is an interpreted high-level programming language created by Guido Van Rossum, first released in 1991. It is designed to be easily readable and for general-purpose programming. Python has a huge set of libraries which can be used for machine learning and artificial intelligence programming (e.g. NumPy, SciPy, Scikit Learn, Tensorflow, Keras). Many of these libraries are using C or Fortran as a backend to preform heavy computation tasks very quickly.

3.2 Jupyter

Project Jupyter[4] is a tool which enable users to write their code into a so called Notebook, providing an interactive computing thanks to running code directly after executing a specified part of code (called Cell). Running a code in a Cell displays the output directly after that Cell. This feature is very useful for rapid prototyping applications, data science or reports as the Notebooks can be easily exported into normal a Python code or a PDF document.

3.3 OpenCV

OpenCV[5] (Open Source Computer Vision Library) is an open-source multi-platform library of programming functions mainly aimed at computer vision. It has C++, Python and Java interfaces and it was designed for computational efficiency.

[3] `https://www.python.org/`
[4] `http://jupyter.org/`
[5] `https://opencv.org/`

OpenCV contains many useful functions and algorithm implementations, which I am using in the following chapter (Chapter 5). Some of them are:

- Video/Camera stream input.
- Haar-cascade classifier.
- Object tracking.
- Drawing tools.
- Image cropping.

3.4 TensorFlow and Keras

TensorFlow. *TensorFlow* is an open source framework for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (GPUs, CPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. It was originally developed by researchers and engineers from the Google Brain team within Google's AI organization. TensorFlow framework provides strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains. TensorFlow is an excellent choice for creating Deep Neural Networks with endless possibilities for fine-tuning each individual component of the network. [31]

Keras. *Keras*, on the other hand, is a high-level Neural Network API. It is also open source. The main focus of this library is to bridge the gap between the low-level TensorFlow's computational functions and nice and easy user-friendly experience, with still taking advantage of fast environment for experiments. When prototyping or researching a Deep Neural Network it might be a tedious work to write and debug Neural Networks in pure TensorFlow framework and this is where exactly Keras could be a right choice. Keras backend can also be easily switched, and besides TensorFlow, CNTK[6] or Theano[7] can be also used. Since TensorFlow, CNTK and Theano all support both GPU and CPU training/predicting it is not a surprise that Keras supports them too. There exists two types of APIs for defining Neural Networks models in Keras:

- Sequential.
- Functional.

[6] `https://www.microsoft.com/en-us/cognitive-toolkit/`
[7] `http://deeplearning.net/software/theano/`

The *Sequential* model is a linear stack of layers and it is straightforward when defining a model architecture. The *Functional* API was designed to make it easier for defining more complex models, such as DAGs, models with shared layers, or models with multiple inputs and/or outputs. [32, 33]

For purpose of this thesis, I am using sequential architecture of model and therefore using Sequential API.

3.4.1 Keras example

Defining a Neural Network architecture with Keras is very easy. See Listing 1 as an example when defining a model.

```
# Import Sequential API and Layer definitions
from keras.models import Sequential
from keras.layers import Dense, Activation

# Build model passing layers to Sequential constructor directly
model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])

# Or by using add method
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

Listing 1: Keras model example. Taken from [33]

3.4.2 In-Browser Machine Learning

Keras.js is a JavaScript library enabling built and trained models with Keras to be run in browser. [34]

The main benefit of running the model in JavaScript at the client side is that the server does not have to compute all the tasks and eliminating or minimizing data transferring/streaming to server and thus respecting data privacy and improving security.

In modern web browsers users can benefit from GPU acceleration support provided by WebGL[8] and thanks to this speeding up the model prediction.

During writing this thesis Google released their own web-based framework called Tensorflow.js, which could be used not only for predicting outputs from already trained model, but also defining, training or transfer learning a Deep

[8] WebGL is graphic library for native graphics rendering on web

Neural Network directly in browser. In case of transfer learning for example re-train the model to match a users needs and improve the results by learning from new data *"on the fly"*. A nice example of building a ML web-based application is a Teachable Machine[9] by Google, enabling users to map their input to specified outputs using transfer learning methods (Figure 3.1). [31]

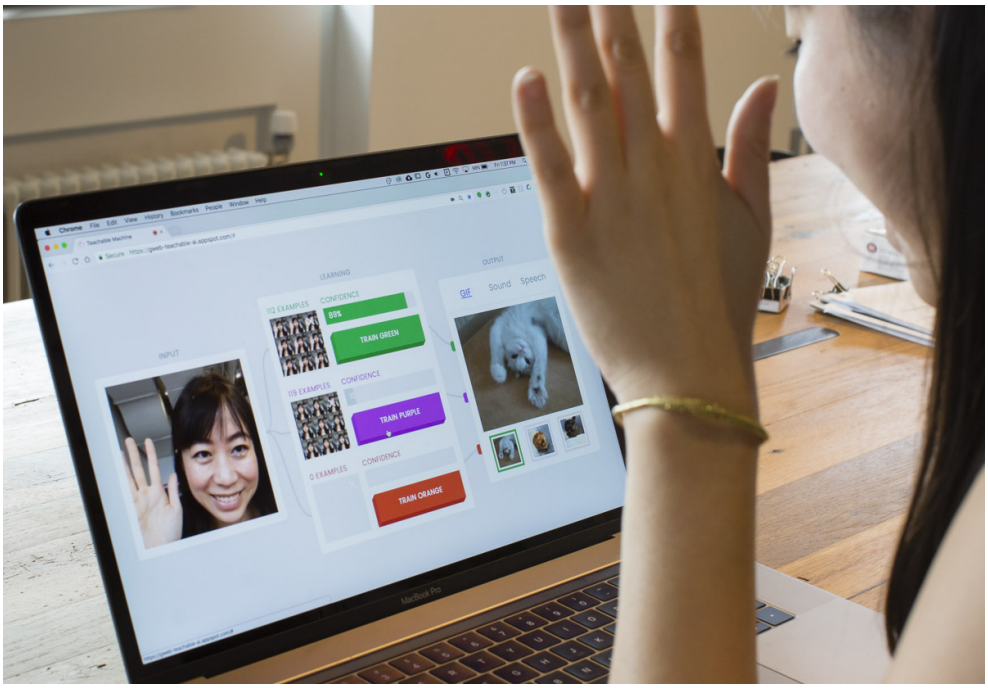


Figure 3.1: Teachable Machine demo. Source: [35]

3.4.3 LRW Dataset

The dataset I use, is, as was already mentioned in the introduction, the Lip Reading in Wild (LRW)[2] dataset (can be downloaded here[10]). This dataset consists of up to 1000 utterances of 500 different words, spoken by hundreds of different speakers, mainly a BBC reporters[11]. All videos are 29 frames (1.16 seconds) in length, and the word occurs in the middle of the video. The words are divided into 3 sets - *Train, Validation, Test*, as can be seen in the Table 3.1.

[9] `https://teachablemachine.withgoogle.com/`
[10] `http://www.robots.ox.ac.uk/~vgg/data/lip_reading/`
[11] British Broadcasting Corporation

Set	Number of classes	Samples per class
Train	500	800-1000
Validation	500	50
Test	500	50

Table 3.1: LRW speaker samples

The dataset can be used for non-commercial, academic research and before getting access a Data Sharing agreement with BBC Research & Development must be signed. Therefore I do not enclose the dataset. Also, images samples from the dataset used in this thesis were approved by BBC Research group.

Figure 3.2: LRW speaker samples. Source: [2]

Preview of the speakers can be seen in Figure 3.2 above. List of all words can be found in Appendix C.

Chapter 4

Design

In this chapter I present the anticipated procedures of data preprocessing, NN architectures, its training and evaluating, and methods of tracking and lip-reading the user when using the web application. These stages can be seen in Figure 4.1 bellow.

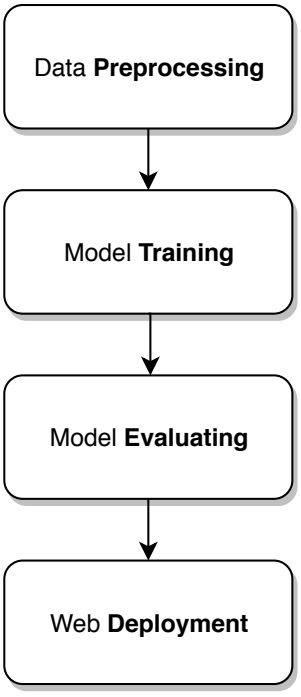


Figure 4.1: Developing stages

4.1 Preprocessing

In the preprocessing stage, the goal is to process the data from raw video to get only the region of interest - the mouth. Although feeding non preprocessed videos and letting the NN learn the features and the area of interest itself from a bigger frame might be also possible, it would require a far bigger dataset. Focusing solely on the mouth area will speed up the training.

Therefore I first process every video from the dataset, locating and cropping the mouth region and then tracking this area for the rest of the frames. These cropped frames are saved in the hard drive, so it is not necessary to preprocess the data before each training.

Because there is not much variation in color across the different frames, I save the frames as a grayscale image, reducing the size 3 times.

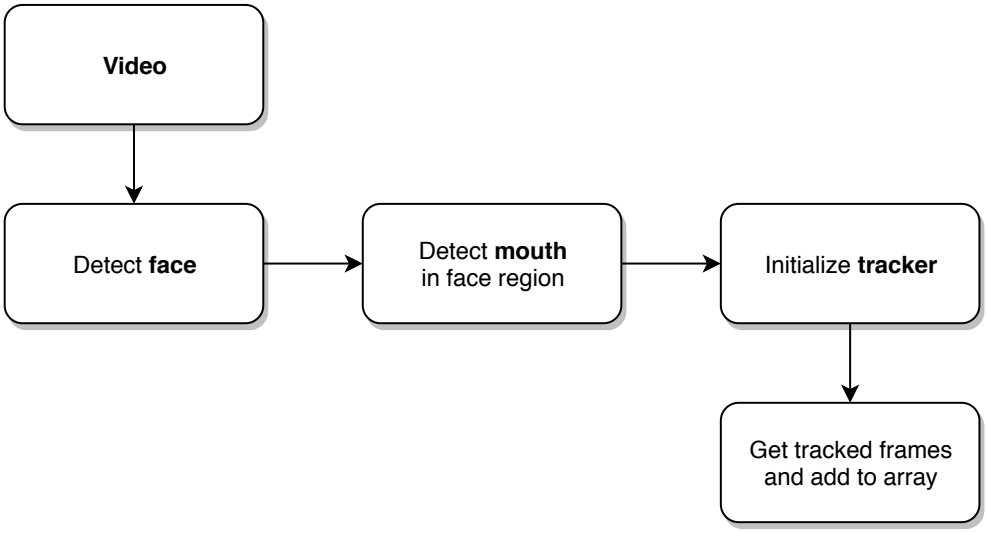


Figure 4.2: Lips detection process

4.2 Training

My goal is to train a good performing model, with a solid accuracy and fast enough to be able to lip read words said by an user on in the website application. Therefore in the implementation part (Chapter 5), I benchmark and compare the accuracy and speed of various model architectures, including the one proposed in [2], and the ones I build and tweak myself.

The model with the best accuracy is then exported and used on within the website application.

4.3 Evaluation

Evaluating the model accuracy is a key task in every ML area. It is necessary to find out if the model had learned patterns that generalize well for unseen data instead of just overfitting (as mentioned in Section 2.2.4) on the data it was shown during training.

There are many metrics of measuring the predictive accuracy of a model. For simplicity I use the *test set* of LRW dataset, and perform a so called *Top-1* accuracy which means that the word with highest probability must be exactly the expected answer.

Some researchers also use *Top-5* accuracy (e.g. in [6, 2]), which means that any of the model 5 highest probability answers must match the expected answer. [36]

4.4 Web Application

By publishing and embedding the trained model into a website application (web app) we can achieve a very new user experience and possibilities. The key idea is to allow the user to utilize his own graphics card, running the model completely on client side, and thus improving speed, minimizing server load and protecting user data.

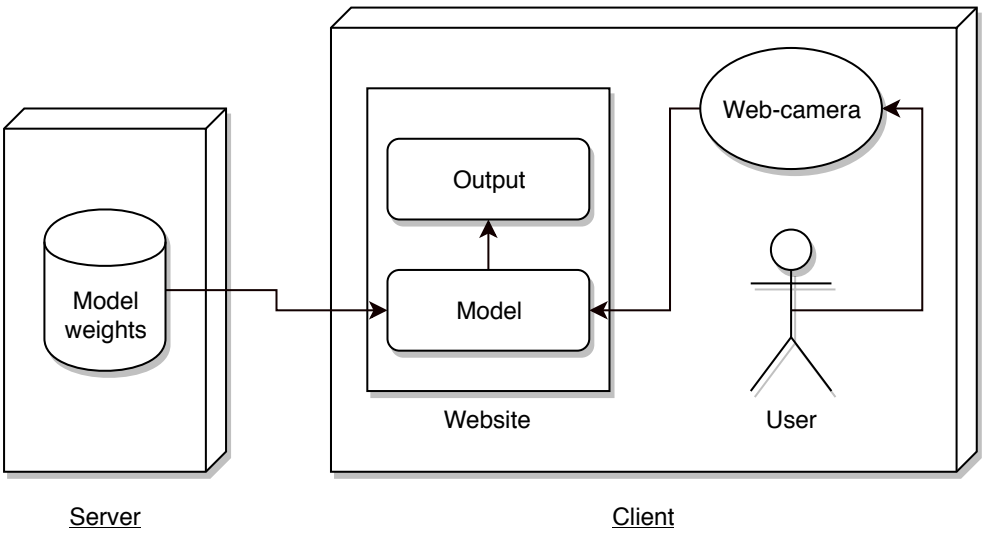


Figure 4.3: Web application design

The only exchange of data between user and server is when accessing the website. The client downloads the website itself, model architecture and trained weights. From this point no more data needs to be send or reviewed from the server. Illustration of this process can be seen in Figure 4.3.

The website application use the user's web-camera in a similar way as when preprocessing the data for training. On each frame the head and mouth is detected and tracked and the cropped area of lips are being held in an array.

In Figure 4.4, I outline various states of the website application for recording and predicting a lip sample.

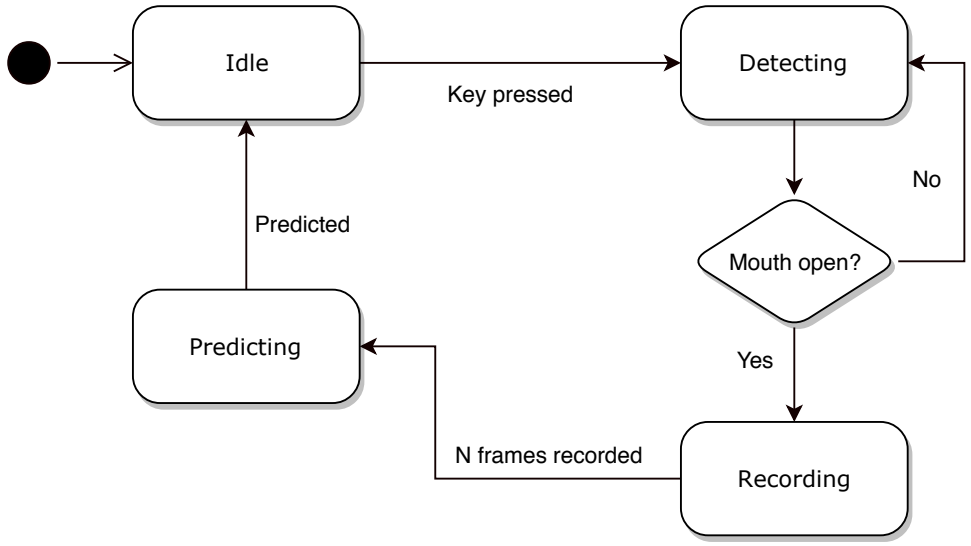


Figure 4.4: Web application state diagram

The process begins in `Idle` state, and after a user hits a given key the process moves to `Detecting` state, where it monitors whether the user opened his mouth, and if so, moves to `Recording` state. From this point it counts a specific number of frames, which are then passed to the model for a prediction (`Predicting` state). After the model predicted the word, process returns back to `Idle` state, and the whole process can start again.

Chapter 5

Implementation

In this chapter I go through training process and its details using Keras library, compare various architectures and impact of their parameters. Next I show a deployment process of the best performing to website application, and describe the application creation details.

5.1 Preprocessing

For the preprocessing part I use primarily Jupyter Notebook with Python 3.6 kernel. The benefits of using the Notebooks are a convenient workflow and easily exportable and reproducible outputs. The final Notebooks or their exported HTML version can be viewed on the enclosed CD medium (Appendix A).

5.1.1 Word Selection

Because of my very limited computation resources (section 5.2.2), and considering ease of usage of the web application, I did not strive to train the entire LRW dataset, which would take days or weeks to train on my GPU. Instead I focused to try different architectures and tuning methods, with a goal to improve the current accuracy and speed on a smaller subset of words.

I chose randomly 30 words, however, when testing the web application I found out that the number was still overwhelming and the words hardly fitted into the result bar plot.

Therefore I manually removed 10 words, preferably the ones that had very similar co-articulation with others or were hardly pronounceable for non-native speakers, and I was left with the following 20 words:

about, absolutely, according, afternoon, already, british, economic, george, history, hospital, information, london, manchester, missing, office, question, remember, sunshine, warning, without

5.1.2 Detection and Tracking

Detection. I run a *Haar classifier* (described in Section 2.1) on 1st frame of every sample during preprocessing. All used pretrained Haar classifiers can be found here[12].

- Frontal face by R. Lienhart.
- Profile face by D. Bradley.
- Mouth by M. C. Santana [37].

First the *frontal face* classifier is used to detect a face in the first frame, if it fails to find a face a *portrait face* classifier is used in the same manner. In the detected face region another Haar classifier is run - this time the *mouth* classifier. If even portrait face or mouth classifier fails to detect a face, the sample is skipped. Otherwise the tracker is initialized on the detected mouth area, which is slightly enlarged for better tracking (more features to track) and to prevent accidentally cropping a part of the mouth in case of fast movement. Detection can be seen in the Figure 5.1 bellow.

```
# Load HAAR classifiers
face_frontal_cascade = cv2.CascadeClassifier('frontalface.xml')
face_profile_cascade = cv2.CascadeClassifier('profileface.xml')
mouth_cascade = cv2.CascadeClassifier('mouth.xml')

# Find frontal faces in the grayscale image
faces = face_frontal_cascade.detectMultiScale(
            gray, scaleFactor=1.2, minNeighbors=3, minSize=(100, 100))
if len(faces) == 0:
    # Frontal face not found, try profile face
    faces = face_profile_cascade.detectMultiScale(
                gray, scaleFactor=1.2, minNeighbors=3, minSize=(100, 100))

...

if len(faces) > 0:
    # Frontal or profile face found, try detecting mouth
    mouths = mouth_cascade.detectMultiScale(
                roi_lower_face, scaleFactor=1.3, minSize=(15, 15))
```

Listing 2: Face and mouth detection process

In Listing 2, the method `detectMultiScale` called on a Haar cascade object detects objects of different sizes in the input image. Its additional parameters that I use are:

[12] `http://alereimondo.no-ip.org/OpenCV/34`

- `scaleFactor` - parameter specifying how much the image size is reduced at each image scale.
- `minNeighbors` - parameter specifying how many neighbors each candidate rectangle should have to retain it.
- `minSize` - minimum possible object size. Objects smaller than that are ignored.

Good values for these parameters I found via a *"trial and error"* method, when trying the different detection parameters on various samples.

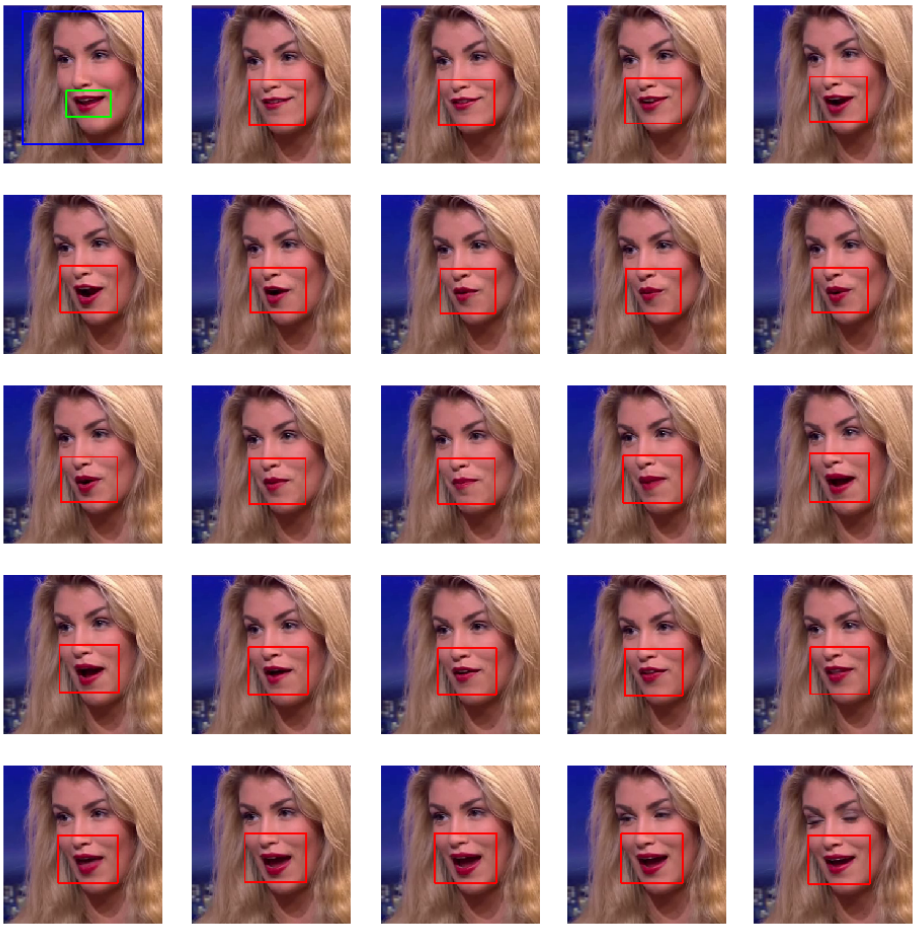


Figure 5.1: Lip detection and tracking (ordered by rows). In the first image, blue rectangle is a detected face area, green is a mouth area, where a tracker is initialized. In the following frames, the red rectangle is the tracked mouth area. Base image: LRW dataset, word "ABOUT" [2]

Tracking. There are many methods of tracking a specified object across multiple subsequent frames, and describing the functioning of each considerably exceeds the scope of this work. *OpenCV* library [38] offers multiple tracking algorithms:

- KCF Tracker,
- MIL Tracker,
- MedianFlow Tracker,

and many more. Each tracking algorithms has its own strengths and weaknesses. Very good comparison and description is available at [39].

I found out that these three algorithms listed above are robust enough for lip tracking, however, a test of processing 100 video samples ended up by losing tracker on 39 samples when using *KCF*, 17 when using *MedianFlow* and 13 when using *MIL*. By taking into consideration the average speed of MedianFlow (65 ms / sample) and MIL (2559 ms / sample), I decided to use MedianFlow tracker. The result can be seen again in the Figure 5.1.

```
# Create MedianFlow tracker to track detected mouth on upcoming frames
tracker = cv2.TrackerMedianFlow_create()

...

# Initialize tracker with first frame and bounding box
tracker.init(frame, detected_mouth_bbox)

...

ok, bbox = tracker.update(next_frame)
if ok:
    # Tracker is successfully matched in following frame
    # TODO: crop and resize the tracked area (bbox)
else:
    # Tracker is lost (tracked area changed significantly)
    # TODO: skip this sample / try again with different tracker
```

Listing 3: Tracking mouth region across frames

In the Listing 3 above, I show the usage of tracker from *OpenCV* library [38] on simplified example from my code. First a MedianFlow `tracker` object is created by calling the `TrackerMedianFlow_create()` function. Then I begin tracking the desired area by calling `init` method with parameters `image` and `bbox` (area described with tuple of x, y, w, h). In the next frame, I just call the `update`, to return the new `bbox` and state `ok` of boolean values, where `True` means tracker found the tracked area, and `False` means the tracker is lost.

5.1.3 Data Storing and Retrieving

Saving. Because preprocessing the data (extracting the mouth region) of the whole dataset take some time, it is not wise to run over the whole process again, when performing lot of testings and adjustments of training. Instead, saving the preprocessed data can save a lot of time. When extracting the mouth regions via the tracking method described above, I convert to grayscale and resize each frame to dimensions 24×32 pixels, which in total of 28 frames produces a tensor of size $28 \times 24 \times 32$, with depth, height and width respectively. The values are populated in a `NumPy 3D array` and stored compressed on a hard disk. The saved array of one sample can be seen in Figure 5.2[13].

Figure 5.2: Saved NumPy array sample

Batch loading. When loading data the data into the Neural Network, multiple approaches can be used. In the beginning, when I was training just on few samples, I had loaded all the training and validation data into the RAM memory. However, this works only up to the point where the size of data do not exceed the total RAM and GPU memory capacity. Luckily a better approach called *batch loading* exists. Idea behind this is to load a smaller batch of data (typically 16, 32, 64, etc.), train the network on this smaller chunk, and then load other data. This approach only uses the required amount of memory to process the current batch at a given time. Furthermore, an augmentation step can be added when loading the data (more on this in following section 5.2).

[13]Some figures in this thesis contains 25 frames instead of 28 (the very first frame is not saved). Initially, I was experimenting with both sizes, but left the illustrations of size 25 to better fit the grid.

Keras has an in-built support for batch loading using Python `generators`. `Generator` can be tough as a function which *yields* a data continuously upon each request. When working with images the Keras provides already implemented `ImageDataGenerator` class[14], with optional augmenting options, but it is not useful for higher dimensional data. In the Listing 4, I show a simplified implementation of `generator` which loads and yields the lip samples (3D arrays).

```
def lip_sample_generator(basedir, set_type, batch_size, nb_classes):
    """Replaces Keras' native ImageDataGenerator."""

    # Directory from which to load samples
    directory = os.path.join(basedir, set_type)

    # Placeholder NumPy arrays for features and labels
    batch_features = np.zeros((batch_size, DEPTH, ROWS, COLS, 1))
    batch_labels = np.zeros((batch_size, nb_classes), dtype="uint8")

    file_list = ... # Populate with file paths

    while True:
        for b in range(batch_size):

            # Each time choose random sample from dataset
            i = np.random.choice(len(file_list), 1)[0]

            # Load the sample
            sample = np.load(file_list[i])

            # ... Normalization and augmentation

            # ... One hot encoding

            # Place sample and its label to the NumPy array
            batch_features[b] = sample
            batch_labels[b] = encoded_label

        yield (batch_features, batch_labels)
```

Listing 4: Batch generator of lip samples

[14] https://keras.io/preprocessing/image/

5.2 Training

In this section I describe the NN architectures used for training the lip samples, compare different parameters, and present achieved results.

5.2.1 Base Training parameters

If not specified otherwise, I perform the training with the following parameters:

Number of epochs - 30 or end if validation accuracy does not improve after 4 consequent epochs.

Optimizer - Adam[15].

Learning rate - 1×10^{-4}.

Other parameters that I tune or change, I specifically name for each model listed in Benchmarks section (Section 5.2.4).

5.2.2 Used Hardware for training

For training I used my own desktop computer with the following specification:

CPU - Intel(R) Core(TM) i5 CPU 750 2.67Ghz,

RAM - 16,0 GB,

GPU - NVIDIA GeForce GTX 1050 Ti (4 GB VRAM),

OS - Windows 10 Home.

5.2.3 Base model architecture

When training the Neural Network I have used various architectures. First I have built a model as is suggested in LRW [2, p. 8], the EF-3 architecture (see Table 5.1), which is based on VGG-M model [40].

[15] `https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/`

Layer type	Output Shape	Param #
Conv3D	(None, 28, 24, 32, 48)	1344
Batch Normalization	(None, 28, 24, 32, 48)	192
MaxPooling3D	(None, 9, 8, 10, 48)	0
Conv3D	(None, 9, 8, 10, 256)	332032
MaxPooling3D	(None, 3, 2, 3, 256)	0
Batch Normalization	(None, 3, 2, 3, 256)	1024
Conv3D	(None, 3, 2, 3, 512)	3539456
Conv3D	(None, 3, 2, 3, 512)	7078400
Batch Normalization	(None, 3, 2, 3, 512)	2048
Flatten	(None, 9216)	0
Dense	(None, 20)	184340
Total params: 11,138,836		
Trainable params: 11,137,204		
Non-trainable params: 1,632		

Table 5.1: EF-3 model architecture

The training of the the EF-3 architecture ended after 16 epochs, as validation accuracy did not improved for 4 consecutive epochs. The resulting *test accuracy* is 80.29%.

I was also experimenting with more lightweight model architectures, and found another well-performing solutions, which trains much faster (due to approximately 10 times less trainable parameters), while still maintaining or even exceeding the final *test accuracy*.

As a base architecture I used the one described by the Table 5.2 bellow, on which I tried other architecture differentiation and parameter tuning.

Layer type	Output Shape	Param #
Conv3D	(None, 28, 24, 32, 32)	4032
MaxPooling3D	(None, 9, 8, 10, 32)	0
Conv3D	(None, 9, 8, 10, 64)	256064
Conv3D	(None, 9, 8, 10, 64)	512064
MaxPooling3D	(None, 3, 2, 3, 64)	0
Flatten	(None, 1152)	0
Dense	(None, 128)	147584
Dense	(None, 20)	2580
Total params: 922,324		
Trainable params: 922,324		
Non-trainable params: 0		

Table 5.2: Lightweight model architecture

5.2.4 Benchmarks

I ran a benchmarks of different architecture tuning to find out the impact of various normalization and generalization methods, described in Section 2.2.6 and section 2.2.5. For the benchmarks I trained the following models listed in Table 5.3, where each model builds on top of the base architecture described by Table 5.2 and adds a small modification to it.

Model	Batch Normalization	Dropout	Kernel size
A1	-	-	$3 \times 3 \times 3$
A2	-	-	$5 \times 5 \times 5$
B	Yes	-	$5 \times 5 \times 5$
C	Yes	20%	$5 \times 5 \times 5$
D	Yes	40%	$5 \times 5 \times 5$

Table 5.3: Model architectures

5.2.5 Kernel size

When comparing the results of model **A1** and **A2**, I found out that enlarging the kernel size of 3D convolutions from $(3, 3, 3)$ to $(5, 5, 5)$ greatly improves the final accuracy - from 75.84% to 80.85%. The training process can be seen in the Figure 5.3. This is probably thanks to better registering the features in the 3D volume in this smaller architecture, and the bigger kernel might be less prone to noise in the image data. Therefore I used the kernel of shape $(5, 5, 5)$ with the training of rest of the benchmark models (**B**, **C**, and **D**).

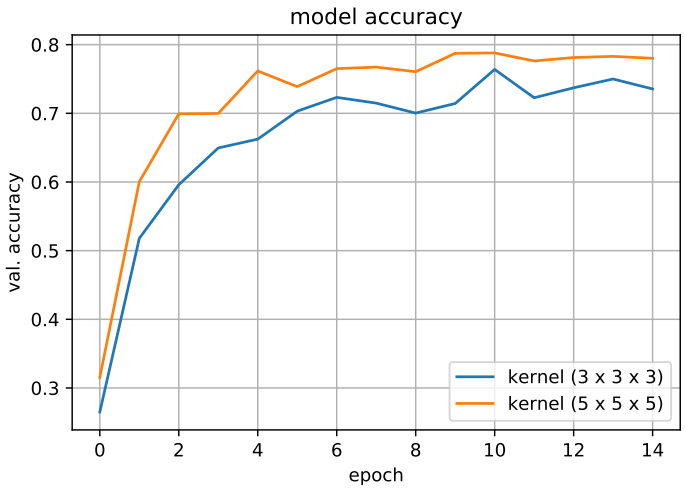


Figure 5.3: Kernel size comparison

5.2.6 Normalization

Before feeding the data into the each network I scale the data to be between -1 and 1, thus converting the `uint8 NumPy array` of values $[0; 255]$ to type `float16` of range $[-1; 1]$. Simple code to achieve this can be seen in Listing 5 bellow.

```
# Normalize to [-1; 1] and convert to type "float16"
sample = (sample.astype("float16") - 128) / 128
```

Listing 5: Sample normalization

I compare the two architectures, **A2** and **B**, where **A2** is an architecture without Batch Normalization (BN), and **B** uses BN layers. The *test accuracy* of **A2** is 80.85%, whereas **B** is 81.85%. Which shows small, yet appreciable improvement. Batch Normalization can, however, have more significant impact while training much bigger and/or multivariate data.

5.2.7 Dropout

As can be seen in Figure 5.4, usage of a Dropout technique has a significant impact on the training. The model without a Dropout regularization (**B**) has a much steeper learning curve, resulting in earlier convergence and the loss stops to improve. On the opposite, **D** with a 40% Dropout achieved much better validation loss, thanks to better regularization. However, using even higher Dropout rate can lead to the model not being able to train at all.

5.2.8 Augmentation

I was experimenting with augmenting the samples using imgaug library [19]. Basic idea to augment a series of images is to remember parameters of applied operation (transformation), and then use it the same way on the rest of frames. However, imgaug library is build to be used for augmenting images, not 3D arrays, which my sample representation were stored in, and iterating over each slice of this array, picking one frame, applying the transformation and putting it back in place posed significant computation overhead. As the feeding of data was done simultaneously, this drastically impacted the time of training via GPU as it was no longer usable.

Solution to this would be to first augment the data, save it onto hard drive and then train the model from this new data. This requires several times more space on hard drive. Another approach is to utilize parallel computation and augment the data in batches, at the same time as training the model. I have not tried these two methods and instead focused on to other methods of tuning the model.

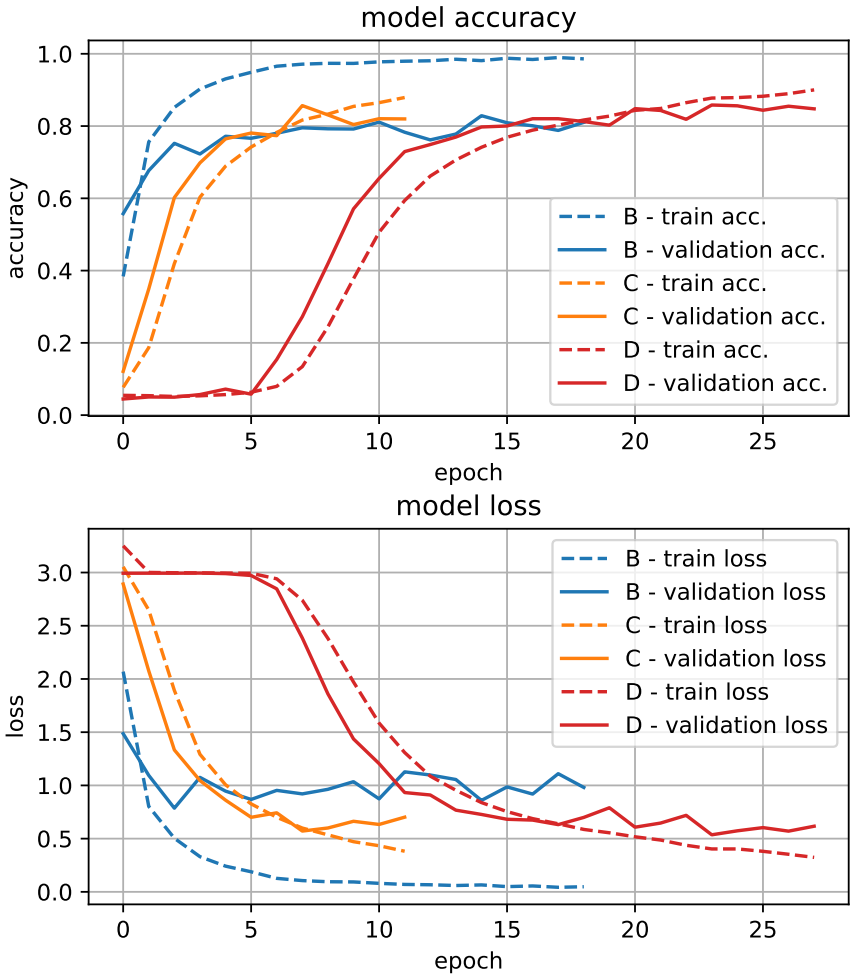


Figure 5.4: Dropout rate comparison

5.3 Evaluation

After training each of the chosen model, the testing accuracy was measured on a *test set* containing 898 samples.

Model	BN	Dropout	Kernel size	# Epochs	Test acc.	Speed[16]
EF-3	-	-	$3 \times 3 \times 3$	16	80.29%	195 ms
A1	-	-	$3 \times 3 \times 3$	14	75.84%	112 ms
A2	-	-	$5 \times 5 \times 5$	14	80.85%	119 ms
B	Yes	-	$5 \times 5 \times 5$	18	81.85%	123 ms
C	Yes	20%	$5 \times 5 \times 5$	11	84.08%	125 ms
D	Yes	40%	$5 \times 5 \times 5$	27	88.31%	119 ms

Table 5.4: Model architectures evaluation

5.3.1 Confusion Matrix

Confusion Matrix (CM) is tool for summarizing the performance of a classification algorithm. It can give a better idea of what classes the model predicts wrongly the most and for which words they are being confused. I generated a confusion matrix for a model with best accuracy - **D2**, and the plotted CM can be seen in Figure 5.5.

All values are normalized and the number in each individual cell represents how much is the given word being confused with another one (in percentage). The darker background illustrates a higher confusion rate.

[16] Average time to predict one standalone sample using my NVIDIA GTX 1050 Ti graphics card. Using GPU can be much more effecting when predicting multiple samples at a time, by utilizing parallel computation and reducing overhead.

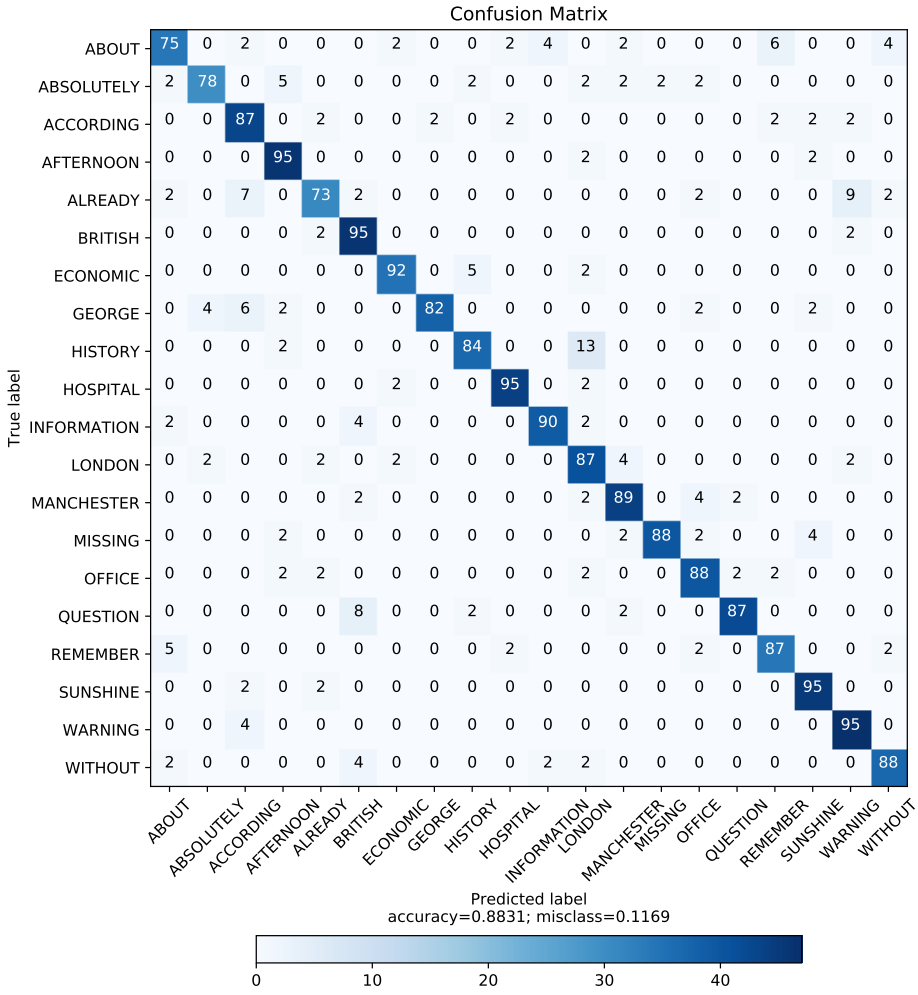


Figure 5.5: Confusion matrix of trained words. Implemented using scikit-learn machine learning library [41]

5.4 Web Application

The web application (web app) is one apart from preprocessing of data and training the models key part of this thesis. As was already mentioned in the Design chapter (Chapter 4), it should be a demonstration of the lip-reading performance of the trained model, by enabling user to see predicted result of a single spoken (lip-articulated) word into the web-camera.

To achieve a near real-time performance I build the web app using a JavaScript and libraries running on client side.

5.4.1 Used Libraries

For easy document traversal and manipulation I use *jQuery*[17] library, for rendering debug and result plots I use *Chart.js*[18] which can be used to draw beautiful, animated and interactive charts.

When building the web app, *TensorFlow.js* had not been realeased yet, and because of training the models in Keras library, I chose to use *Keras.js* [34] library for easy model deployment.

Finally, for face/mouth detection and tracing I found a perfectly fitting, well-performing library called *clmtrackr* [42], which is in fact a JavaScript implementation of *"Deformable Model Fitting by Regularized Landmark Mean-Shift"* [43]. It can detect and track a face and also guess and map 70 face feature points (face coordinates) on a face, which I use for tracking a mouth area. All feature points, their location and identification number can be seen in Figure 5.6.

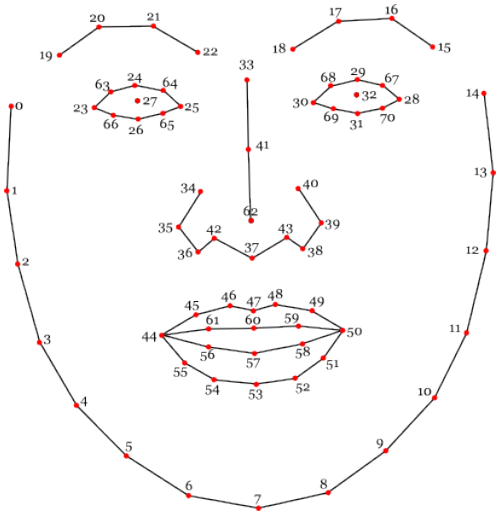


Figure 5.6: Face coordinates from clmtrackr library. Source: [42]

[17] `https://jquery.com/`
[18] `http://www.chartjs.org/`

5.4.2 Model deployment

Keras.js uses a custom protocol buffer format binary file for the Keras model weights file. Before using the weights with the model in web app, a conversion of original weights file has to be made. This could be done easily with Keras.js provided `encoder.py` script by running it with a path of model weight as an argument (Listing 6).

```
python encoder.py "/path/to/model_weights.h5"
```

Listing 6: Keras.js weights encoding

5.4.3 Tracking and Recording

Thanks to *clmtrackr* library, detection and tracking in `video` element is very easy. First, a `tracker` object is inicialized, and then attached to `video` element by passing it as an argument of `start` method.

Clmtrackr then tries to find a face, and if it is successful, it then maps face coordinates to fit the face as closely as possible. This coordinates can be retrieved by calling `getCurrentPosition()` method and then individually checked for obtaining a single coordinate position. I use this to figure out the position of mouth and its width and height. I then expand this area and store it into an array. The process of using clmtrackr in my web application can be seen in Listing 7

```
// Initialize clmtrackr
var ctrack = new clm.tracker();
ctrack.init();

...

// Start tracking
ctrack.start(vid);

... // Loop to obtain coordinates on every frame

var pos = ctrack.getCurrentPosition();

var mouth_left_top_corner = [pos[44][0], pos[46][1]];
var mouth_right_top_corner = [pos[50][0], pos[46][1]];
var mouth_width = mouth_right_top_corner[0] - mouth_left_top_corner[0];
var mouth_height = pos[53][1] - pos[47][1];
```

Listing 7: Clmtracker tracking and detection

As can be seen in state diagram (Figure 4.4) in previous chapter, the *recording* stage (acquiring of the lip frames) has two *"switches"*. First, hitting a *spacebar* enables the second, *mouth openness* switch, which when activated starts to count acquired frames to fill the whole sample for prediction.

I use the first one as a prevention against unwanted activation of recording when a user is not yet ready. The mouth openness switch has to be very sensitive to begin recording as soon as the user open his mouth to pronounce a word, because delaying this action may result results to shifted sample and prediction may not be correct.

Originally, I used clmtrackr coordinates in the middle of upper and lower lip (60 and 57) and measured their relative distance, however, for some words with smaller lips movements in the beginning, this approach did not work as the coordinates sometimes did not make a significant move to activate the threshold.

To solve this issue I take another approach and monitor the change of brightness in the center of mouth and save it into an array. If a new value surpasses the *standard deviation* of past values significantly, the *recording* state is activated. Plotting of the standard deviation can be seen in Figure 5.7.

Figure 5.7: Mouth openness detection chart

5.4.4 Prediction

The resulting array of lip frames is, after capturing 28 frames, *flattened*[19], normalized in a same way as when training, and the resulting vector is given to the model for prediction. When the model is done predicting, it outputs the vector of probabilities for each word, which are then displayed in a chart for easy review. The prediction function can be see in Listing 8.

5.4.5 Issues and Debugging

During the implementation I encountered many problems, and debugging any ML applications is not an easy task, because we can not directly see what happens inside the model, when the model is not working as it should.

To better debug the web app I use two visual aids. One is a plotting the last 25 frames under the previewing window, to better understand what frames are being fed into the model, and second is a graph of *mouth openness*

[19]Since 1 lip frame is a 2D array, and the model accepts a 1D vector (array), all frames are joined by lines into a one array.

```
async function predict(arr) {

    // Convert input array to Float32Array
    const inputData = {
        input: new Float32Array(arr)
    }
    // Perform a prediction
    var outputData = await model.predict(inputData);

    // Convert Float32Array into a normal array of Integers
    var outputDataArr = Array.prototype.slice.call(outputData["output"]);
    var outputDataArrInt = [];
    for(var i = 0; i < outputDataArr.length; i++) {
        var whole =  parseFloat(outputDataArr[i]);
        outputDataArrInt.push(Math.round(whole * 100));
    }
    // Plot the resulting probabilities on the chart
    updateChart(myChart, outputDataArrInt);
}
```

Listing 8: Keras.js prediction

mentioned in previous subsection. This chart can be shown when clicking the *cog* icon in the top-left corner of the web app (as seen in Figure 5.7).

To name some of the problems: One, very hard to sort out was, that not all web-cameras behaves the same, and there is no way how to force to run at the united and desired FPS on the web (through Media Capture and Streams API[20]). To address this problem web app grabs a frame in fixed interval which ensures the equally long samples ($\approx$ 1 second) by downsampling the frames of faster running web-cameras and interpolating the ones from web-cameras running bellow a desired frame rate (25-28 FPS).

Another issue is caused by having the model trained on words 1 second long, cropped from continuous speech, where the word itself occurs in middle of the video. Now when predicting model would have worse accuracy if the sample starts right when a mouth opens as the whole sample would be shifted to the beginning. So when a *mouth opening* occurs I take into account 6-7 past frames (thanks for buffering all the frames) and the rest till the full capacity of sample (28 frames). Some longer, slowly spoken words could, however, not be fully pronounced whole in time before beginning of prediction due to this approach.

This could be also (better) solved by training the model of larger dataset and augmenting it by shifting and scaling a time domain, thus obtaining a better robustness and regularization.

[20] https://developer.mozilla.org/en-US/docs/Web/API/Media_Streams_API

5.4.6 Results on Web

Although I did not focus on any profiling or optimization for improving time efficiency, apart from designing lightweight model architecture (Section 5.2) the prediction runs very fast even in web environment. One sample prediction lasts about 500 - 700 milliseconds to output using my GPU and would be even faster using a better one. The prediction accuracy is also very good, although some words have to be lip-articulated precisely, or quickly in a case of long words (to fit in a 1 second recording interval).

The final web application appearance can be seen in Figure 5.8 bellow.

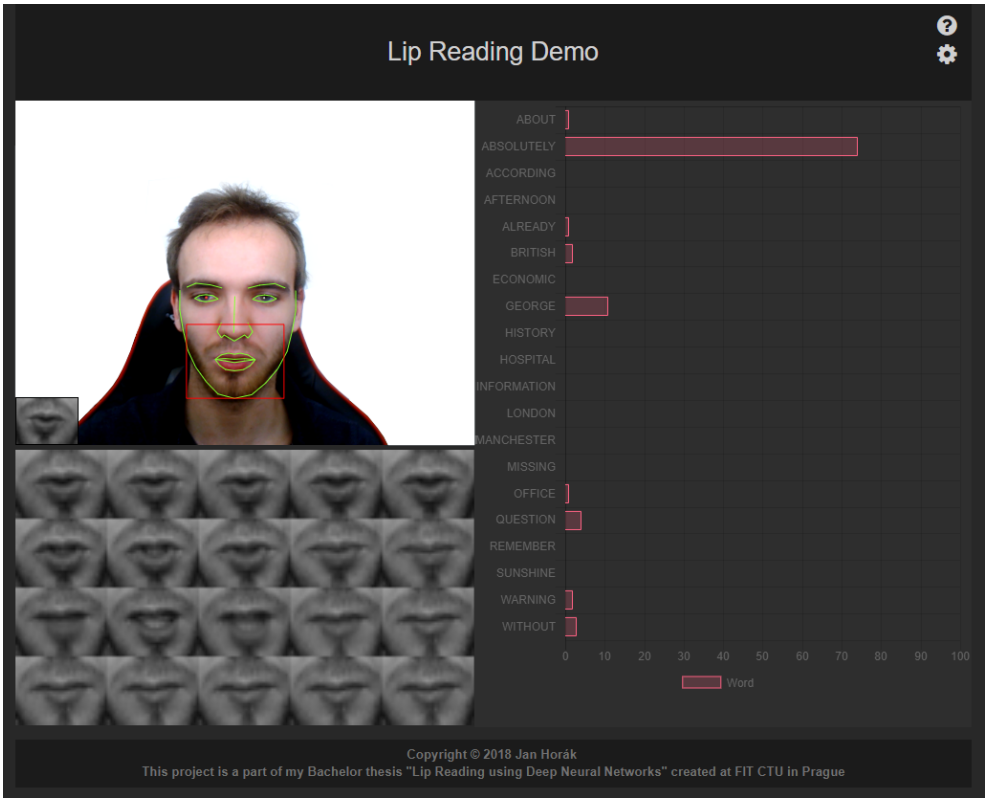


Figure 5.8: Lip reading web application. **Top**: Title with buttons to show help screen and to show mouth openness chart. **Top left**: Video element with clmtracker mapped on face. **Bottom left**: Preview of past 25 lip frames for easier debugging. **Right**: Bar chart of predicted result words. **Bottom**: Footer with reference to this thesis. Online version of the web application can be found at: `www.janhorak.info/lip`

Conclusion

The main goal of this thesis was to create a demo website application, enabling visitors to try a lip reading in browser on their own computer.

During implementation, I familiarized with various technologies and new tools, such as Object Detection techniques, Convolutional Neural Networks and Keras.js framework enabling a trained NN model to run on client side using JavaScript.

In the first section I surveyed the most used Image Classification and Object Detection methods as well as the current State of the Art in a field of Automated Lip Reading.

The result of this Bachelor's thesis is a Jupyter Notebook containing a script for preprocessing the LRW dataset and training Deep Neural Networks of various architectures and parameters, achieving very good accuracy of classifying lip-articulated words, both with the testing data and in the demo website application, which design and implementation is a next output of this thesis.

This created website application can be easily extended in order to be used for other similar purposes, or further improved by supplying even a higher accurate model or a model capable of lip-reading whole sentences.

Bibliography

[1] Hassanat, A. B. A. Visual Speech Recognition. *CoRR*, volume abs/1409.1411, 2014, 1409.1411. Available from: http://arxiv.org/abs/1409.1411

[2] Chung, J. S.; Zisserman, A. Lip Reading in the Wild. In *Asian Conference on Computer Vision*, 2016.

[3] Ouaknine, A. Review of Deep Learning Algorithms for Object Detection. *Medium [online]*, 2018. Available from: https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852

[4] Bosch, A.; Zisserman, A.; et al. Image Classification using Random Forests and Ferns. In *2007 IEEE 11th International Conference on Computer Vision*, Oct 2007, ISSN 1550-5499, pp. 1–8, doi:10.1109/ICCV.2007.4409066.

[5] Zheng, Y.; Meng, Y.; et al. Object recognition using a bio-inspired neuron model with bottom-up and top-down pathways. *Neurocomputing*, volume 74, 2011: pp. 3158–3169.

[6] Krizhevsky, A.; Sutskever, I.; et al. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[7] Szegedy, C.; Liu, W.; et al. Going Deeper with Convolutions. *CoRR*, volume abs/1409.4842, 2014, 1409.4842. Available from: http://arxiv.org/abs/1409.4842

[8] Viola, P.; Jones, M. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR*

2001, volume 1, 2001, ISSN 1063-6919, pp. I–511–I–518 vol.1, doi: 10.1109/CVPR.2001.990517.

[9] OpenCV. *Face Detection using Haar Cascades*. 2018, [cit. 04-17-2018]. Available from: https://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html

[10] Hornik, K.; Stinchcombe, M.; et al. Multilayer Feedforward Networks Are Universal Approximators. *Neural Netw.*, volume 2, no. 5, July 1989: pp. 359–366, ISSN 0893-6080, doi:10.1016/0893-6080(89)90020-8. Available from: http://dx.doi.org/10.1016/0893-6080(89)90020-8

[11] Stanford University. Convolutional Neural Networks. *CS231n Convolutional Neural Networks for Visual Recognition [online]*, 2017. Available from: http://cs231n.github.io/convolutional-networks/

[12] Trujillo J., A. Summarization of video from Feature Extraction Method using Image Processing and Artificial Intelligence. 01 2018.

[13] Deshpande, A. A Beginner's Guide To Understanding Convolutional Neural Networks. *Personal Github blog [online]*, 2016. Available from: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

[14] Veličković, P. Deep learning for complete beginners: convolutional neural networks with keras. *Cambridge Spark [online]*, 2016. Available from: https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html

[15] runhani (https://stackoverflow.com/users/6730309/runhani). What do you mean by 1D, 2D and 3D Convolutions in CNN? *Stackoverflow [online]*, 2017, [cit. 30-04-2018]. Available from: https://stackoverflow.com/a/44628011

[16] Pedregosa, F.; Varoquaux, G.; et al. Underfitting vs. Overfitting. *Scikit-learn documentation [online]*, 2017, [cit. 28-04-2018]. Available from: http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

[17] Koehrsen, W. Overfitting vs. Underfitting: A Conceptual Explanation. *Towards Data Science [online]*, 2018. Available from: https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852

[18] van Laarhoven, T. L2 Regularization versus Batch and Weight Normalization. *CoRR*, volume abs/1706.05350, 2017, 1706.05350. Available from: http://arxiv.org/abs/1706.05350

[19] Jung, A. *imgaug*. 2016. Available from: https://github.com/aleju/imgaug

[20] Srivastava, N.; Hinton, G.; et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, volume 15, 2014: pp. 1929–1958. Available from: http://jmlr.org/papers/v15/srivastava14a.html

[21] Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, volume abs/1409.1556, 2014, 1409.1556. Available from: http://arxiv.org/abs/1409.1556

[22] Szegedy, C.; Liu, W.; et al. Going Deeper with Convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. Available from: http://arxiv.org/abs/1409.4842

[23] He, K.; Zhang, X.; et al. Deep Residual Learning for Image Recognition. *CoRR*, volume abs/1512.03385, 2015, 1512.03385. Available from: http://arxiv.org/abs/1512.03385

[24] Uijlings, J.; van de Sande, K. E. A.; et al. Segmentation as Selective Search for Object Recognition. In *ICCV*, 2011, doi: 10.1109/ICCV.2011.6126456. Available from: http://www.huppelen.nl/publications/ICCV2011SelectiveSearch.pdf

[25] Girshick, R. B.; Donahue, J.; et al. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, volume abs/1311.2524, 2013, 1311.2524. Available from: http://arxiv.org/abs/1311.2524

[26] Girshick, R. Fast R-CNN. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, Washington, DC, USA: IEEE Computer Society, 2015, ISBN 978-1-4673-8391-2, pp. 1440–1448, doi:10.1109/ICCV.2015.169. Available from: http://dx.doi.org/10.1109/ICCV.2015.169

[27] Ren, S.; He, K.; et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems 28*, edited by C. Cortes; N. D. Lawrence; D. D. Lee; M. Sugiyama; R. Garnett, Curran Associates, Inc., 2015, pp. 91–99. Available from: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf

[28] Owano, N. When AI is made by AI, results are impressive. *Tech Xplore [online]*, 2017, [cit. 02-05-2018]. Available from: https://techxplore.com/news/2017-12-ai-results.html

[29] Chung, J. S.; Senior, A. W.; et al. Lip Reading Sentences in the Wild. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 3444–3453, doi:10.1109/CVPR.2017.367. Available from: https://doi.org/10.1109/CVPR.2017.367

[30] Akbari, H.; Arora, H.; et al. Lip2AudSpec: Speech reconstruction from silent lip movements video. *CoRR*, volume abs/1710.09798, 2017, 1710.09798. Available from: http://arxiv.org/abs/1710.09798

[31] Abadi, M.; Agarwal, A.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Available from: https://www.tensorflow.org/

[32] Chollet, François and others. *Keras*. 2015. Available from: https://keras.io

[33] Chollet, François and others. *Keras: Sequential Model Guide*. 2015, [cit. 04-26-2018]. Available from: https://keras.io/getting-started/sequential-model-guide/

[34] Chen, L. *Keras.js*. 2016. Available from: https://github.com/transcranial/keras-js

[35] Google. Teachable Machine. *AI Experiments [online]*, 2017. Available from: https://teachablemachine.withgoogle.com/

[36] Pinto, R. Evaluation & Calculate Top-N Accuracy: Top 1 and Top 5. *Stackoverflow [online]*, 2016, [cit. 07-05-2018]. Available from: https://stackoverflow.com/a/37670482

[37] Castrillón Santana, M.; Déniz Suárez, O.; et al. ENCARA2: Real-time Detection of Multiple Faces at Different Resolutions in Video Streams. *Journal of Visual Communication and Image Representation*, April 2007: pp. 130–140.

[38] Itseez. *The OpenCV Reference Manual*. Third edition, April 2014. Available from: http://opencv.org/

[39] Mallick, S. Object Tracking using OpenCV (C++/Python). *Learn OpenCV [online]*, 2017, [cit. 05-07-2018]. Available from: https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/

[40] Chatfield, K.; Simonyan, K.; et al. Return of the Devil in the Details: Delving Deep into Convolutional Nets. *CoRR*, volume abs/1405.3531, 2014, 1405.3531. Available from: http://arxiv.org/abs/1405.3531

[41] Pedregosa, F.; Varoquaux, G.; et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, volume 12, 2011: pp. 2825–2830.

[42] Mathias, A. clmtrackr. *Github repository [online]*, 2017. Available from: `https://github.com/auduno/clmtrackr`

[43] Saragih, J. M.; Lucey, S.; et al. Deformable Model Fitting by Regularized Landmark Mean-Shift. *Int. J. Comput. Vision*, volume 91, no. 2, Jan. 2011: pp. 200–215, ISSN 0920-5691, doi:10.1007/s11263-010-0380-4. Available from: `http://dx.doi.org/10.1007/s11263-010-0380-4`

Glossary

debugging is the process of finding and resolving defects or problems within a computer program that prevent correct operation of computer software. 42

JavaScript is a multiplatform, object oriented scripting language used as a core technology of the World Wide Web, enabling creation of interactive web pages. 2, 19, 40, 45

NumPy is a popular Python library adding support for large, multi-dimensional arrays and matrices and enabling high level API for working with them. xiii, 31, 36

tensor can be tough as a generalization of matrix. E.g. a 3D tensor is a volume, which can be represented by stacking 2D matrices in a depth. 31

Acronyms

AI Artificial Intelligence. 1, 18

ALR Automated Lip Reading. 2

API Application Programming Interface. 18, 19, 43, 53

CPU Computer Processing Unit. 18

DAG Directed Acyclic Graph. 19

FPS Frame per Second. 4, 43

GPU Graphic Processing Unit. 14, 18, 19, 27, 31, 36, 38, 44

HOG Histogram of oriented gradients. 4

LRW Lip Reading in Wild. 1, 2, 20, 21, 25, 27, 33, 45

ML Machine Learning. 1, 6, 7, 20, 25, 42

NN Neural Network. ix, 2, 4–9, 12, 13, 15, 18–20, 23, 24, 31, 33, 45

PDF Portable Document Format. 17

RAM Random Access Memory. 31

SVM Support Vector Machines. 4, 13

TPU Tensor Processing Unit. 18

Appendix A

Contents of Enclosed CD

- `README.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `lip` ..... implementation sources
    - `LipReading.ipynb` ..... main implementation Jupyter Notebook
    - `LipReading.html` ..... html export of implementation
    - `Plotting.ipynb` ..... auxiliary Jupyter Notebook for plotting
    - `Plotting.html` ..... html export of plotting
    - `requirements.txt` ..... list of Python packages to install
    - `haar` ..... used Haar cascades XML files directory
    - `models` ..... trained Keras models and weights directory
    - `outputs` ..... outputs of training directory
    - `web` ..... web application implementation
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `BP_Horak_Jan_2018.pdf` ..... the thesis text in PDF format

Appendix B

Setup Guide

Here I describe the setup process of running the content of the source codes. The hierarchical structure of CD can be found in Appendix A.

I enclose both the source codes in Jupyter Notebooks, and their exported version to HTML, such that they can be viewed in a web browser directly without any installation.

For opening and running the code, a Python interpreter of version 3 or newer has to be installed (it can be downloaded here[21]), together with libraries listed in `requirements.txt` file.

Installing libraries can be done automatically with `pip` by issuing following command in the root folder of CD.

```
pip3 install -r src/lip/requirements.txt
```

Listing 9: Keras model example.

After a successful installation of all dependencies, a Jupyter localhost server needs to be started in the directory with Jupyter Notebooks in order to open them (see Listing 10).

```
cd src/lip/
jupyter notebook --ip=localhost --port=8888
```

Listing 10: Jupyter localhost start

Now the Notebooks can be accessed and opened in browser by visiting the following URL: `http://localhost:8888`.

[21] `https://www.python.org/downloads/`

Appendix C

LRW Dataset Words

about, absolutely, abuse, access, according, accused, across, action, actually, affairs, affected, africa, after, afternoon, again, against, agree, agreement, ahead, allegations, allow, allowed, almost, already, always, america, american, among, amount, announced, another, answer, anything, areas, around, arrested, asked, asking, attack, attacks, authorities, banks, because, become, before, behind, being, believe, benefit, benefits, better, between, biggest, billion, black, border, bring, britain, british, brought, budget, build, building, business, businesses, called, cameron, campaign, cancer, cannot, capital, cases, central, certainly, challenge, chance, change, changes, charge, charges, chief, child, children, china, claims, clear, close, cloud, comes, coming, community, companies, company, concerns, conference, conflict, conservative, continue, control, could, council, countries, country, couple, course, court, crime, crisis, current, customers, david, death, debate, decided, decision, deficit, degrees, described, despite, details, difference, different, difficult, doing, during, early, eastern, economic, economy, editor, education, election, emergency, energy, england, enough, europe, european, evening, events, every, everybody, everyone, everything, evidence, exactly, example, expect, expected, extra, facing, families, family, fight, fighting, figures, final, financial, first, focus, following, football, force, forces, foreign, former, forward, found, france, french, friday, front, further, future, games, general, george, germany, getting, given, giving, global, going, government, great, greece, ground, group, growing, growth, guilty, happen, happened, happening, having, health, heard, heart, heavy, higher, history, homes, hospital, hours, house, housing, human, hundreds, immigration, impact, important, increase, independent, industry, inflation, information, inquiry, inside, interest, investment, involved, ireland, islamic, issue, issues, itself, james, judge, justice, killed, known, labour, large, later, latest, leader, leaders, leadership, least, leave, legal, level, levels, likely, little, lives, living, local, london, longer, looking, major, majority, makes, making, manchester, market, massive, matter, maybe, means, measures, media, medical, meeting, member, members, message, middle, might, migrants, military, million, millions, minister, ministers, minutes, missing, moment, money, month, months, morning, moving, murder, national, needs, never, night, north, northern, nothing, number, numbers, obama, office, officers, officials, often, operation, opposition, order, other, others, outside, parents, parliament, parties, parts, party, patients, paying, people, perhaps, period, person, personal, phone, place, places, plans, point, police, policy, political, politicians, politics, position, possible, potential, power, powers, president, press, pressure, pretty, price, prices, prime, prison, private, probably, problem, problems, process, protect, provide, public, question, questions, quite, rates, rather, really, reason, recent, record, referendum, remember, report, reports, response, result, return, right, rights, rules, running, russia, russian, saying, school, schools, scotland, scottish, second, secretary, sector, security, seems, senior, sense, series, serious, service, services, seven, several, short, should, sides, significant, simply, since, single, situation, small, social, society, someone, something, south, southern, speaking, special, speech, spend, spending, spent, staff, stage, stand, start, started, state, statement, states, still, story, street, strong, sunday, sunshine, support, syria, syrian, system, taken, taking, talking, talks, temperatures, terms, their, themselves, there, these, thing, things, think, third, those, thought, thousands, threat, three, through, times, today, together, tomorrow, tonight, towards, trade, trial, trust, trying, under, understand, union, united, until, using, victims, violence, voters, waiting, wales, wanted, wants, warning, watching, water, weapons, weather, weekend, weeks, welcome, welfare, western, westminster, where, whether, which, while, whole, winds, within, without, women, words, workers, working, world, worst, would, wrong, years, yesterday, young