Prepoznavanje vrste govornog jezika korištenjem TensorFlow biblioteke

Hadžić, Ibrahim

UNIVERSITY OF ZAGREB

FACULTY OF ORGANIZATION AND INFORMATICS

V A R A Ž D I N

Ibrahim Hadžić

RECOGNITION OF NATURAL LANGUAGE TYPE USING TENSORFLOW

BACHELOR'S THESIS

Varaždin, 2018

UNIVERSITY OF ZAGREB

FACULTY OF ORGANIZATION AND INFORMATICS

V A R A Ž D I N

Ibrahim Hadžić

Identification number: 44108/15–R

Study programme: Information Systems

RECOGNITION OF NATURAL LANGUAGE TYPE USING TENSORFLOW

BACHELOR'S THESIS

Mentor:

Assoc. Prof. dr. sc. Markus Schatten

Varaždin, December 2018

Ibrahim Hadžić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Abstract

This paper introduces the reader to the problem of language identification (LID) based on audio content and to the implementation of a solution that uses a convolutional recurrent deep neural network (CRNN) and visual representation of sound – spectrograms. It will first guide through the current state of LID systems and then it will proceed to describe the method used for obtaining a dataset from YouTube and the steps taken in preprocessing and extracting features from that data. Furthermore, the paper will briefly explain how convolutional neural networks (CNN) and recurrent neural networks (RNN) work, present the hybrid architecture used for building the LID system and its results. Finally, it will give insight into a possible practical usage of such systems and showcase a proof of concept which, by combining the LID system with Google APIs for speech and translation, acts as real time translator.

Keywords

Language identification, language recognition, deep learning, audio classification, audio preprocessing, python, tensorflow.

Sadržaj

1. Introduction 1
2. Data 3
   2.1. Acquiring the Dataset 3
3. Data Preprocessing and Feature Extraction 4
   3.1. Audio loudness normalisation 4
   3.2. Exclusion of Irrelevant Frequencies 4
   3.3. Conversion of Audio Recordings to Spectrograms 4
      3.3.1. Fourier Transform and DFT 5
      3.3.2. Windowing and Hann Window 7
   3.4. Spectrograms – End Result 8
4. Deep Learning and Language Identification 9
   4.1. Tensorflow and Keras 9
   4.2. Neural Network Architectures 11
      4.2.1. Convolutional Neural Networks 11
      4.2.2. Recurrent Neural Networks 14
   4.3. Proposed Architecture 16
      4.3.1. Inception-Resnet v2 17
      4.3.2. Bidirectional LSTM 21
      4.3.3. Final Architecture 22
5. Results 23
6. Web Application 25
7. Conclusion 26
8. Literature 27
9. Table of Figures 31

1. Introduction

Natural language recognition, also known as language identification (LID), is the task of determining the language used in textual or audio content. Today, identifying the language from a text performs well and is widely used, very much unlike language identification from audio samples. Speech recognition systems, such as Alexa or Siri, are still not able to inherit the language in which the user's command was given and require the input language to be predefined for their proper functioning. Furthermore, LID systems of high accuracy seem to be the missing block for creating translation devices that can seamlessly translate audio from other languages to one's mother tongue, and vice versa.

Current state-of-the-art approaches to language identification are based on the classification of visual representation of audio samples. That means that the problem of language identification is transferred from audio domain to the image domain using signal processing techniques. There is an ongoing research being done on architectures inspired by human hearing with the goal of developing the field of machine hearing [1]. Convolutional neural networks were developed in a similar fashion [2] - through the influence of Fukushima's neocognitron neural network [3] that was inspired by the work in cat's visual cortex by Nobel prize winning neuroscientists Hubel and Viesel [4].

The biggest challenge that the language identification is facing today is that the current technology is not yet capable of having satisfactory accuracy on short audio samples. For example, saying the sentence „Hello, how are you today?“ takes up to two seconds if said in a normal, conversational, speed. Not only is it difficult for a LID system to detect a language from such a short sample, but it also gets more difficult with the implementaton of more languages, as it can get more easily confused. It is a problem relatable to humans as well – Muthusamy, Jain and Cole [5] experimented with and measured humans' ability to detect language in speech utterances of different durations. The subjects of the experiment had to classify utterances between ten languages. One of the experiments involved 10 native speakers of English and 2 native speakers of each of the other language and all of them participated in a training session. During that training, the subjects listened to 2 utterances of each language and did a dry run of the experiment where they listened to 8 utterances per language more. The average perfomance reported on all languages excluding English were 65.3%, 62.3%, 52.8% and 44.8% on the utterances of duration of 6, 4, 2 and 1 second, respectively.

The aim of this project is to create an LID model that can classify short speech utterances between three languages – Croatian, French and Spanish. Audio samples of duration of 3 seconds seemed the most suitable option for the method used in this paper. While being relatively short, 3-second speech have shown contain enough information in order for the neural network to generalise and have satisfactory results [6]. Moreover, a lot of attention will be given to audio data preprocessing and feature extraction as the crucial step that facilitates neural network's learning.

The first part of this paper will explain in which way was the data obtained, provide detailed information about the dataset used and will look into the importance of having high-quality data in supervised learning.

The second part of the paper will guide through the steps of preparing, preprocessing and extracting features from the data: segmentation of audio files to 3-second samples, audio normalisation, exclusion of mostly irrelevant frequencies and conversion of audio samples to spectrograms.

The third part will introduce the machine learning framework used in this project – Tensorflow [7], and it's high-level API for building and training deep learning models – Keras [8]. It will then explain how convolutional neural networks and recurrent neural networks function and their applications. Afterwards, it will present the architecture of the neural network, the procedure of training and the results of the trained model.

Finally, it will show a proof of concept of a real-time speech translator that combines Google APIs with the LID model developed in this project.

2. Data

Deep learning tasks require a great amount of labelled data and acquiring it is often costly both in time and resources. Moreover, having high-quality data is much more important than the neural network architecture used since the neural networks can only get as good as the data is. Data is like a road – if the road is bad, the car (neural network) won't get far and it won't be of importance if it's a lower or a higher class model.

2.1. Acquiring the Dataset

The method for obtaining the dataset used in this paper was proposed by Bartz, Herold, Yang and Meinel [9]. Since the Youtube contains hundreds of thousands of hours of content in virtually every relevant language, the mentioned method relies on extracting audio from selected channels and playlists that are labelled with the language that their videos contain. Therefore, instead of having to label each audio file separately, it is enough to specify only the playlist's or channel's language which will be then be inherited as the label of each separate audio file.

The playlists used in the project consist mainly of news and talk show videos. These two categories were used because they incorporate a variety of speakers – it is important to avoid the biases that could result from the neural network's association of a certain voice or a pitch with a language. Also, these categories contain occasional background noise like crowd noise or music which is useful for making the model more robust.

On the other side, this method has the disadvantage of introducing some noise to the data - parts of audio files that contain something other than speech, such as intro music, laughter, clapping or silence. Nevertheless, the number of these is not significant and the neural network should be able to generalise well in spite of them. It is a small price to pay for such an efficient way of obtaining a dataset.

Optionally, this method could be extended with a model that can classify audio files between those that contain speech and those that don't which would clean most of the noise in the dataset, but that has not been done in this project.

The dataset contains 2920 hours of speech data for three languages: Croatian, French and Spanish.

3. Data Preprocessing and Feature Extraction

In order to get the data ready for feeding it into the neural network, a number of steps are to be done. First of all, since the goal of this project is to have an LID system that can identify language from a 3-second speech utterance, all the downloaded audio files in the dataset are separated into 3-second segments and downsampled to 16000 Hz.

3.1. Audio loudness normalisation

Loudness normalisation is the process of levelling the amplitudes of an audio recording to a specified norm in order to adjust the perceived loudness of it. Normalising the data is a common step in machine learning pipelines because it allows the algorithms or neural networks to capture the patterns in an easier manner.

The norm for loudness normalisation used in this project is EBU R128 [10]. The mentioned guideline was created and recommended by the European Broadcasting Union and its primary use is for making the loudness level even between different TV and radio stations as well as for creators of commercials that are broadcasted on them.

3.2. Exclusion of Irrelevant Frequencies

Since the phonemes that the languages in the dataset consist of fall under 5kHz [9], all the frequencies above the mentioned level are excluded. The reasoning behind this decision is that the excluded frequencies do not contain any properties associated with speaking or with languages and therefore are irrelevant to the task of language identification.

3.3. Conversion of Audio Recordings to Spectrograms

A spectrogram is a two-dimensional visual respresentation of a sound. Time is usually located on the x-axis while the frequencies are set on the vertical axis. Even though a spectrogram is two-dimensional, it also contains a third dimension represented as a heat map that shows the amplitude of a frequency at a specific time. In practice, the amplitude is associated with a darker color if it is lower while the higher amplitudes are indicated with brighter colors.

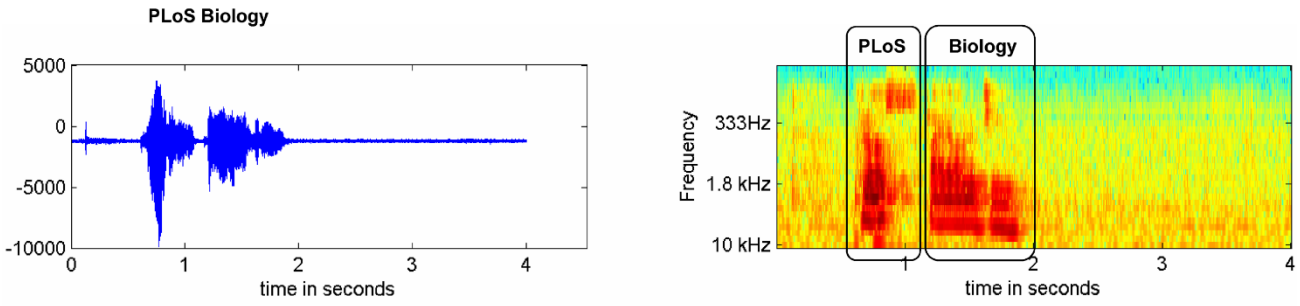


Figure 1: Waveform and spectrogram representation of a male person saying "PloS Biology" [11]

This project opts for the shades of grey for representing the spectrograms. This choice is made because greyscale images contain a single color dimension whereas RGB images contain three dimensions – red, green and blue dimension. As a result, the neural network will have less parameters to compute.

The sound processing utility called SoX [12] is used for spectrogram generation in the project. When SoX generates spectrograms, it does it by using mathematical functions crucial for signal processing – discrete Fourier's transform (DFT) and Hann windowing.

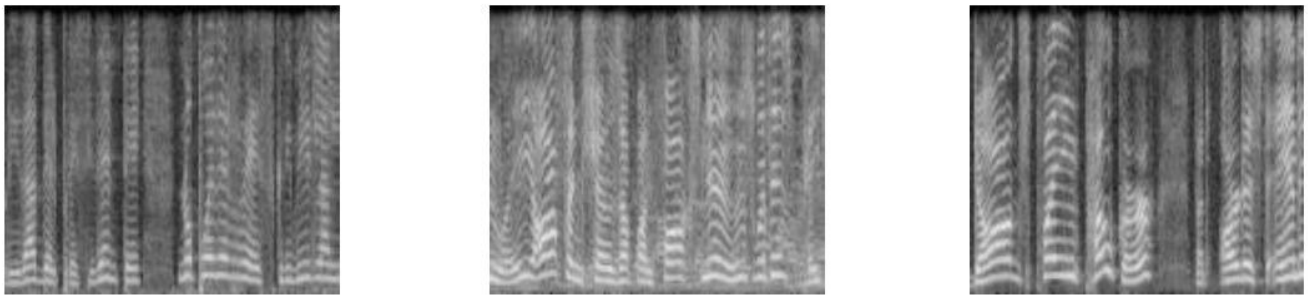


Figure 2: Example of spectrograms [author's work]

3.3.1. Fourier Transform and DFT

As it is already stated, spectrograms describe the energy of a frequency at a given time. With that said, and regarding the fact that sound is a waveform usually composed of more frequencies, an interesting question arises – What is being done in the process of generating a spectrogram that it is able to tell the energy of each, separate, frequency? Put simply – How is the decomposition of a sound done?

The Fourier transform (FT) is a mathematical function that, as its input, takes a waveform which, in essence, is a time-based function, and decomposes it into the frequencies that make it up [13]. In other words, it transfers the waveform from time-domain to frequency- domain. It has various practical usages such as picking-out the frequency of a

radio station that one would like to listen to or in data compression for ignoring the least important frequencies – just like how, for example, MP3 format does.

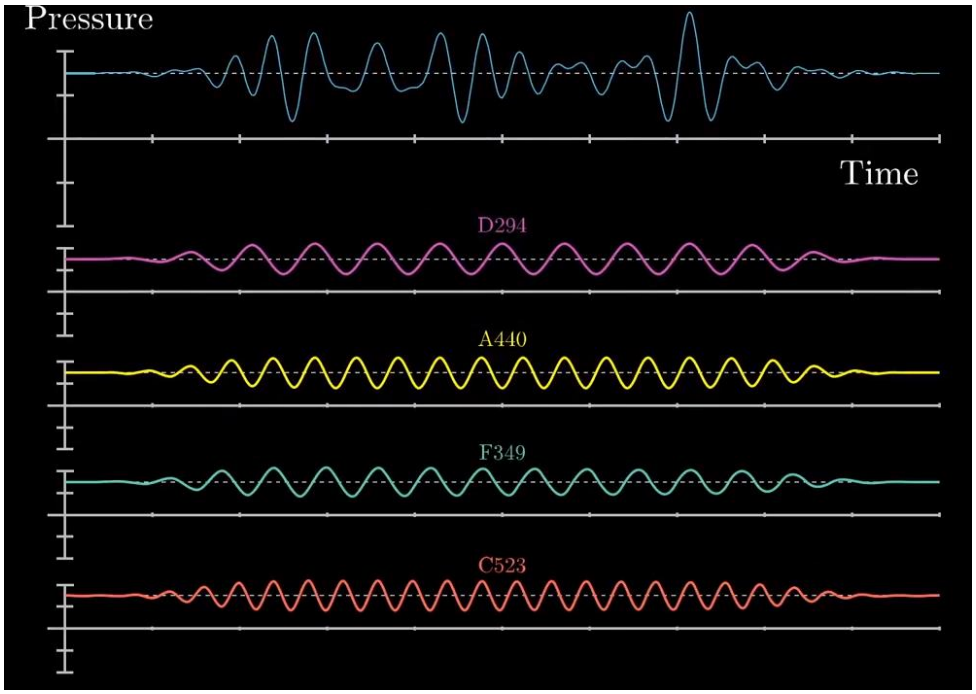


Figure 3: Bottom four waves make up the top waveform [14]

The waves that make a waveform are sines with different frequencies and amplitudes [15]. Once the Fourier transform decomposes a waveform, it provides the information for both the energy (intensity) of the amplitudes of the wave and its frequency, which is essentially the recipe for making that specific waveform.

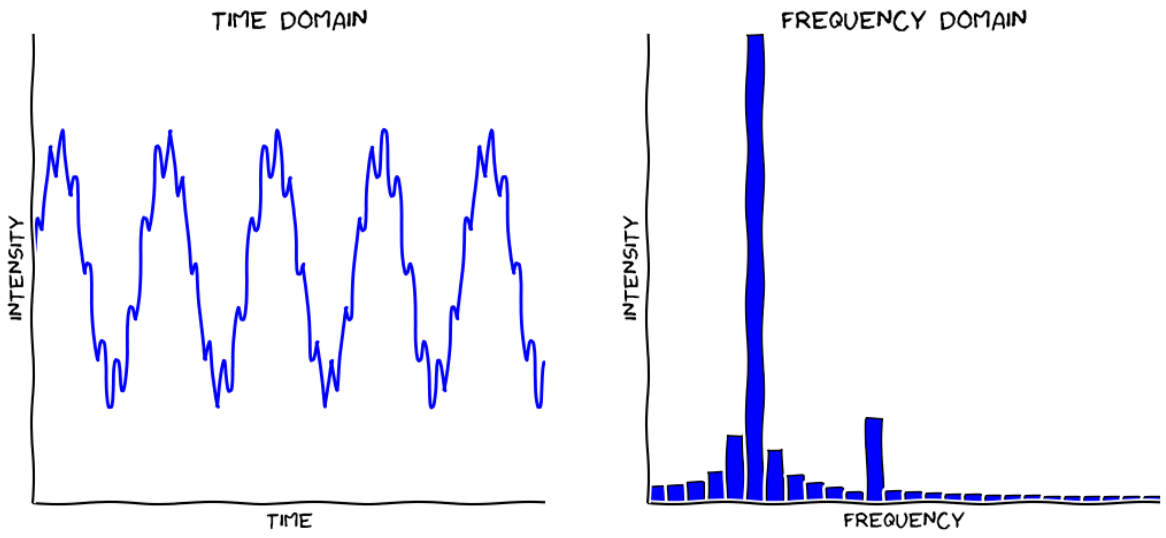


Figure 4 Waveform and the output of the Fourier transform [16]

When talking over a phone, the waveforms that the interlocutors produce are being digitalised by sampling them into discrete values. That means that the digitiser is “describing” the original waveform with a set of values for the purpose of making it understandable to digital devices like computers or telephones.

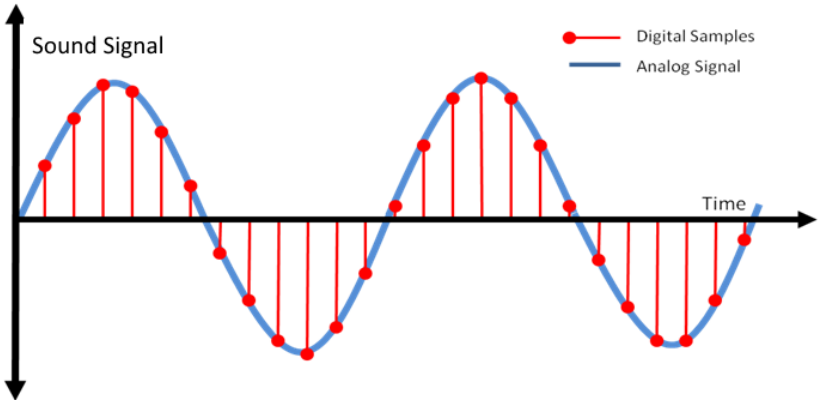


Figure 5: Sampling a signal [17]

The original Fourier transform, also know as continuous Fourier transform, works with continuous signals and is therefore not applicable to discrete signals. On the other hand, the discrete Fourier transform (DFT) is made just for that purpose. The DFT uses frequency bins for discretising frequencies – it acts like histograms do, ranges of frequency are put into separate frequency bins. In the project, 129 frequency bins are used.

3.3.2. Windowing and Hann Window

The way in which DFT processes a signal is by looking at its endpoints as if they were connected. When the signal is periodic and the number of times it cycles is an integer, the endpoints connect well and no problem is presented. Unfortunately, it usually does not repeat an integer number of times and it is when the transition between two endpoints become sharp. The term for sharp transitions is discontinuities. [15]

To avoid spectral leakage, the discontinuities can be smoothed using window fuction. Windowing, the process of applying a window function, reduces the discontinuities on each finite sequence which makes the aforementioned endpoints meet (figure 6) [15]. When windowing is done, the side lobes of a signal are weakened and thus the technique of overlapping the sequence with each other is applied in order to preserve the information.

Different window functions are suitable for different tasks, but the most commonly used one is Hanning window, which is also used in the project.

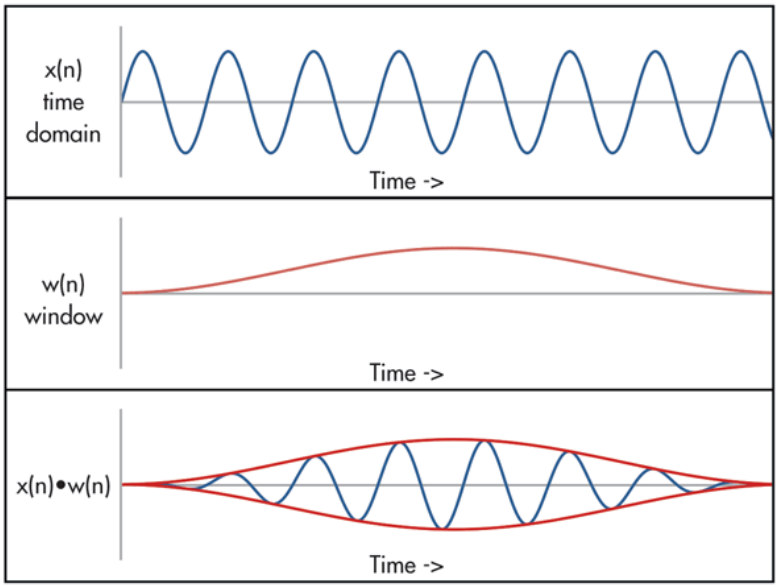


Figure 6: Applying a windows function on a signal [18]

Hanning window has the shape as it is shown in the figure 6. Unlike some other window functions, such as Hamming window, Hann window eliminates all discontinuity by being closed at both of its ends [15].

3.4. Spectrograms – End Result

The generated spectrograms are greyscale 150×129 images where the vertical axis is the representation of the 129 frequency bins produced by the DFT. Regarding the fact that the audio samples in the dataset were segmented into 3 seconds and since the time axis was rendered at 50 pixels per second, the resulting number of pixels equals 150, which explains the horizontal axis of the spectrogram resolution.

The advantage of visually respresenting audio is that it results in lower dimensionality compared to raw audio files while retaining enough information required for the task of sound classification [6].

4. Deep Learning and Language Identification

The approach in this paper is based on the "Language Identification Using Deep Convolutional Recurrent Neural Networks" by Bartz, Herold, Yang and Meinel [9]. The authors proposed a hybrid neural network combinations that joins convolutional and recurrent neural networks in order to capture both the spatial and sequential information. They achieve current state-of-the-art results using audio samples of duration of 10 seconds.

The aim of this paper is to explore the prospect of using significantly shorter audio samples in applying deep learning on the task of language identification. The neural network used is a combination of Inception-Resnet-v2 with a long short-term memory (LSTM) unit. The project was done in Python using Tensorflow.

4.1. Tensorflow and Keras

Tensorflow is an open-source machine learning library for both research and production. It was created by the Google Brain team and it incorporates Python front-end API for convenient application development with the backend written in C++ for performance reasons. Nowadays, fronted APIs for languages like R, Java and C++ exist as well, but Python remains being the best supported. It also features implementation of it for mobile and embedded devices, Tensorflow Lite, as well as for training and deploying machine learning models in the browser, Tensorflow.js. [7]

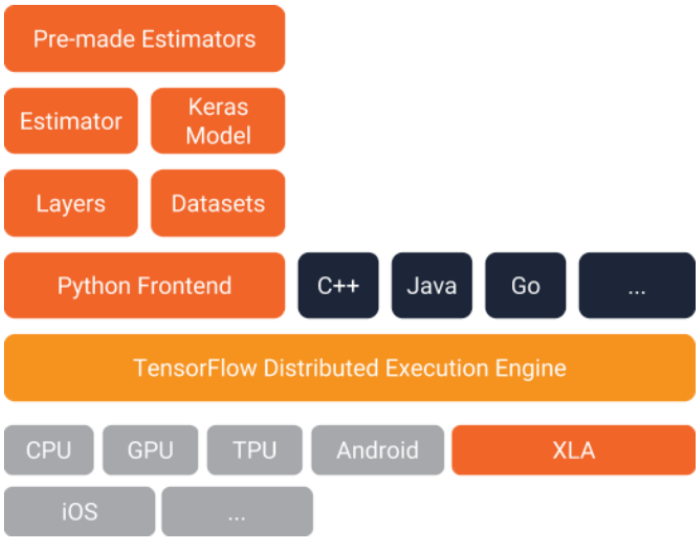


Figure 7 Tensorflow architecture [19]

The principle in which Tensorflow works is simple: firstly, the graph of computations is defined in the front-end API and then it is ran efficiently in the backend. A very important feature that Tensorflow has is its ability to break up a graph into several chunks for running it across multiple CPUs or GPUs. Furthermore, distributed computing is supported as well, where the computation can be done across hundreds of servers. [20]

Keras is a Python API for neural networks with a higher level of abstraction than what Tensorflow's front-end API offers. Keras is imagined as an interface that supports multiple backends, amongst which is also Tensorflow. It allows fast experimentation and prototyping through its user friendliness, extensibility and modularity which made it gain a lot of popularity. Tensorflow has implemented its specification of the Keras API and has been, ever since, recommending it as the way to go for general purpose. [7][8]

Getting started with Tensorflow can be as easy as writing couple of lines of code. Tensorflow provides the users with the most popular datasets, such as MNIST hand written digit database [21] used in the example in the figure 8.

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Figure 8: Tensorflow code for creating, training and evaluating a neural network on MNIST dataset [7]

In the code above we can see that the data is divided into two sets, one for training and one for testing (evaluating) it. In machine learning code, *Y* usually denotes the label of the data, while the actual data (image, text etc.) is represented by *X*. Since the MNIST dataset contains images of hand written digits, those images have to be normalised before passing

them to the neural network. A pixel is a value between 0 and 255, so the images are normalised by dividing its pixels with 255 which as a result has the values of pixels being floats between 0 and 1. This step is an example of what is called data preprocessing.

Afterwards comes the definition of a neural network, which in the example is a very simple one. In order to train a neural network, one needs to specify the optimiser and the loss function for the neural network. In a simplified explanation, they are what guides the neural network in its training. Finally, the neural network can be trained by fitting it the prepared data and labels. If the training was successful, the model will be able to recognise previously unseen digits with a certain accuracy.

4.2. Neural Network Architectures

The convolutional recurrent neural network (CRNN) that is used in the project is a hybrid between convolutional neural network (CNN) and recurrent neural network (RNN). Thus, the mentioned two separate neural network architectures will be explained separately before presenting the hybrid one.

4.2.1. Convolutional Neural Networks

As shown in the introduction of this paper, the convolutional neural networks are an example of neuroscientific principles influencing deep learning [22]. Since convolutional neural networks took inspiration from visual cortex, one could imagine that they would be successful on the computer vision tasks. They indeed have revolutionised the field – in the ILSVRC ImageNet challenge [23] the top-5 error went from 26% to just 3% [20]. Convolutional networks have also found their application in the field of natural language processing.

Unlike the traditional fully-connected neural networks, neurons in a CNN are connected only with a number of other neurons. Each neuron in a convolutional layer, the building block of a CNN, covers only a number of pixels in its receptive field (figure 9). The sparse interaction (figure 11) between neurons results in fewer parameters to compute but also presents a fundamental feature of CNNs – hierachical learning. Since the receptive field of a neuron increases as it is located deeper in the network, it allows it to find complex features by combining the elements from the shallower neurons. For example, the first hidden layer will learn low-level features such as edges, the hidden layers in the middle may learn features such

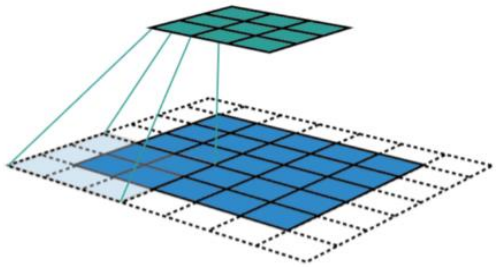


Figure 9: Receptive field [24]

as rectangles, curves or circles and the last hidden layer will learn complex features by combining the previous lower-level ones into something that resembles a shape of a car, house, or anything else depending on what the task was (figure 10). [20][22]

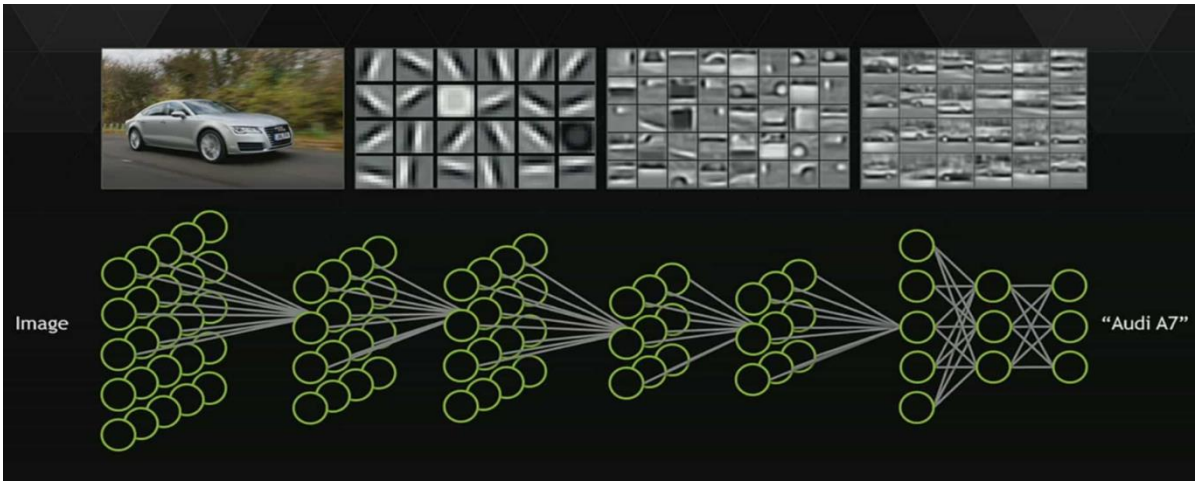


Figure 10: Examples of learned feature maps [25]

The weights of a specific neuron in a CNN are called filters. Each filter is able to detect a certain pattern that it learned, as explained in the previous paragraph. In CNNs, every member of a filter is used at every position of the input which enables the detection of a pattern at any location. This characteristic of CNNs is called *translation invariance*. Using the same weights on different locations, known as parameter sharing, also greatly reduces the number of parameters to compute. Traditional neural networks, on the contrary, calculate and use each neuron's weights only once. Furthermore, stacking up the feature maps allows the CNN to detect multiple features from an input at once. [22]

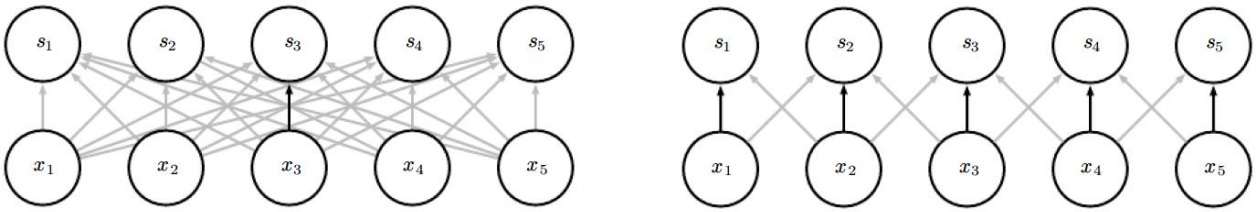


Figure 11: Fully connected neurons (left) and sparsely connected neurons (right) [22]

In order to preserve the height and the width of the output of the convolution, zero padding is used. Zero padding, as the name says, pads the input with zeros before computing the output.

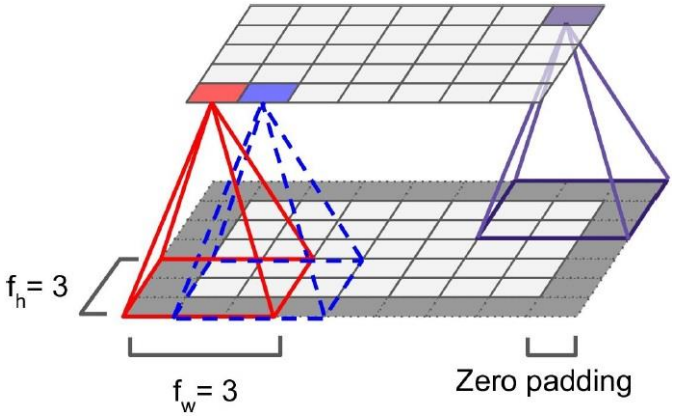


Figure 12: Zero padding [20]

When applying a filter on an input, a very useful parameter for defining the dimensionality of the output can be tweaked – stride. Stride is the number of steps between two consecutive receptive fields and, if the number of steps is increased, results in an output of lower height and width [20].

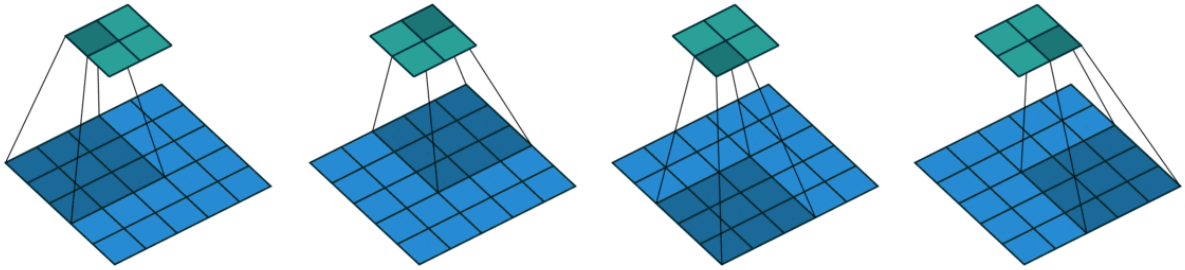


Figure 13: Stride with step size of 2 [26]

These were the parts of the convolutional layer in a CNN. Another fundamental part of CNNs are pooling layers. A pooling layer's purpose is to reduce the computational load and memory usage by subsampling the input. It shares the same parameters as convolutional layer does: size, stride and, optionally, padding. Pooling layers do not learn, thus they have no weights – their sole function is aggregating the inputs and returning, most commonly, the max value or the average value. [20]

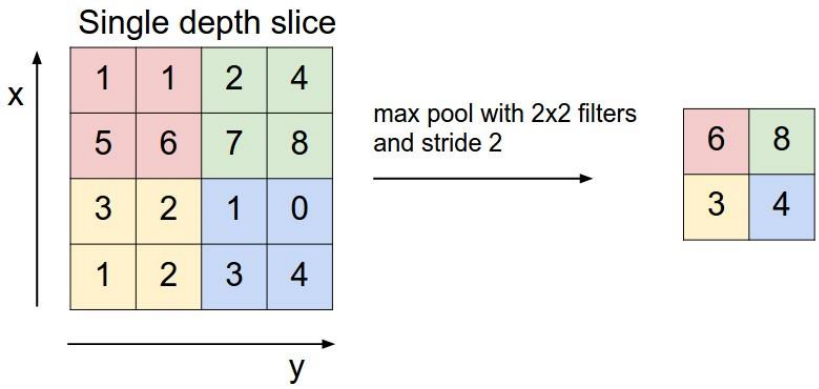


Figure 14: Max pooling with 2x2 filter and stride 2 [27]

CNN architectures often repeat a number of stacked convolutional layers followed by a pooling layer for as much as it is necessary, and are usually ended with one or two fully connected layers for outputing the prediction, possibly with a dropout layer between them in order to avoid overfitting.

4.2.2. Recurrent Neural Networks

Both the traditional and convolutional neural networks require the input and the output to be of a fixed size. For example, a CNN does not accept any other image size than the size that it was trained on and will return a vector of a specific length containing the class probabilities. While such behaviour is suitable for numerous problems, there are cases in which such limitation makes them impossible to be applied on.

Machine translation, sentiment classification and image captioning are some of those cases. In image captioning, the input (image) is of a fixed size, but the output size, being the description of the content on the image, is variable. On the other hand, evaluating products' reviews in order to find out if the customers are satisfied (sentiment classification) is the other way round. It outputs a fixed-size value – the percentage of the positivity of a review, but does it by evaluating the review text that can be of arbitrary length. Machine translation, unlike the previous two examples, has sequential data on both of its ends. It accepts an arbitrary number of words in one language and outputs a number of words in another language, which may be of a different size than the input due to the nature of the languages.

Recurrent neural networks are type of neural networks that, instead of looking only at the input given at the moment, take into account both the actual input and the history of inputs fed in the past [28]. This feature of RNNs allows them to "remember" and use that knowledge

to make a better judgement. For example, RNNs can be used to generate music because they can recall which chords it already generated and build progressions based on it.

The mentioned property of RNNs is achieved by expanding the neurons with a connection that points backwards. The added connection serves for sending the produced output back to itself for aiding the generation of the following outputs. The connection can also be represented in a more intuitive way by *unrolling the network throught time* (figure 15). [20]

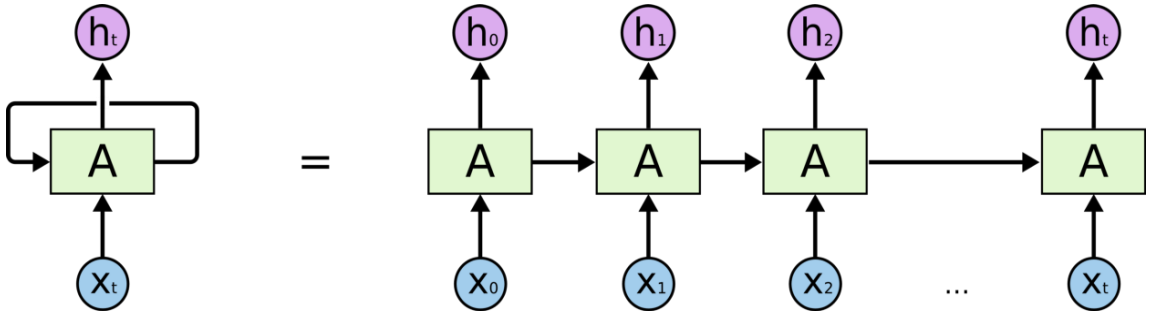


Figure 15: Recurrent neuron (left) and recurrent neuron unrolled through time (right) [29]

There are several different types of RNNs based on the type of input and output they have. One-to-many takes a single input and outputs multiple values (e.g. image captioning). Many-to-one takes a sequence and outputs a single value (e.g. sentiment classification of reviews). There are two types of many-to-many RNNs: the simple many-to-many architecture and the encoder-decoder architecture. The example of the first one would be music generation – when RNN produces music, the first input may be the desired genre but the neural network will, at each step of producing new sound, take the output from the step before and use it as its input. On the other hand, the encoder-decoder architecture is very useful in tasks such as machine translation where the input and output are often of different length. The encoder encodes the input into a vector and the decoder unwraps it to produce the output. [20][28]

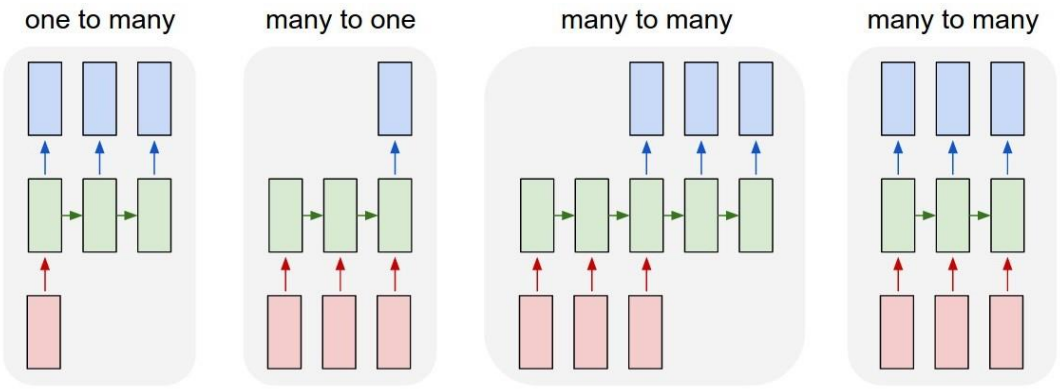


Figure 16: Types of RNNs, from left to right, one to many, many to one, many to many (encoder-decoder) and simple many to many [28]

When training an RNN, the backpropagation is done by unrolling the network and calculating the loss and updating the weights at each time step. It is oftend denoted as *backpropagation through time*.

Sometimes it is useful to make the RNN able to look into the future as well. Fortunately, it is not as hard to implement it as it may sound. To make an RNN bidirectional, as presented in [30], it is only necessary to specify that it should "read" the sequence not only from left to right, but also from right to left. In that way, the bidirectional recurrent neural network (BRNN) can make a prediction based on all of the inputs around it. For example, when doing speech recognition the sentence "The rain is falling down", a typical RNN would have a problem determining if the word "rain" or "reign" is told, whereas a BRNN would be able to look into what was told afterwards in order to maximise the chance of making a correct prediction.

The architecture that will be proposed in this paper contains a special kind of RNN called long short-term memory (LSTM) and will be discussed in the next section.

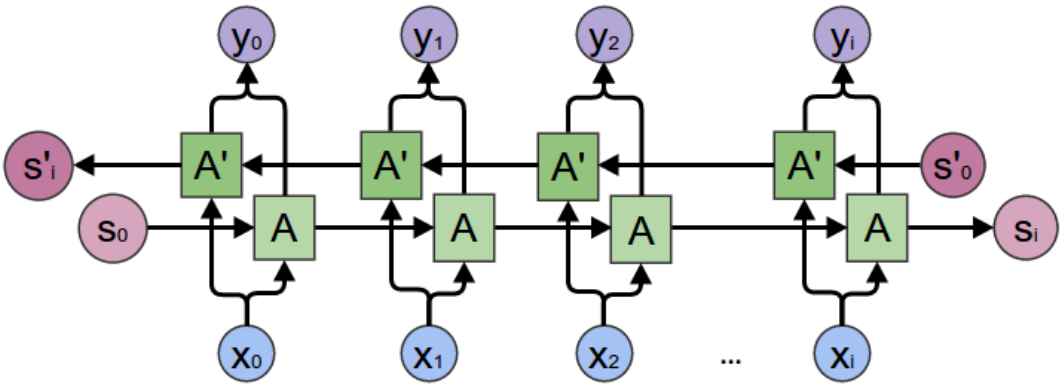


Figure 17: Bidirectional RNN [29]

4.3. Proposed Architecture

In [9], the authors experimented with a combination of Inception v3 CNN architecture and a bidirectional LSTM, achieving test accuracy as high as 96% on the dataset they collected. By comparing it to a simpler architecture of the same form, in which the CNN part is much shallower, the deeper architecture demonstrates greater robustness on noisy data – achieving 89% accuracy on the noisiest data.

The architecture that will be used in this thesis is a step forward from the mentioned architecture. Instead of Inception v3, the deeper Inception-Resnet v2 will be used in

combination with the bidirectional LSTM unit. In this section, both of the components of the final architecture will be presented separately before explaining the joint architecture.

4.3.1. Inception-Resnet v2

Inception-Resnet v2 [31] is a CNN architecture from the Inception family. It gradually evolved from the first Inception neural network – GoogLeNet [32], over Inception v2 and Inception v3 [33] and finally, Inception v4 [31]. Each one of the architectures brought improvements mainly regarding the Inception module, the building block of Inception networks. Inspired by the results that residual neural networks achieved [34], the authors of [31] experimented with combining the residual connections, main characteristic of residual networks, with the Inception module. The end-result, Inception-Resnet v2, is a very deep network that is able to achieve higher accuracy on a lower number of epochs.

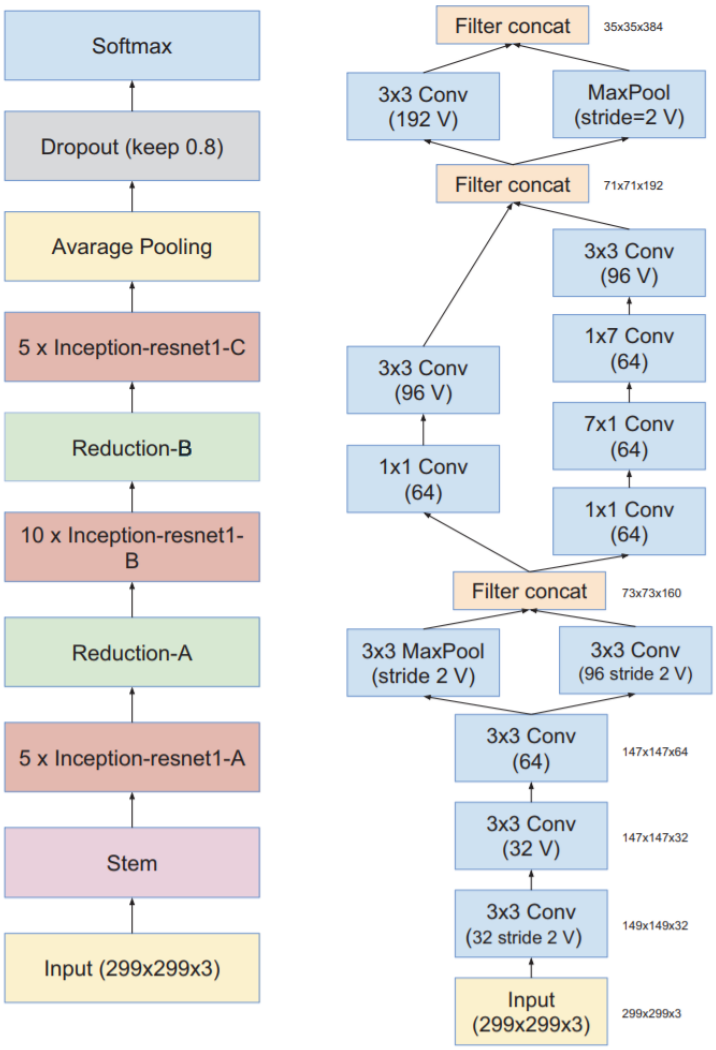


Figure 18: Inception-Resnet v2 architecture (left) and its stem (right) [31]

The architecture, as shown in the figure 18, features three different types of Residual Inception modules, named A, B and C, and two distinct reduction blocks. In order to understand the Residual Inception module, it is necessary to be familiar with the residual connections.

Residual connection (figure 19) is a simple concept invented as a way to resolve the problems of the vanishing gradient and exploding gradient that can appear in very deep neural network. The intuition behind it is that instead of computing the output "from scratch", the residual block can only calculate the changes that would make the input better, add them to the input and return it as its output. In the words of the authors of residual neural networks it is explained as follows: "Instead of hoping each stack of layers directly fits a desired underlying mapping, we explicitly let these layers fit a residual mapping. (…) The original mapping is recast into F(x)+x. We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers." [34, pp. 2].

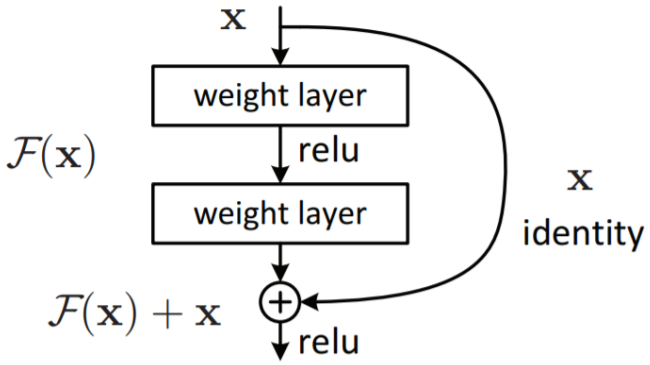


Figure 19: Residual learning [34]

The Residual Inception module combines the residual learning with Inception module by adding a residual connection to it. The modules used in the Inception-Resnet v2 can be seen in the figure 20. The principle on which Inception module works is by making the neural network wider instead of deeper. The expanded width allows it to capture complex patterns at various scales [20]. The initial 1×1 convolutions are used only to reduce dimensionality on the channel axis in order to make the following convolutions lighter. All of the convolutions use zero padding for preserving the width and height. It is important because after each separate calculation in the module, their outputs are concatenated along the depth dimension, which would not be possible if they differed in their heights and widths. Another characteristic of Inception modules is the split of an n×n convolution to two convolutions – 1×n and n×1 stacked on top of each other. The rationale for it is that it requires much fewer parameters while

producing the same result. An important detail in the Residual Inception module is that there is a 1×1 convolution after the concatenation for the reason of permitting the addition between the identity passed through the residual connection and the actual output by aligning their dimensions [31].

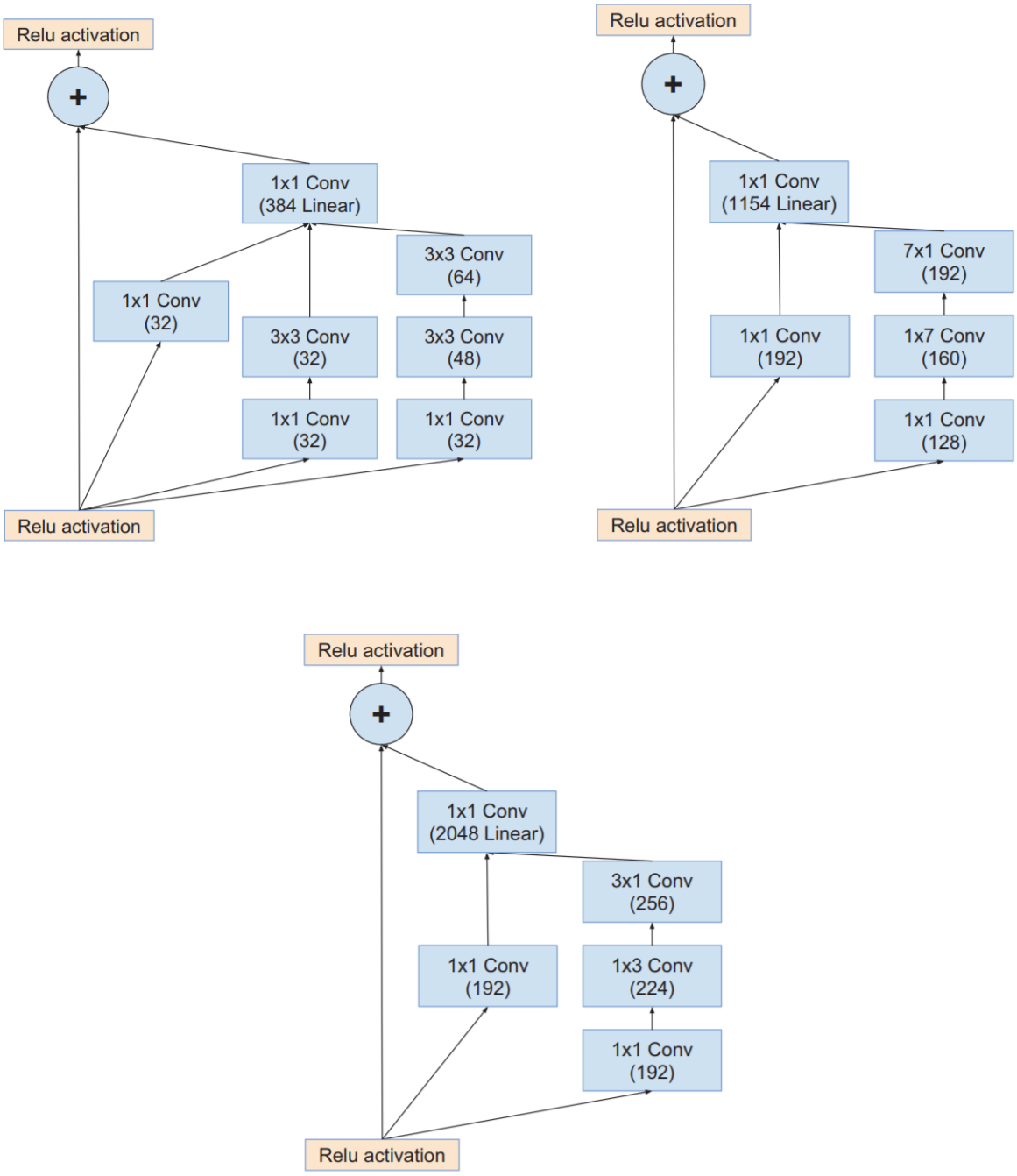


Figure 20: Residual Inception modules A (left), B (right), C (bottom) [31]

The reduction modules in the Inception-Resnet v2 are essentially slightly modified Inception modules. (figure 21). These modules, unlike the ones previously explained, contain average pooling followed by a 1×1 convolution on one of their branches.

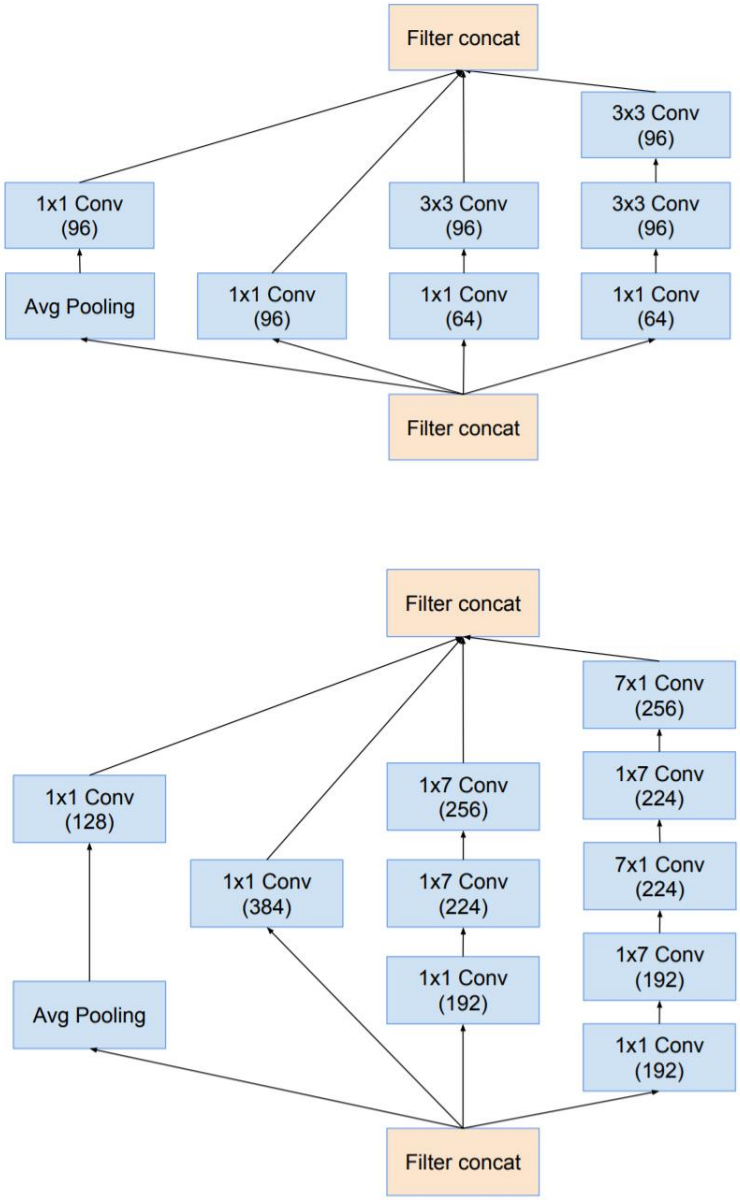


Figure 21: Reduction modules A (top) and B (bottom) [31]

4.3.2. Bidirectional LSTM

A problem faced by RNNs when handling long sequences is their lack of ability to remember the information from the beginning of the sequence. The inputs that are farther in the past tend to fade away which makes it easy for the network to misinterpret the sequence. For example, if a review mentioned at the beginning that the movie was great and continued to explain what could have been better, an RNN could lose the information from the start of the review and rate the review as negative. In order to fight that behaviour, long short-term memory cells were introduced. [20]

The idea behind the long short-term memory (LSTM) [35] is to have two different states of memory – long-term ($c_{(t)}$) and short-term ($h_{(t)}$). As the short-term state flows through the network, the LSTM tries to learn what to store in the long-term state and what to discard [20].

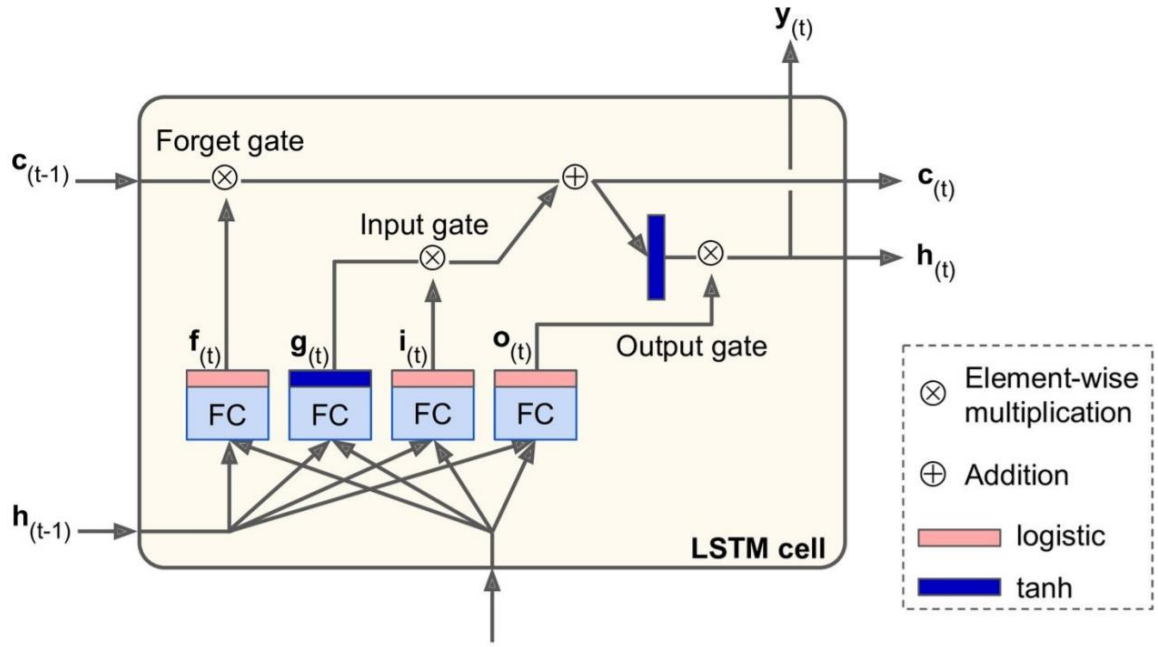


Figure 22: LSTM [20]

To be able to do that, LSTM uses several gates and fully connected layers (figure 22). The layer $g_{(t)}$ analyses the input $x_{(t)}$ and the previous short-term state $h_{(t-1)}$ and partially stores the output into the long-term state. The three other layers use logistic activation so their outputs range from 0 to 1, which are then used for closing or opening the gates that they are connected to. The forget gate ($f_{(t)}$) controls which parts of the long-term state are to be erased, the input gate ($i_{(t)}$) decides what is to be added to the long-term state and the output gate ($o_{(t)}$) controls which parts of the long-term state should be read and output at the actual time step. [20]

Making an LSTM cell bidirectional gives it the ability to process long sequences with taking into account the inputs from both before and after the input, no matter how "far" they are.

4.3.3. Final Architecture

The final architecture, Inception-Resnet-LSTM CRNN (figure 23) comprises the elements explained in the previous two sections with an addition of the "binding material" between them. After the last convolution of the Inception-Resnet part of the network, the usual layers are removed. Instead of them goes a permute layer, which changes the order of the dimensions whose output is then stacked and sliced along the time axis. Finally, the prepared input is fed to the bidirectional LSTM layer which is followed by a fully-connected layer and softmax activation for determining the the output (language).

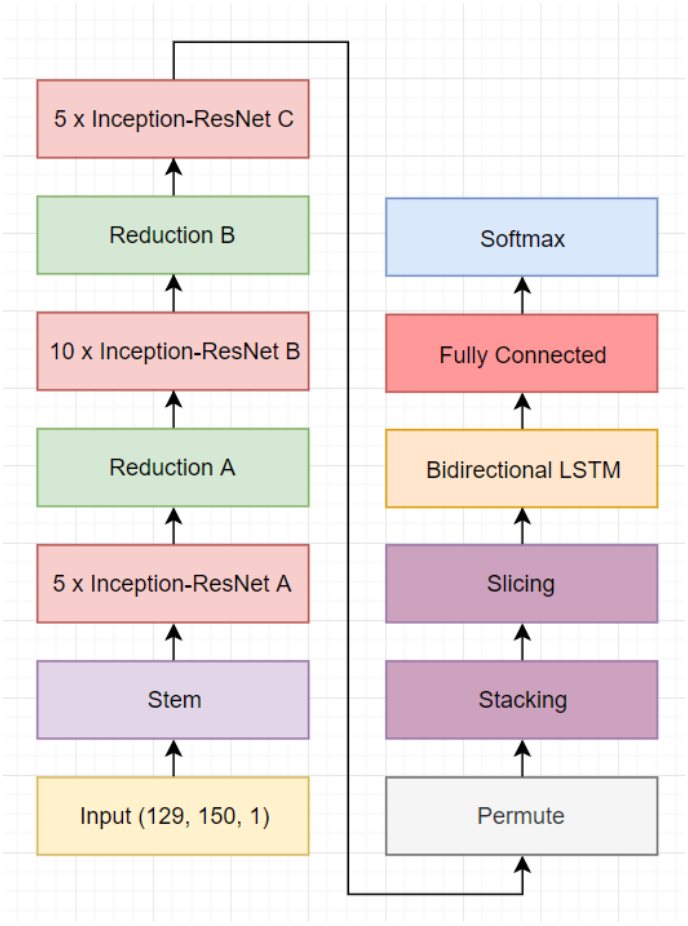


Figure 23: Inception-Resnet-LSTM CRNN [author's work]

5. Results

The network was trained on the collected dataset using two Nvidia Tesla P100 GPUs, batch size of 256 and Adam optimiser with the learning rate of 0.001. The data is evenly distributed and divided into training (70%), validation (20%) and testing (10%) sets.

Due to the budget limitations, the training was stopped after the 7th epoch. In total, the training lasted 19 hours.

```
Training using multiple GPUs..
Epoch 1/50
9591/9592 [============================>.] - ETA: 0s - loss: 0.1962 - acc: 0.9235
Epoch 00001: val_acc improved from -inf to 0.93711, saving model to logs/2019-01-05-21-33-35/weights.01.model
9592/9592 [==============================] - 10049s 1s/step - loss: 0.1962 - acc: 0.9235 - val_loss: 0.1611 - val_acc: 0.9371
Epoch 2/50
9591/9592 [============================>.] - ETA: 0s - loss: 0.1120 - acc: 0.9572
Epoch 00002: val_acc did not improve from 0.93711
9592/9592 [==============================] - 9681s 1s/step - loss: 0.1120 - acc: 0.9572 - val_loss: 0.3546 - val_acc: 0.8722
Epoch 3/50
9591/9592 [============================>.] - ETA: 0s - loss: 0.0898 - acc: 0.9660
Epoch 00003: val_acc improved from 0.93711 to 0.95103, saving model to logs/2019-01-05-21-33-35/weights.03.model
9592/9592 [==============================] - 9668s 1s/step - loss: 0.0898 - acc: 0.9660 - val_loss: 0.1332 - val_acc: 0.9510
Epoch 4/50
9591/9592 [============================>.] - ETA: 0s - loss: 0.0769 - acc: 0.9709
Epoch 00004: val_acc improved from 0.95103 to 0.95468, saving model to logs/2019-01-05-21-33-35/weights.04.model
9592/9592 [==============================] - 9665s 1s/step - loss: 0.0769 - acc: 0.9709 - val_loss: 0.1198 - val_acc: 0.9547
Epoch 5/50
9591/9592 [============================>.] - ETA: 0s - loss: 0.0676 - acc: 0.9743
Epoch 00005: val_acc did not improve from 0.95468
9592/9592 [==============================] - 9693s 1s/step - loss: 0.0676 - acc: 0.9743 - val_loss: 0.1298 - val_acc: 0.9517
Epoch 6/50
9591/9592 [============================>.] - ETA: 0s - loss: 0.0597 - acc: 0.9775
Epoch 00006: val_acc did not improve from 0.95468
9592/9592 [==============================] - 9684s 1s/step - loss: 0.0597 - acc: 0.9775 - val_loss: 0.1393 - val_acc: 0.9518
Epoch 7/50
9591/9592 [============================>.] - ETA: 0s - loss: 0.0527 - acc: 0.9802
Epoch 00007: val_acc improved from 0.95468 to 0.96197, saving model to logs/2019-01-05-21-33-35/weights.07.model
9592/9592 [==============================] - 9682s 1s/step - loss: 0.0527 - acc: 0.9802 - val_loss: 0.1033 - val_acc: 0.9620
```

Figure 24: Output of the network's training [author's work]

The neural network achieves the accuracy of 96% on the test set. Considering the size of the dataset and the behaviour of the neural network while training, it is very likely that even better results would have been achieved if it was trained longer. Furthermore, it shows to perform well in the practice, or more specifically, when using the web application developed as a part of this project which allows users to record the input and to get the language prediction by the trained model (see section 6). It achieves a high accuracy when a native speaker is talking, while the accuracy is noticeably lower when a foreign accent is present.

The confusion matrix (figure 25) shows that the model performs best, by a narrow margin, on Croatian. False predictions for both Spanish and French were more often cofused for Croatian than for the other language.

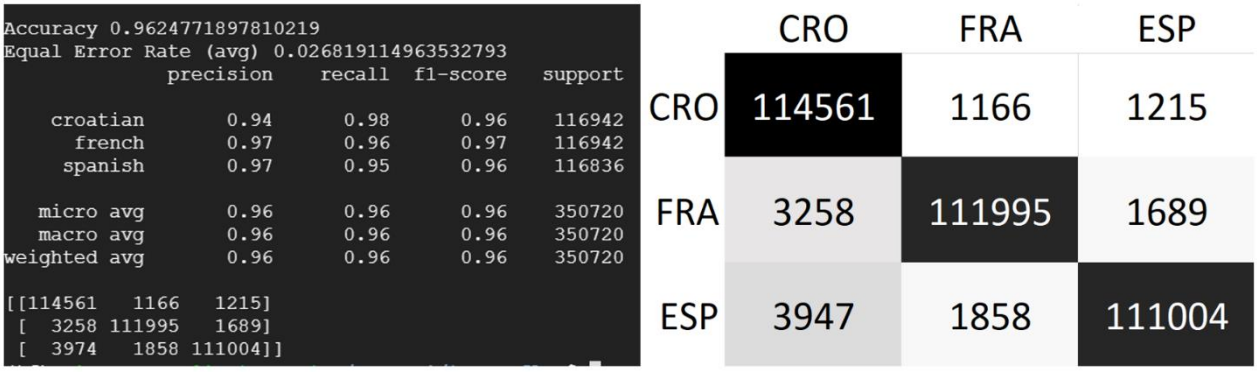
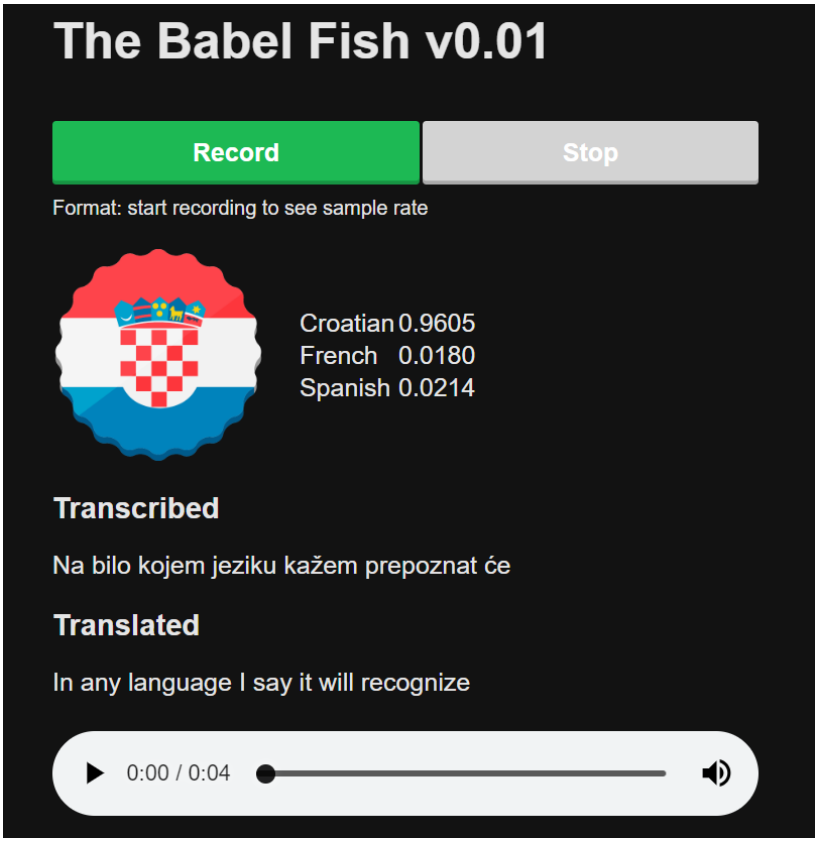


Figure 25: Confusion matrix; output after the evaluation of the model (left) and a heatmap of it (right)

The results have shown that it is possible to train a language identificator on very short audio samples with surprisingly satisfactory results. In the next section a possible use case for its application will be presented.

6. Web Application

The web application is meant as a mean of demonstrating the performance of the trained model by allowing the users to record their own sentences and get a prediction of the language they were told in. The web application's main functionality is to output the flag of the detected language and the probabilities of its confidence for each of the languages. On top of that, three Google APIs were added: Speech-to-Text [36], Translate [37] and Text-to-Speech [38]. Their purpose is as follows: once the model detects the language of the recording, the language and the recording are passed to the Speech-to-Text API which needs to know in which language is the recording in order to transcribe it. Once the transcription is obtained, it is passed on to the Translate API that translates it to English. Finally, the English translation of the recording is given to the Speech-to-Text API to convert it to speech which is then played by the web application. Essentially, the user perceives the application as a speech-to-speech translator.



7. Conclusion

In this paper, a large dataset for three languages was created for the task of language identification. The proposed convolutional recurrent neural network architecture is able to capture both spatial and sequential information from the visual representation of sound – spectrograms. By doing it, it achieves high performance in spite of the speech utterances being relatively short – 3 seconds. Furthermore, a possible use case is presented: an application that allows its users to record a sentence told either in Croatian, French or Spanish which is then processed by the model in order to detect the language it was told it. With that information, Google APIs are used in order to provide the English translation of the sentence and to return it in speech form. The application demonstrates the performance of the trained model which achieves very accurate predictions for native speakers while having a higher error when a foreign accent is present. The code used for this paper is open-sourced and available: the code for obtaining and preprocessing a dataset and training a neural network can be found at [39], while the speech-to-speech translator web application is at [40].

8. Literature

[1] R. Lyon, “Machine Hearing: An Emerging Field [Exploratory DSP],” *IEEE Signal Processing Magazine*, vol. 27, no. 5, pp. 131–139, 2010.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[4] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cats visual cortex,” *The Journal of Physiology*, vol. 160, no. 1, pp. 106–154, Jan. 1962.

[5] Y. Muthusamy, N. Jain, and R. Cole, “Perceptual benchmarks for automatic language identification,” *Proceedings of ICASSP 94. IEEE International Conference on Acoustics, Speech and Signal Processing*, vol.1, pp. 333-336, 1994.

[6] L. Wyse, *Audio spectrogram representations for processing with convolutional neural networks*, arXiv:1706.09559 [cs.SD], Jun 2017.

[7] Martín Abadi et al. (2015). Tensorflow: Large-Scale Machine Learning on Heterogeneous Systems. [Online]. Available at: https://tensorflow.org [Accessed 26 Dec. 2018].

[8] F. Chollet et al. (2015). Keras. [Online]. Available at: https://keras.io/ [Accessed 26 Dec. 2018].

[9] C. Bartz, T. Herold, H. Yang, and C. Meinel, “Language Identification Using Deep Convolutional Recurrent Neural Networks,” *International Conference on Neural Information Processing*, pp. 880–889, 2017.

[10] European Broadcasting Union, “R128. Loudness Normalisation and Permitted Maximum Level of Audio Signals”. [Online]. 2014. Available at: https://tech.ebu.ch/docs/r/r128.pdf [Accessed 27 Dec. 2018].

[11] C. Büchel and M. Sommer, “What Causes Stuttering?,” *PLOS ONE*. [Online]. Available: https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.0020046. [Accessed: 28-Dec-2018].

[12] SoX - Sound eXchange. [Online]. Available: http://sox.sourceforge.net/. [Accessed: 28-Dec-2018].

[13] “What is the Fourier Transform?,” Techopedia.com. [Online]. Available: https://www.techopedia.com/definition/7292/fourier-transform. [Accessed: 28-Dec-2018].

[14] 3Blue1Brown, “But what is the Fourier Transform? A visual introduction.,” *YouTube*, 26-Jan-2018. [Online]. Available: https://www.youtube.com/watch?v=spUNpyF58BY. [Accessed: 29-Dec-2018].

[15] “Understanding FFTs and Windowing,” National Instruments. [Online]. Available: http://www.ni.com/white-paper/4844/en/. [Accessed: 29-Dec-2018].

[16] “Think DSP – Introduction to Signal Processing Using Python,” *Adafruit Industries - Makers, hackers, artists, designers and engineers!*, 12-Jan-2015. [Online]. Available: https://blog.adafruit.com/2015/01/12/think-dsp-introduction-to-signal-processing-using-python/. [Accessed: 29-Dec-2018].

[17] Cocoon, “Audio, ADC and Raspberry Pi - oh my! | Blog,” *Cocoon*, 23-Jan-2018. [Online]. Available: https://cocoon.life/blog/audio-adcs-raspberry-pis-oh/. [Accessed: 29-Dec-2018].

[18] L. Chioye, “Choose The Right FFT Window Function When Evaluating Precision ADCs,” *Electronic Design*, 06-Dec-2013. [Online]. Available: https://www.electronicdesign.com/analog/choose-right-fft-window-function-when-evaluating-precision-adcs. [Accessed: 29-Dec-2018].

[19] “Introduction to TensorFlow Datasets and Estimators,” *Google Developers Blog*, 12-Sep-2017. [Online]. Available: https://developers.googleblog.com/2017/09/introducing-tensorflow-datasets.html. [Accessed: 29-Dec-2018].

[20] Géron Aurélien, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: OReilly Media, 2017.

[21] Y. LeCun, C. Cortes, and C. Burges, “The MNIST Database,” MNIST handwritten digit database. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 30-Dec-2018].

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. Cambridge, MA: The MIT Press, 2017.

[23] O. Russakovsky et al., “ImageNet Large Scale Visual Recognition Challenge,” International Journal of Computer Vision, vol. 115, no. 3, pp. 211–252, Nov. 2015.

[24] D. H. T. Hien, “A guide to receptive field arithmetic for Convolutional Neural Networks,” *medium.com*, 05-Apr-2017. [Online]. Available: https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807. [Accessed: 30-Dec-2018].

[25] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Unsupervised learning of hierarchical representations with convolutional deep belief networks,” *Communications of the ACM*, vol. 54, no. 10, p. 95, Jan. 2011.

[26] “Convolution arithmetic tutorial,” *deeplearning.net*. [Online]. Available: http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html. [Accessed: 30-Dec-2018].

[27] *CS231n Convolutional Neural Networks for Visual Recognition*. [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed: 30-Dec-2018].

[28] A. Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks. [Online]. Available: https://karpathy.github.io/2015/05/21/rnn-effectiveness/. [Accessed: 01-Jan-2019].

[29] R. Gandhi, “Introduction to Sequence Models - RNN, Bidirectional RNN, LSTM, GRU,” *Towards Data Science*, 26-Jun-2018. [Online]. Available: https://towardsdatascience.com/introduction-to-sequence-models-rnn-bidirectional-rnn-lstm-gru-73927ec9df15. [Accessed: 01-Jan-2019].

[30] M. Schuster and K. Paliwal, “Bidirectional recurrent neural networks,” IEEE Transactions on Signal Processing, vol. 45, no. 11, pp. 2673–2681, 1997.

[31] C. Szegedy, S. Ioffe, V. Vanhoucke and A. A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” *AAAI*, vol. 4, pp. 12, 2017

[32] C. Szegedy et al., “Going deeper with convolutions,” *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[33] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[34] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[35] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[36] “Cloud Speech-to-Text - Speech Recognition | Cloud Speech-to-Text API | Google Cloud,” *Google*. [Online]. Available: https://cloud.google.com/speech-to-text/. [Accessed: 03-Jan-2019].

[37] “Cloud Translation API - Dynamic Translation | Translation API | Google Cloud,” *Google*. [Online]. Available: https://cloud.google.com/translate/. [Accessed: 03-Jan-2019].

[38] “Cloud Text-to-Speech - Speech Synthesis | Cloud Text-to-Speech API | Google Cloud,” *Google*. [Online]. Available: https://cloud.google.com/text-to-speech/. [Accessed: 03-Jan-2019].

[39] I. Hadzic, “ibro45/Language-Identification-Speech,” *GitHub*, 22-Jan-2018. [Online]. Available: https://github.com/ibro45/Language-Identification-Speech. [Accessed: 07-Jan-2019].

[40] I. Hadzic, “ibro45/Speech-to-Speech-Translator,” *GitHub*, 06-Jan-2019. [Online]. Available: https://github.com/ibro45/Speech-to-Speech-Translator. [Accessed: 07-Jan-2019].

9. Table of Figures

Figure 1: Waveform and spectrogram representation of a male person saying “PloS Biology” [11] 5
Figure 2: Example of spectrograms [author’s work] 5
Figure 3: Bottom four waves make up the top waveform [14] 6
Figure 4 Waveform and the output of the Fourier transform [16] 6
Figure 5: Sampling a signal [17] 7
Figure 6: Applying a windows function on a signal [18] 8
Figure 7 Tensorflow architecture [19] 9
Figure 8: Tensorflow code for creating, training and evaluating a neural network on MNIST dataset [7] 10
Figure 9: Receptive field [24] 12
Figure 10: Examples of learned feature maps [25] 12
Figure 11: Fully connected neurons (left) and sparsely connected neurons (right) [22] 12
Figure 12: Zero padding [20] 13
Figure 13: Stride with step size of 2 [26] 13
Figure 14: Max pooling with 2x2 filter and stride 2 [27] 14
Figure 15: Recurrent neuron (left) and recurrent neuron unrolled through time (right) [29] . 15
Figure 16: Types of RNNs, from left to right, one to many, many to one, many to many (encoder-decoder) and simple many to many [28] 15
Figure 17: Bidirectional RNN [29] 16
Figure 18: Inception-Resnet v2 architecture (left) and its stem (right) [31] 17
Figure 19: Residual learning [34] 18
Figure 20: Residual Inception modules A (left), B (right), C (bottom) [31] 19
Figure 21: Reduction modules A (top) and B (bottom) [31] 20
Figure 22: LSTM [20] 21
Figure 23: Inception-Resnet-LSTM CRNN [author’s work] 22
Figure 24: Output of the network’s training [author's work] 23
Figure 25: Confusion matrix; output after the evaluation of the model (left) and a heatmap of it (right) 24