BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

INTERPRETATION OF NEURAL NETWORKS IN SPEECH PROCESSING

INTERPRETACE NEURONOVÝCH SÍTÍ VE ZPRACOVÁNÍ ŘEČI

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MAREK SARVAŠ**
AUTOR PRÁCE

SUPERVISOR **Ing. KATEŘINA ŽMOLÍKOVÁ**
VEDOUCÍ PRÁCE

BRNO 2021

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2020/2021

Bachelor's Thesis Specification

Student: **Sarvaš Marek**
Programme: Information Technology
Title: **Interpretability of Neural Networks in Speech Processing**
Category: Speech and Natural Language Processing

Assignment:

1. Get acquainted with methods of neural network interpretation, such as Layerwise relevance propagation.
2. Get acquainted with application of neural network to speech processing tasks, such as gender classification.
3. Apply selected interpretability method to a speech processing task. Aim to replicate results available in literature.
4. Extend the experiments by e.g. using additional methods, or application to a different speech processing task.
5. Analyze the obtained results. Discuss the problems and potential extensions.

Recommended literature:

- Samek, Wojciech, et al. "Toward Interpretable Machine Learning: Transparent Deep Neural Networks and Beyond." *arXiv preprint arXiv:2003.07631* (2020).
- dle doporučení vedoucí

Requirements for the first semester:

- Items 1 to 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Žmolíková Kateřina, Ing.**
Head of Department: Černocký Jan, doc. Dr. Ing.
Beginning of work: November 1, 2020
Submission deadline: May 12, 2021
Approval date: October 30, 2020

Abstract

With the growing popularity of deep neural networks, the lack of transparency caused by their black box representation is raising demand for their interpretability. The goal of this thesis is to gain new insights into deep neural networks in speech processing tasks. Specifically, gender classification task on AudioMNIST dataset and speaker classification task on filterbanks from VoxCeleb dataset using convolutional and residual neural network. Layer-wise relevance propagation was used for the interpretation of these neural networks. This method produced heatmaps highlighting features that contributed positively and negatively to the correct classification. As results of interpretation show, classifications were mainly based on lower frequencies in time. In the case of gender classification, I managed to find the model's high dependency on a small number of features. Using obtained information, I created an augmented training set that increased the model's robustness.

Abstrakt

S rastúcou popularitou hlbokých neurónových sietí, nedostatok transparentnosti spôsobenej ich funkciou čiernej skrinky, zvyšuje dopyt po ich interpretácii. Cieľom tejto práce je získať nový pohľad na hlboké neurónové siete v úlohách spracovania reči. Konkrétne klasifikácia pohlavia z AudioMNIST datasetu a klasifikácia rečníka z filter bánk VoxCeleb datasetu s použitím konvolučnej a reziduálnej neurónovej siete. Na interpretáciu týchto neurónových sietí bola použitá metóda propagácie relevancií cez vrstvy. Táto metóda vytvorí tepelnú mapu, ktorá vyznačí príznaky, ktoré prispeli ku správnej klasifikácii pozitívne a ktoré negatívne. Ako výsledky interpretácie ukazujú, klasifikácie boli založené najmä na nižších frekvenciách v reči a čase. V prípade klasifikácie pohlavia sa mi podarilo nájsť vysokú závislosť modelu na veľmi malom počte príznakov. Pomocou získaných informácií som vytvoril rozšírený trénovací set, ktorý zvýšil robustnosť modelu.

Keywords

deep neural networks, convolutional neural networks, speech processing, interpretation of neural networks, Layer-Wise Relevance Propagation

Kľúčové slová

hlboké neuónové siete, konvolučné neurónové siete, spracovanie reči, interpretácia neurónových sietí, Layer-Wise Relevance Propagation

Reference

SARVAŠ, Marek. *Interpretation of neural networks in speech processing.* Brno, 2021. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Kateřina Žmolíková

Rozšírený abstrakt

V dnešnej dobe sú hlboké neurónové siete veľmi rozšírené a používané v rôznych oblastiach aj mimo informačných technológií ako sú napríklad zdravotníctvo alebo doprava. Avšak stále pre nás predstavujú akúsi čiernu skrinku, do ktorej vchádzajú vstupy v podobe dát napr. obrázok a vychádzajú iné dáta ako napr. čo je na obrázku. Tento nedostatok transparentnosti vyvoláva otázky ohľadom spoľahlivosti či dôveryhodnosti týchto neurónových sietí alebo ich odolnosti voči útokom. Z týchto dôvodov je v poslednej dobe zvýšený dopyt po interpretácií hlbokých neurónových sietí s cieľom viac porozumieť ich správaniu a odhaliť na základe akých vstupov robia svoje rozhodnutia.

Pretože pre ľudí je jeden z najlepších a najjednoduchších spôsobov ako niečo vysvetliť vizualizácia, práve metódy, ktoré interpretujú rozhodnutia neurónových sietí pomocou vizualizácie sú zatiaľ najlepšia možnosť. Toto je ľahšie realizovateľné pri modeloch neurónových sietí vytvorených a používaných pre spracovanie obrazu oproti modelom pre spracovanie reči. Tieto metódy dokážu odhaliť nedostatky alebo chyby neurónových sietí, ktoré vznikli napríklad pri trénovaní modelu a môžu byť spôsobené artefaktmi nachádzajúcimi sa v trénovacích dátach. V minulosti boli odhalené modely pre klasifikáciu z obrazu, ktoré vykazovali vysokú presnosť klasifikácie na predpripravených trénovacích a testovacích dátach, avšak správna klasifikácia bola založená práve na artefaktoch alebo príznakoch špecifických pre dané dáta a nie pre klasifikovaný objekt. Takéto modely sa nazývajú "Clever Hans predictors". Príklad týchto modelov je klasifikácia koňa na obrázku na základe vodoznaku alebo rozlíšenie medzi psom huskym a vlkom na základe prítomnosti snehu.

Jedna z metód pre interpretáciu neurónových sietí a metóda použitá v tejto práci je propagácia relevancií cez vrstvy neuronóvej siete (angl. Layer-wise Relevance propagation), ktorá vytvorí tepelnú mapu dát, vstupujúcich do neurónovej siete, zvýrazňujúcu príznaky alebo časti vstupu ktoré sú dôležité pre splnenie danej úlohy. Úvod do neurónových sietí a hlavne rôzne metódy pre ich interpretáciu so zameraním práve na Layer-wise Relevance propagation je popísaný v teoretickej časti na začiatku tejto práce. V nasledujúcich kapitolách sú podrobnejšie popísané použité dátové sady, architektúry neurónových sietí a implementácia propagácie relevancií cez vrstvy týchto neuronóvých sietí. Dátová sada AudioMNIST skladajúca sa z nahrávok 60 ľudí spolu s konvolučnou neurónovou sieťou ktorá má AlexNet architektúru sú použité pre klasifikáciu pohlavia z nahrávky. Pre klasifikáciu rečníka je použitá VoxCeleb dátová sada a reziduálna neurónová sieť.

Natrénovaním AlexNet modelu sa mi podarilo dosiahnuť správnu klasifikáciu pohlavia s 97.83% presnostou. Interpretáciou tohto modelu bolo zistené, že model klasifikuje na základe nízkych frekvencií a, v tomto prípade, len na základe malého množstva príznakov indikujúcich nízku robustnosť modelu. Pri nastavení 0.5% najdôležitejších časovo-frekvenčných rámcov, vzhľadom na vytvorené tepelné mapy, na 0, presnosť modelu klesla na 10.8%. Na základe získaných informácií a tepelných máp som rozšíril pôvodnú trénovaciu sadu a znova natrénoval model. Takto natrénovaný model mal v prípade rovnakého nastavenia 0.5% najdôležitejších časovo-frekvenčných rámcov na 0, presnosť 28.67%. Pri klasifikácii rečníka vytvorené tepelné mapy naznačujú, že klasifikácia je robená opäť na základe nižších frekvencií, tentokrát je však dôležitý aj výskyt príznakov v čase. Tieto heatmapy sa dajú považovať do istej miery dôveryhodné vzhľadom na ich vyhodnotenie metódou "pixel-flipping", kde presnosť siete klesne rýchlejšie, ak sa na 0 nastavujú najdôležitejšie rámce vzhľadom na vytvorené tepelné mapy.

Interpretation of neural networks in speech processing

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. Kateřina Žmolíková. The supplementary information was provided by Ing. Ondřej Glembek, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Marek Sarvaš
May 8, 2021

Acknowledgements

I would like to thank my supervisor Ing. Kateřina Žmolíková for her valuable advice, guidance and patience in improving this work. I would like to thank Ing. Ondřej Glembek, Ph.D. for providing ResNet model and dataset needed for experiments. Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

Contents

Chapter 1

Introduction

Deep neural networks are, nowadays, heavily used as state-of-the-art solutions to problems like image, audio processing, or natural language understanding. Yet, they still represent a black box where input comes into the neural network and prediction comes out, but inner decision-making remains hidden. By analyzing some high-performing models trained for image classification, discoveries showed that predictions were dependant on artifacts such as image watermark[19] or background[31]. Even though these models have high accuracy of predicting ground truth on train or test datasets, the reasons for these predictions are considered wrong. Such problems of the models are hard to uncover on limited datasets and end up revealed after a while, if at all. As demand for explainable neural networks is rising, more discoveries and experiments are made. Because the easiest way to explain and understand something is through visualization, interpretation of image classification models can be easily understood.

This thesis aims to bring more insight into how are deep neural network models making their predictions in selected audio signal classification tasks. The first task is a gender classification from speech recording processed as a spectrogram using a convolution neural network with AlexNet architecture, following previous work on this topic presented by S. Becker et al. [4], and Samek et al. [33]. The second is speaker ID classification extending previously done experiments for a more complex audio classification task using a residual neural network model.

The method chosen for interpretation for selected tasks is Layer-wise relevance propagation. This method creates heatmaps highlighting relevant features in data that have a positive and negative contribution to the correct prediction of the model. This method was chosen because of its efficiency in computing such heatmaps and good human interpretability of these heatmaps.

Chapter 2 describes artificial neural networks (ANN), their training process, and more complex deep neural networks, specifically convolutional neural networks because it is the main type of ANN used in this thesis. Chapter 3 is an introduction to the interpretation of neural networks and provides information about different methods used for interpretation. It describes Layer-wise relevance propagation (LRP) in more depth because it is a method used for neural network interpretation in this thesis. Chapter 4 describes used datasets and neural network architectures. Chapter 5 provides some insight into machine learning libraries and the implementation of crucial parts of LRP computation. Chapter 6 describes experiments with different models, produced heatmaps, and performed experiments.

Chapter 2

Artificial Neural Networks

Artificial Neural Networks (ANNs) are getting more popular in last few decades, especially with increasing computer power. In recent years ANNs have been highly used in a variety of tasks that are simple for humans but difficult for computers, such as image or voice recognition, translation, processing a large amount of data, etc. This chapter briefly describes the concepts of deep feed-forward neural networks used in this thesis.

2.1 Artificial representation of a biological neuron

Artificial Neural Networks represent a group of algorithms inspired by the structure of a biological brain which consists of neurons and connections between them. Biological neurons are cells connected with dendrites used as input (electrical signal) receivers and axons, used for propagation of output to other neurons. Inputs are processed inside of the neuron's cell body and sent further to other neurons through the axon [39].

Although artificial networks are a significantly simplified version of how a brain works and information is processed, the principle remains similar. Artificial neurons are connected through weights representing dendrites [39]. Input data are scaled by weights and summed with bias creating activation energy of a neuron as depicted in Figure 2.1.

In 1958 F. Rosenblatt[32] described Perceptron — a neural network model using only one artificial neuron as described above. If the sum of input values scaled by weights is larger than a selected threshold, the output is one, otherwise zero. Therefore it can only solve binary linear classification problems.

Activation functions in neural networks simulate responses to input in a biological neuron [9]. For a neural network to perform non-linear tasks, an activation function needs to be used in a neuron. There are several activation functions used for different tasks profiting from their advantages. Rectified linear unit (ReLU) is one of the most widely used activation functions in neural networks, used mainly in hidden layers [15].

To make a neural network model the desired function, we perform training first, i.e., updating weights of neurons w.r.t. input data and model output. Model training is performed on a dataset (collection of input data) using a backpropagation algorithm (Section 2.3).

Depending on network architecture there are two main types of how a neural network can learn, based on the provided data [15]. **Supervised learning** is based on training the model with data samples pre-labeled with the correct class. This type of learning is mainly used for regression and classification tasks, where a model is supposed to map inputs to a labeled output. **Unsupervised learning**, on the other hand, is performed with previously

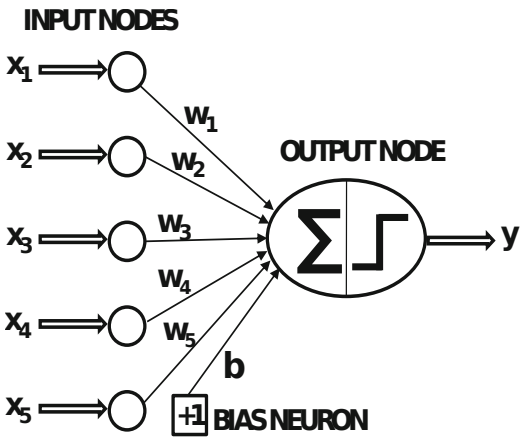

Figure 2.1: Artificial representation of biological neuron. Image taken from [1].

unlabeled data. A goal of such model learning is to identify information and patterns in provided data. This method can sometimes achieve better results than supervised learning in the same tasks [35][38]. In this thesis, all models are trained with supervised learning, specifically for classification problems.

2.1.1 Deep Neural Networks

Information presented in this subsection is obtained from [21, 13, 11]. Deep neural networks are structured as a chain of several different connected functions. These functions represent layers of the network and are composed as follows: input layer, hidden layers, and output layer. The number of hidden layers determines the depth of the neural network. Utilizing hidden layers that perform non-linear operations on inputs allows to better approximate desired function f. There are different types of deep neural networks based on the information flow or type of used layers. Neural networks where the information flows from the input layer to the output layer are called Feed-forward neural networks. Neural networks extended with connections that feed the model's output to itself are called Recurrent Neural Networks.

The goal of Feed-Forward neural network is to approximate some function f, i.e., map one vector space onto another $y = f(x; \theta)$ by learning parameters θ. Specific types of Feed-forward neural networks are Convolutional neural networks or Residual neural networks. Both convolutional and residual neural networks are used in this thesis. The residual neural networks contain shortcut connections that perform identity mapping and skips some layers. These shortcuts are a solution to the saturation followed by the degradation of accuracy during training of deep neural networks.

2.2 Activation functions

Activation functions in neural networks define the "activity" of a neuron, i.e., the output of the neuron and thus the output of a network. Neural networks (NNs) without activation functions produce their output only as a linear function. Also, a multi-layer neural network that uses only linear activation functions behaves just like a single-layer network and can be simplified into one. Both models, NNs without activation functions and Multi-layer NNs

only with linear activation, represent linear models such as logistic regression and have their limitations. Description of activation functions in this chapter is based on information from [9][36][10].

Although linear models are simple, they perform well only on data that can be separated linearly and do not benefit from a multi-layer architecture as non-linear models do. The simplest example of an activation function is the Binary Step Function (2.1). It is a threshold-based activation function, where the threshold value determines if a neuron is activated (its output is used as input to neurons in the next layer) or not. This significantly narrows Binary Step Function usage to only binary classification. Also, the gradient of this function is zero. Therefore, such a network cannot be trained using back-propagation.

$$f_{bin}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2.1}$$

One of the most common and widely used non-linear activation functions is the Sigmoid function. Sigmoid produces output values in the range $(0, 1)$ and is defined as follows

$$f_{sigmoid}(x) = \frac{1}{1 + e^{-x}} \ , \tag{2.2}$$

where the derivate can be easily computed as

$$f'_{sigmoid}(x) = \frac{e^{-x}}{(1 + e^{-x})^2} \ , \tag{2.3}$$

which results in its broad usage in shallow neural networks with some significant disadvantage — **Vanishing Gradient problem** [43] [10]. This problem is caused by saturation during the training process, specifically in regions where $f(x)$ approaches 0 or 1, where the gradient approaches zero. This results in minor to none output signal transmitted, therefore weights of first layers are ineffectively updated.

Another activation function similar to the sigmoid is the Hyperbolic Tangent function also called the Tanh function. Unlike the sigmoid, tanh is symmetric around the origin and produces a value in the range of $(-1,\ 1)$ and is defined as follows

$$f_{tanh}(x) = 2f_{sigmoid}(2x) - 1 \ , \tag{2.4}$$

where $f_{sigmoid}$ is from Equation 2.2. Tanh is preferred over sigmoid because it has a steeper gradient that converges faster and has lower classification error. However, computing the derivative is more difficult for the Tanh function than for Sigmoid and Tanh also suffers from the vanishing gradient.

However, the most popular activation function in deep neural networks is the ReLU function and its optimizations. ReLU stands for Rectified Linear Unit and is defined as

$$f_{relu}(x) = \max(0, x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \ . \tag{2.5}$$

ReLU solves the vanishing gradient problem because the derivative is constant 1 for numbers greater than zero, the derivative function is defined as

$$f'_{relu}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2.6}$$

The absence of exponential functions during computation makes the usage of ReLU more efficient and cheaper compared to Sigmoid or Tanh. Another improvement achieved by using the ReLU function is that not all of the neurons are activated at once. Unlike models utilizing Sigmoid or Tanh functions where all neurons are activated at the same time, with ReLU artificial networks can function a bit more like the biological neural network in the brain, where only a small fraction of neurons are activated simultaneously. This boosts the efficiency in learning by allowing the model to acquire sparse activations in case of input being lower than zero. On the other hand, when input is ≥ 0, the model can obtain a large number of features from data provided during training [10].

The main downside of the basic ReLU function is its left side saturation since the derivate constant is zero when $x < 0$, causing some neurons to become permanently deactivated. Weights of the dead neurons will no longer be updated during the training, which has a negative effect on a whole deep neural network. In order to eliminate the dying ReLU problem a modified version of ReLU called Leaky ReLU(LReLU) is used, comparison shown in Figure 2.2. The solution lies in a small constant such as 0.01 that determines the slope of the function for negative values. Leaky ReLU gradient for inputs < 0 is a small constant and not zero, thus no neuron can be permanently deactivated, potentially creating a dead part of the deep neural network [10][43].

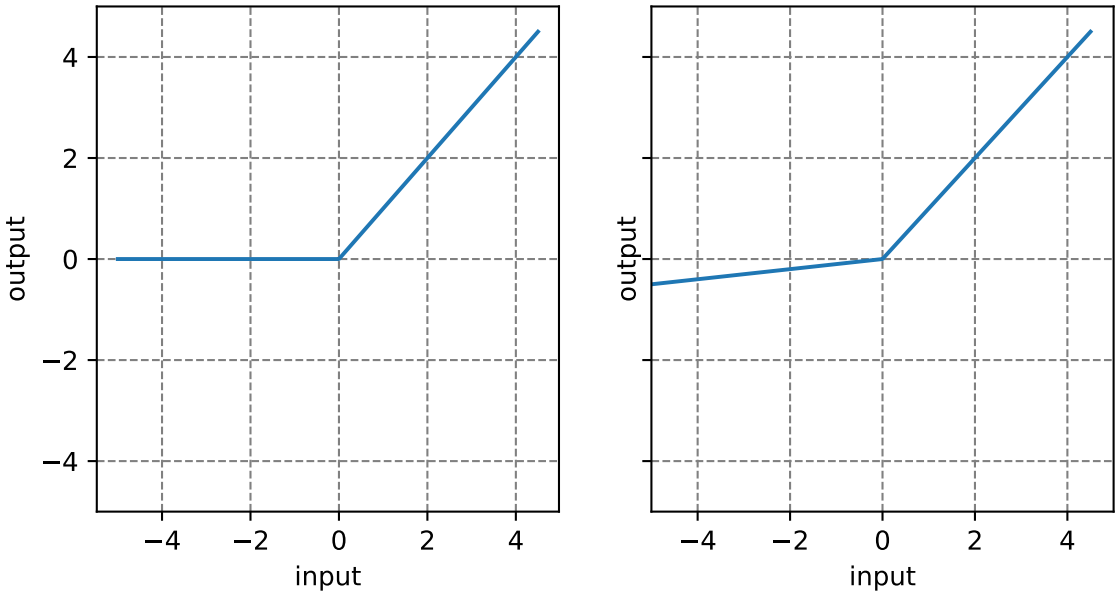

Figure 2.2: Comparison of ReLU(left) and LReLU(right) activation function with slope coefficient of 0.01

Another variation of ReLU is Parametric Rectified Linear Unit (PReLU). In this case, the parameter for the left side of the function for inputs < 0 is learned during training unlike in LReLU where the parameter is a constant given in advance.

2.3 Neural network training process

Training a neural network, also called learning of the neural network, is a process that aims to make the neural network model the desired function, i.e. to minimize the error

between the neural network output and dataset targets, by updating the values of the model parameters. This process is performed in two phases, forward and backward.

In the **forward phase**, input data from a training dataset are fed into a neural network creating a computational graph across the network as the data flows from an input layer towards an output layer (assuming the Feed-forward neural network architecture), using current weight values. The output of the neural network from the forward phase is used to compute an error of network w.r.t. observed target from a dataset. In the **backward phase**, the gradient of the error function is computed and weights are updated (e.g. using stochastic gradient descent). The most used algorithm to compute gradient in the backward pass is the back-propagation. Paired with a learning algorithm such as gradient descent, it allows more simple and efficient learning in comparison to finding the best weights by brute force. Especially in multi-layer models, the error of a model is not a simple function of its weights [11][29].

Loss function

One of the optimization steps in neural network training is evaluating how far the prediction is from the correct value presented in a dataset, i.e., error of the set of weights in a model. The model is evaluated with a function called the objective function. Usually, in neural network models, the aim is to minimize the function. In that case, the objective function is called the loss function, also referred to as the cost function. The product of the loss function, evaluating how far off the prediction of the network is, is called "loss" or "cost". Loss functions can be divided into two groups, for regression and classification problems, based on the type of task a neural network is designed for [1][6].

In regression problems, neural networks aim to approximate a mapping function with numerical or continuous output. For regression problems, some of the basic loss functions are the mean absolute error (MAE), or the most used, mean squared error (MSE)

$$\frac{1}{n}\sum_{n=1}^{N}(t_n - y_n)^2 \ , \tag{2.7}$$

where t_n is the true value, y_n is the predicted value, and N is the number of data points. However, mean or least squared type errors can lead to a solution highly dependent on a small number of edge points. These points are also called outliers and have a lot higher values than the rest. Such errors are prone to incorrectly labeled data and can be solved by using more robust loss functions [41][6].

Classification problems on the other hand aim to approximate mapping function with discrete output usually as positive integers representing different classes or labels. However, the output can be a continuous value when predicting a probability. This probability is often interpreted as the likelihood that given input belongs in the predicted class. Mostly used loss function for classification problems is the cross-entropy [5]

$$L(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w}) \ , \tag{2.8}$$

because it computes the error between probability distributions [7][6]. In this case, a training set is composed of a set of input vectors $\{x_n\}$, where $n = 1, \ldots, N$ and a set of corresponding target vectors $\{t_k\} \in \{0, 1\}$. K represents the number of classes and y_k is the output of a network, where $\mathbf{w}$ is a learnable parameter.

Back-propagation

The back-propagation algorithm can perform an inexpensive computation of the gradient. This computation is performed from the output layer to the input layer during the backward phase of the model training. The loss function is, in this phase, used to compute the gradient w.r.t. network weights using the chain rule of differential calculus.[1][11].

This algorithm assumes that a neural network has a set of hidden layer inputs $h_1, h_2...h_k$, followed by output o, loss function L, and the weights between two layers h_r and h_{r+1} are $w_{(h_r,h_{r+1})}$. If only one path from h1 to o exists, the gradient can be computed as follows:

$$\frac{\partial L}{\partial w_{(h_{e-1},h_r)}} = \frac{\partial L}{\partial o} \cdot \left[\frac{\partial o}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right] \frac{\partial h_r}{\partial w_{(h_{e-1},h_r)}} \qquad \forall r \in 1...k \ . \tag{2.9}$$

In a multi-layer neural network, the number of these paths grows exponentially. So many paths can seem to be difficult and computationally demanding to solve. However, the computational graph of a neural network is acyclic, and the chain rule can be computed recursively from the layer closest to the output o using dynamic programming. Therefore, the expression for computing gradient for a set of paths P is generalized equation1 described as:

$$\frac{\partial L}{\partial w_{(h_{e-1},h_r)}} = \frac{\partial L}{\partial o} \cdot \left[\sum_{[h_r,h_{r+1},...h_k,o] \in P} \frac{\partial o}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right] \frac{\partial h_r}{\partial w_{(h_{e-1},h_r)}} \ . \tag{2.10}$$

The previous information about computing the gradient using the chain rule and respective equations were acquired and are described in more detail in [1].

2.4 Convolutional Neural Networks

Convolutional Neural Network or CNN is a special type of Feed-forward model with state-of-the-art performance in tasks focusing on pattern recognition such as image or voice recognition. To process more complex data such as image data and lower the computational complexity CNNs utilize convolutional and pooling layers. A big improvement over classic neural networks lies in the reduced number of learnable parameters using convolutional layers, which results in an efficiency boost when computing an output or training the model. Another advantage is the presence of Equivariance and Invariance to the translation of input features [11]. Equivariance means that output changes as input changes, allowing detecting edges and shapes in different places through the image, or in time-series data showing where features are present in time. This is achieved using shared weights in convolution. Invariance, on the other hand, reduces the importance of the precise location of features, when it does not matter whether a detected object is on the left or right side of an image. Pooling layers allow the CNN to be invariant to some translations of the input.

The architecture of a Convolutional neural network usually consists of an input layer, convolutional layers alternating with pooling layers implementing, for example, the max-pooling method shown in Figure 2.3. The neural network is completed by fully connected layers producing scores for classification. The convolutional layer produces activations that are used as inputs to the non-linear activation function, such as ReLU. Subsequently pooling layer performs down-sampling of the output of activation functions.

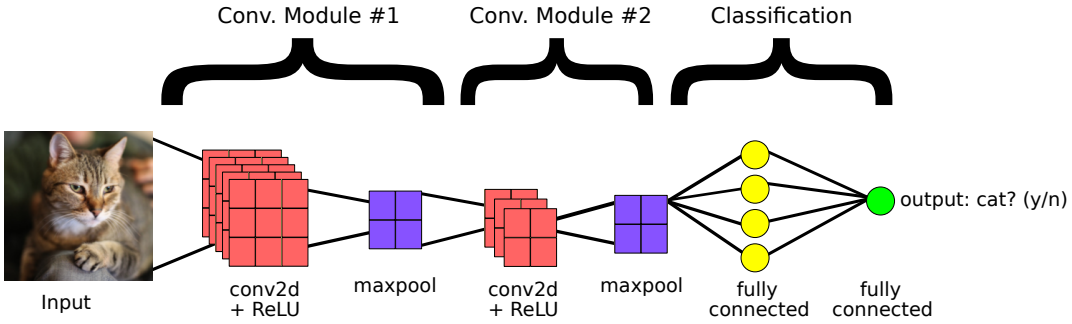

Figure 2.3: Convolution Neural Network basic architecture. Taken from [12].

Convolutional layer

The convolution layer plays a significant role in how the Convolutional neural networks work and their success in solving tasks with grid-like data topology. Convolution can be defined as an operation of two functions producing a third function and it is given as

$$s(t) = \int x(a)w(t-a)\ \mathrm{d}\,a\ , \tag{2.11}$$

denoted with an asterisk

$$s(t) = (x * w)(t)\ , \tag{2.12}$$

where x is an input function and w is a weighting function called the kernel. Because data in the computer are processed as discrete and convolution is usually used over more than a single dimension, it can be defined as discrete convolution with two-dimensional input and kernel as follows

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n). \tag{2.13}$$

This form of the equation is achieved because of commutative property, by flipping the kernel relatively to the input. In CNNs convolution is often implemented as **cross-correlation** achieving the same results without the need of flipping the kernel

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n). \tag{2.14}$$

Convolutional Networks trained using convolution would learn the same values of parameters, but they would be flipped [11].

Unlike the classic ANNs, where the relation between each input and output unit is determined by a specific parameter, in CNNs, kernels allow detecting features more effectively and significantly reduce the number of stored parameters. This is called **sparse connectivity** because a single input unit does not connect to all output units but only to a few based on the kernel size and vice versa the output is not affected by all input units. For example, a neural network layer for processing an RGB-colored image of size 64x64x3 would have 12288 parameters, whereas using a kernel of size 6x6x3 produces only 108 parameters.[28] In the convolutional layer, kernels convolve along the input producing activation maps, and thus each weight value of kernel is used on every input unit. This is referred to as **weight**

sharing, and it is another characteristic present in the convolutional layer that increases the effectivity of CNNs over ANNs. This is based on the hypothesis that one set of learned features can be present in multiple regions in the input, and therefore it is redundant to learn the set of features more than once.

Pooling layer

The purpose of the pooling layer is to downsample the output of the convolutional layer. The reduction in spatial dimensions is achieved by replacing the input with a statistic of neighboring input units in a particular region. In this thesis, the max-pooling function, shown in Figure 2.4, will be used. It downsamples the input by taking only the max value in the neighborhood of a particular size. The size of the max-pool kernel should not be larger than 3 due to the destructive effect of the max-pooling layer [28].

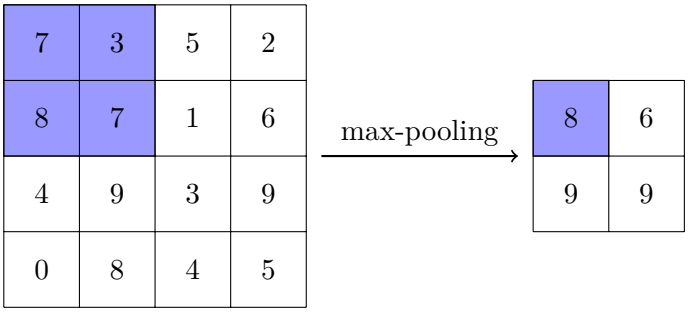

Figure 2.4: Max-pool function with kernel size 2 and stride 2

There are other popular pooling functions such as an average of a neighborhood or a weighted average of a neighborhood, where weights are based on the distance of the input unit from the center of the kernel. In addition to decreasing the number of the parameters, the pooling layer allows the network to be invariant to some translations made in small regions [11].

Chapter 3

Understanding Neural Networks decision making

Artificial neural networks (ANNs) as part of Artificial Intelligence (A.I.) is state-of-the-art technology with broad use in various industries such as information technology, engineering, e.g. self-driving cars, medicine and others. It increases the productivity and capability of these industries to produce discoveries, technologies, or products that otherwise would either not have been discovered or they would have taken significantly more time. ANNs (mentioned in Chapter 2) are capable of performing tasks difficult if not impossible to solve with other programming approaches. Tasks such as image or voice recognition, natural language understanding, or creating something new. This chapter describes what are the ANNs learning, what they "see", or what has a significant impact on their functioning, as well as methods for obtaining this information, i.e. interpreting the ANNs.

3.1 Interpretable and explainable deep neural networks

Even though ANNs are nowadays heavily used, they are still a black box in behavior and decision-making. Lack of transparency of the neural networks' decision-making process and learned patterns may lead to a problem, where if something goes wrong it is hard to say what exactly. Another vulnerability is adversarial attacks against neural networks in speech and image recognition and/or classification. Besides, the lack of transparency of these models increases distrust in neural networks' decisions and their accuracy. The lack of trust, in this case, is legitimate as, for example, deep neural network models' decisions and performance in image classification are extremely good on train and validation datasets, but they may fail in real-life applications. Some neural networks, for example, in image recognition, may perform very well on validation datasets predicting correct outputs but, their predictions are based on artifacts in images such as background or a copyright watermark. Such neural networks are called *Clever Hans predictors*, and an example of such predictions could be a classification of a horse[19] or garbage truck[33], shown in Figure 3.1, based on the watermark present in the image. Another example presented in the [31] shows that a logistic regression classifier distinguished husky and wolf based on snow in the background. These flaws could remain, as in the horse classification, unnoticed for a very long time. Because of the mentioned cases, sometimes it is hard to verify deep neural network models' credibility outside prepared datasets. Methods for explaining these models are researched and developed to provide information about models and their potential flaws.

Figure 3.1: Image (right) and its heatmap of pixels relevant to the model prediction. This heatmap uncovers a Clever Hans predictor, because as can be seen, the logo is highlighted.(Taken from [33])

According to the [33], at first, there were attempts to explain predictions of machine learning models on a global scale by verifying that the output function of a model produces high values only for correct targets. These approaches did not shed any light on what features are important for the prediction. Therefore, methods based on the idea of Pixel-wise decomposition[20] become popular. These methods aim to produce an output that determines how relevant to the model is each pixel. Methods producing heatmaps, which show the contribution of each pixel, are described and compared in this chapter.

Explaining deep neural networks comes with three main difficulties as the models are more complex [33]. The complexity comes from the number of layers that perform linear and non-linear transformations on the input. In such networks of layers some neurons are activated by the small fraction of data points, whereas other neurons are activated more globally. Thus the output of the neural networks is affected by global as well as local effects in the input. The second difficulty comes from the presence of a *shattered gradient*[3] effect in ReLU neural networks with higher depth, where the gradient becomes more noisy. This can cause problems in explanation methods that depend on the usage of the model's gradient such as sensitivity analysis or simple Taylor decomposition[24]. The last difficulty is finding a reference point as the base of the explanation. The reference point is some root point, which is not present in actual data, that some methods use to compute an explanation. For example, the output can change rapidly based on the reference point, but the reference point itself does not carry any significant information for further interpretation.

Interpretation of speech classification

Applying different explanation methods in image classification revealed new information about neural network models, for example, uncovering Clever Hans predictors. With the success of these methods, they are gradually starting to be used in other domains. Here we give an example of speech classification, specifically, predicting gender from audio record-

ings. Because one of the best ways, how to interpret neural networks is through visualization, interpretation of audio signals can be more challenging than image interpretation. To gain new insight into speech classification, several experiments were proposed in [4] and [33]. The experiments were done on raw waveforms and audio spectrograms. Layer-wise relevance propagation (Section 3.2) was chosen as an explanation method for used CNN models. In both cases, raw waveforms and audio spectrograms, LRP highlighted features based on their contribution to the prediction. Blue features have negative relevance on the prediction, whereas red features have positive relevance towards correct prediction.

Raw waveform explanation is presented in Figure 3.2, showing that the model's prediction was based on the outer hull[33][4]. However, this information is hard to interpret for an observer.

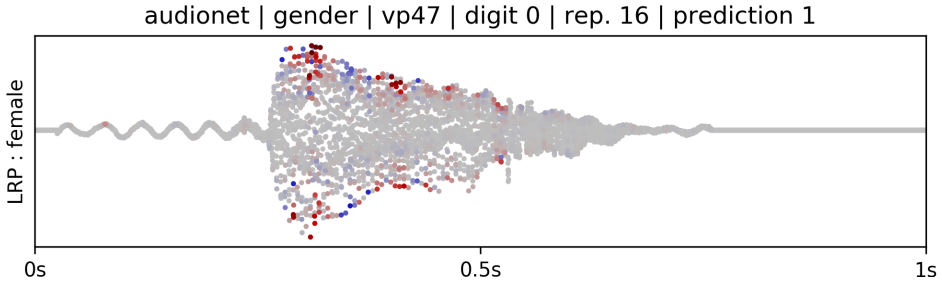

Figure 3.2: Explanation of audio signal based on raw waveform acquired by LRP. Taken from [33]

To raise interpretability for people observing the results, the same method (LRP) was used for a second model that was trained and explained on spectrograms. Spectrograms provided more information about the model and revealed that gender predictions depended mainly on the lowest fundamental frequencies and immediate harmonics[42] (fig. 3.3).

3.2 Layer-wise relevance propagation

Layer-wise relevance propagation (LRP) belongs to a group of backward propagation techniques for explaining deep neural networks utilizing their layered structure. These techniques scale better when used on complex deep neural networks than simple gradient-based methods [24]. Considering a deep neural network as a series of connected layers as in [33]:

$$f(\boldsymbol{x}) = f_L \circ \cdots \circ f_1(\boldsymbol{x}) \ , \tag{3.1}$$

where $\boldsymbol{x}$ is the input of the network and f_l if the function performed by lth layer in the network. LRP computes activation scores in forward pass and subsequently propagates the output score $f(x)$ in backward direction towards the input layer using propagation rules[24] (subsection 3.2.1) as shown in Figure 3.4.

The propagation process is conservative analogous to Kirchhoff's current law in electrical circuits [34]. In neural networks, this means that all activation energy or relevance (in backward propagation) flowing into the neuron has to flow out of the neuron, i.e. be redistributed into the lower layer. Conservation property for neuron k is described in [24] as:

$$\sum_j R_{j \leftarrow k} = R_k \ , \tag{3.2}$$

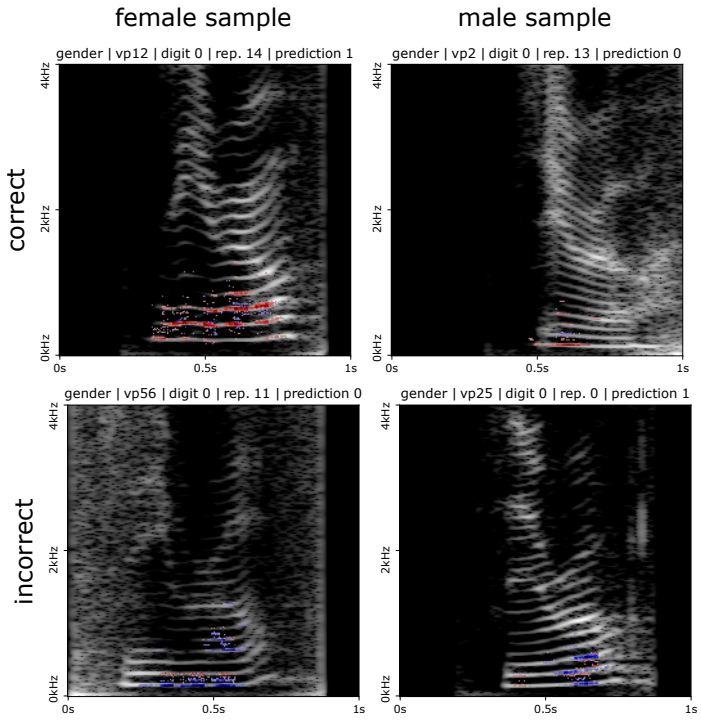

Figure 3.3: Explanation heatmaps of audio signal based on spectrogram acquired by LRP. Taken from [33].

where j and k are indices for neurons of two successive layers and R_k is the relevance of a neuron k at the upper layer. $R_{j \leftarrow k}$ represents redistributed share of relevance R_k into the neuron j in the lower layer. Similarly, the relevance of neuron j in the lower layer is the sum of the relevance propagated from an upper layer:

$$R_j = \sum_k R_{j \leftarrow k} \ . \tag{3.3}$$

These relevance values, which are propagated up to the input layer, make the final heatmap. Heatmap represents data points with positive and negative contributions to a model prediction.

3.2.1 Different LRP propagation rules

The information in this subsection is obtained from [34][16][17]. The simplest basic LRP propagation rule denoted as **LRP-0** redistributes relevances of the upper layer in proportion to inputs of given layer

$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_j a_j w_{jk}} R_k \ , \tag{3.4}$$

where R_j is relevance of given layer j and R_k is relevance propagated from previous layer k. Input of the neuron in given layer j is a_j and $w_j k$ is weight connecting layer j with layer k. Although this rule satisfies properties such as $(a_j = 0) \vee (w_{j:} = 0) \implies Rj = 0$, applying only this rule on the whole network produces a similar result as explaining via $Gradient \times Input$. The gradient in deep neural networks can be noisy, therefore, more robust

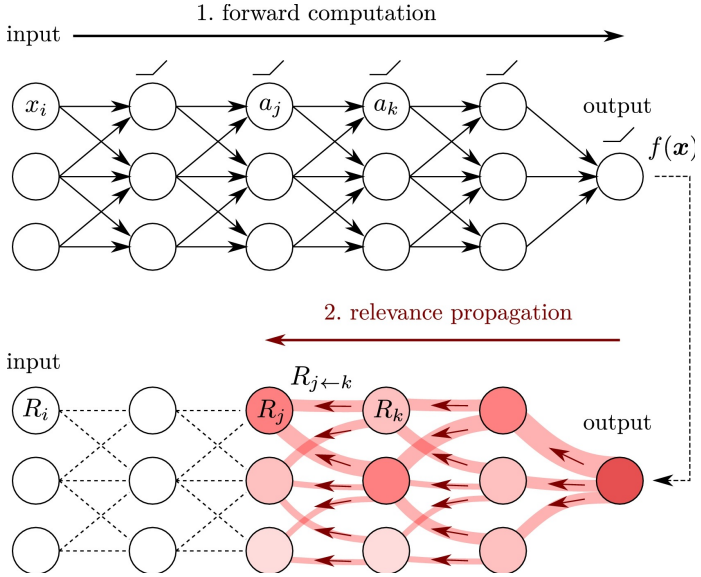

Figure 3.4: Illustration of how each neuron redistributes all relevance (or activation energy) flowing into it from the lower layer into the upper layer. Image taken from [33].

rules are a better option for explaining such networks. In addition to LRP-0, enhanced rules were proposed to increase the explainability of deep neural networks.

The first improvement from basic LRP-0 consists of constant value ϵ added to the denominator. This improvement is denoted as **LRP-$\boldsymbol{\epsilon}$**[17]:

$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk} + \epsilon \cdot sign(\sum_{0,j} a_j w_{jk})} R_k \ . \tag{3.5}$$

The addition of ϵ causes small or contradictory relevances of neuron k to be absorbed. Only the most significant features are propagated as the value of ϵ grows. Explanation utilizing this rule tends to be less noisy with fewer input features presented in a heatmap than explanation made by uniform usage of LRP-0.

Another possible improvement from LRP-0 is a rule denoted as **LRP-$\boldsymbol{\gamma}$** (equation 3.6) is achieved by disproportionately favoring the positive contribution of relevances.

$$R_j = \sum_k \frac{a_j \cdot (w_{jk} + \gamma w^+_{jk})}{\sum_{0,j} a_j \cdot (w_{jk} + \gamma w^+_{jk})} R_k \ . \tag{3.6}$$

The value of γ determines how much are positive relevances favored over negative ones. By limiting the growth of negative and positive relevances, LRP-γ explanation becomes more stable, smooth, and less noisy. Although the **LRP-$\boldsymbol{\alpha\beta}$** rule [20] was originally proposed as a method for treating positive and negative relevances in a disproportion fashion, the equivalent result can be achieved by choosing gamma in LRP-γ.

Even though the above rules provide an enhancement in some way over LRP-0, using any of them uniformly results in suboptimal results. According to [23], every rule has a negative effect in terms of faithfulness and understanding of interpretation when used uniformly. LRP-0 produces a noisy heatmap by highlighting many local artifacts, resulting in unfaithful and inexplicable explanation. LRP-ϵ produces a faithful heatmap by highlighting relevant features, but they are too sparse to be easily interpretable. LRP-γ, on

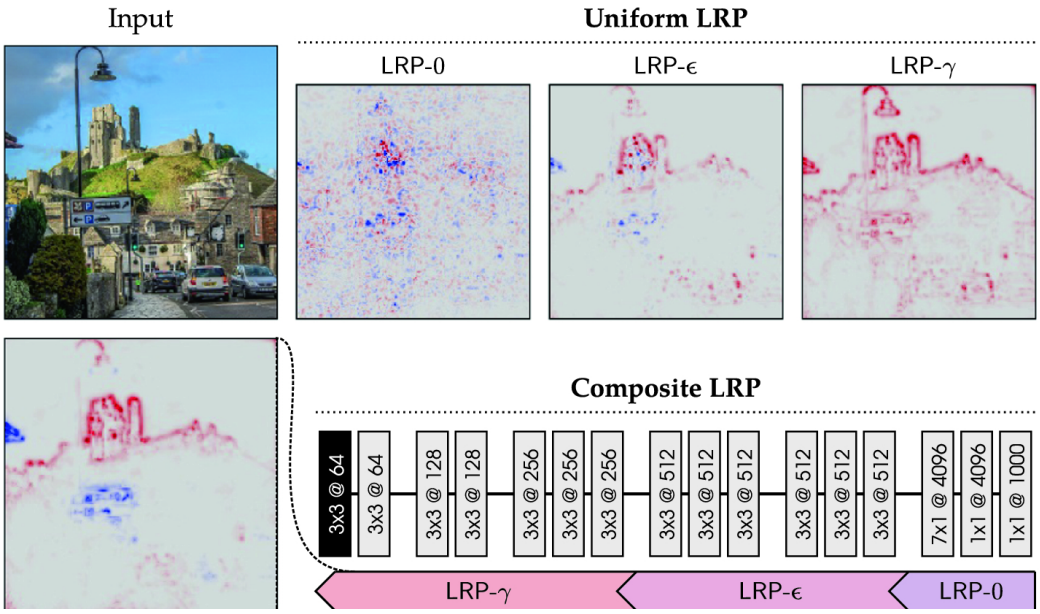

Figure 3.5: Pixel-wise explanation of castle in the image using different LRP rules with parameters $\gamma = 0.25,\ \epsilon = 0.25$. Taken from [23]

the other hand, highlights features more densely than LRP-ϵ but picks unrelated features as relevant, making this method to be considered unfaithful. The best explanation was achieved by combining all three rules in one network using different rules for different parts of the network as shown in the Figure 3.5.

3.2.2 How to implement LRP rules for different neural network layers

Efficiency plays a great role in computing and can save a lot of time. The rules presented in the Subsection 3.2.1 can be generalized into one Equation 3.7, allowing them to be implemented efficiently [23].

$$R_j = \sum_k \frac{a_j \rho(w_{jk})}{\epsilon + \sum_j a_j \rho(w_{jk})} R_k \ . \tag{3.7}$$

Rho represents a copy of a given neural network layer to whose weights and biases was applied a mapping function $\theta \longmapsto \rho(\theta)$ and ϵ is small increment. The propagation of relevance is made in four steps [23]: The first step(1) is a forward pass through a copy of a given layer. The second(2) and fourth(4) steps are division and product, respectively, element-wise operations. The third(3) step is a backward pass of relevance, which can be also computed as a gradient [23].

1: $\forall_k : z_k = \epsilon + \sum_j a_j \rho(w_{jk})$
2: $\forall_k : s_k = R_k / z_k$
3: $\forall_k : c_j = \sum_k \rho(w_{jk}) \cdot s_k$
4: $\forall_k : R_j = a_j c_j$

The information below, about relevance propagation through different neural network layers is obtained from the thesis by Lapuschkin [17]. Relevance propagation implementa-

tions mentioned below are only for layers present in the model architectures used in this thesis. The backpropagation implementations below assume that, in general, the mapping of from neurons x_i at one layer to neuron x_j at the other layer is computed as an equation

$$x_j = \sum_i x_i w_{ij} + b_j \ , \tag{3.8}$$

where x_i is the input neuron of layer i, w_{ij} is the value of weight connecting neuron x_i with neuron x_j, and b_j is bias.

Linear layers that perform a linear transformation on the input using weights, such as fully connected layers and convolutional layers. Relevance propagation in backward pass through these layers is implemented as

$$R_{i\leftarrow j}^{(l,l+1)} = \frac{x_i w_{ji}}{\sum_i x_i w_{ij} + b_j} \cdot R_j^{(l+1)} \ . \tag{3.9}$$

Equation 3.8 can be adapted to implement relevance propagation through **pooling layers**, specifically to this thesis, average- and max-pooling. **Average pooling layers** can be implemented as convolutional layers, where all weights have the value of $w = \frac{1}{n}$, where n is the number of input neurons. **Max pooling layers** implement function

$$x_j = \max_i(x_i) \ , \tag{3.10}$$

where the output neuron x_j is assigned to a single maximal value from all of its input neurons x_i. Therefore, in backward propagation, the incoming relevance value $R_j^{(l+1)}$ is propagated to the single neuron x_i with maximal activation value as follows:

$$R_{i\leftarrow j}^{(l,l+1)} = \begin{cases} R_j^{(l+1)} & \text{if } \arg\max_i(x_i) \\ 0 & \text{else} \end{cases} \ . \tag{3.11}$$

Backward pass through **ReLU** activation functions uses the identity rule[14], $R_i^{(l)} = R_i^{(l+1)}$. **Batch normalization layers** implementing function

$$z = \frac{\mathbf{x} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \gamma + \beta \ , \tag{3.12}$$

where γ and β are parameters learned during a model training. μ_B is the mini-batch mean value, and σ_B^2 is the mini-batch variance, both values are fixed after the training. ϵ is a small constant preventing division by zero. The batch normalization, in a forward pass, can be computed as a sequence of equations:

$$\mathbf{x}' = \mathbf{x} - \mu_B \tag{3.13}$$

$$\mathbf{x}'' = \mathbf{x}' \cdot s \tag{3.14}$$

$$z = \mathbf{x}'' + \beta \tag{3.15}$$

where s is a substitution from the equation, $s = \gamma \cdot (\sigma_B^2 + \epsilon)^{-\frac{1}{2}}$. The backward pass of relevance through equations 3.13 to 3.15 creates an equation [17]

$$R^{(l)} = \frac{\mathbf{x} \odot s \odot R^{(l+1)}}{z} \ , \tag{3.16}$$

where $\odot$ is element-wise multiplication.

3.3 Other methods for neural network interpretation

Beside Layer-wise relevance propagation and its use of the structure of neural networks, methods based on other principles were proposed to gain new insights into neural networks. These methods do not use layers as such but rely on different aspects of neural networks. They aim to explain models using approaches such as gradients in combination with input data, various analyses of model sensitivity to certain input features, and their perturbations. In this section, some of these methods are mentioned, and later in section 3.4 described their advantages and disadvantages.

3.3.1 Occlusion

Another method available to explain neural networks is Occlusion analysis [33][2]. It is a specific type of perturbation analysis, where during neural network analysis, input features or whole patches are being occluded. For example, when explaining models trained for image classification, square regions of the input image are replaced with grey or zero values. The relevance is obtained by measuring the effect of occluded regions on the prediction and accuracy of the explained model. However, the relevance can be computed in two ways, based on the problem the neural network is used for. In terms of prediction, the heatmap is built from scores computed as the difference between functions.

$$R_{x_i} = f_c(\boldsymbol{x}) - f_c(\boldsymbol{x}_{|x_i=0}) \ . \tag{3.17}$$

Regions or features that caused the biggest decrease in prediction accuracy are highlighted in such heatmap, this type of occlusion is also referred to as $Occlusion_{f-diff}$. In terms of explaining classification, it is computed as the difference between probabilities $\boldsymbol{x}$ and perturbed $\boldsymbol{x}_{|x_i=0}$:

$$R_{x_i} = P_c(\boldsymbol{x}) - P_c(\boldsymbol{x}_{|x_i=0}) \ , \tag{3.18}$$

and referred to as $Occlusion_{P-diff}$. Because visual artifacts can occur in heatmaps produced by occlusion of input images, there were proposed enhancements such as inpainting the patches instead of setting them to grey [33].

3.3.2 Gradient based explanations

Integrated Gradients is one of the methods for explaining deep neural networks based on their gradients. Another variant is, for example, SmoothGrad[33]. The Integrated Gradient method utilizes sensitivity of backpropagation methods and implementation invariance of gradients[3]. On the other hand, it suffers from the shattered gradient problem. This problem can be minimized by averaging relevance scores of multiple integrations paths as proposed in the experiments by W. Samek and G. Montavon[33]. Let $f : R^n \to [0, 1]$ be a function representing some deep neural network, $\boldsymbol{x} \in R^n$ networks input, and $\boldsymbol{x'} \in R^n$ networks baseline input, for example, a black image in computer vision. The Integrated Gradient defines relevance scores of features, where these scores are produced by integrating gradients along a straight path from $\boldsymbol{x'}$ to $\boldsymbol{x}$. Integrated Gradient along i^{th} dimension[3] is defined as:

$$R_{x_i} = (x_i - x'_i) \cdot \int_{\alpha=0}^{1} \frac{\partial f(\boldsymbol{x'} + \alpha \cdot (\boldsymbol{x} - \boldsymbol{x'})}{\partial x_i} d\alpha \ . \tag{3.19}$$

This method satisfies *completeness* because the sum of the attributions is equal to the difference between function f with input x and baseline x'. The baseline for most networks can be chosen as $f(\boldsymbol{x'}) \approx 0$ therefore the attributions are propagated among input features[3].

3.4 Requirement for neural network explainability methods

Methods described in Sections 3.2 and 3.3 implement different approaches towards neural network explainability. Therefore, every one of them produces slightly distinct relevance scores and heatmaps. In computer vision, for example, LRP tends to highlight features mostly in favor of positive relevances. The occlusion method highlights important regions in the image. And the integrated gradient highlights relevant pixels but shows more negative relevance in heatmap than LRP, Figure 3.6.

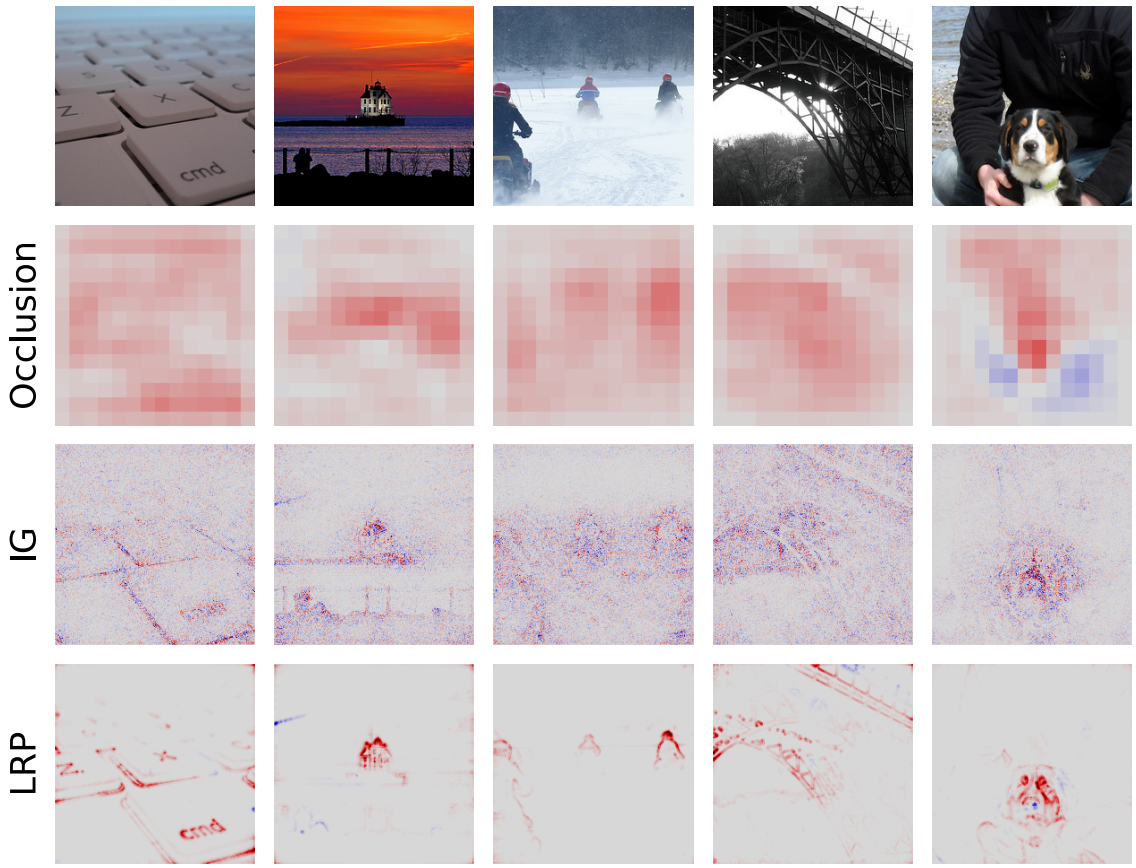

Figure 3.6: Preview of heatmaps obtained by Occlusion, Integrated Gradient and Layer-wise relevance propagation (top to bottom) in correct classified images of 'space bar', 'beacon/-lighthouse', 'snow mobile', 'viaduct', 'greater swiss mountain dog' (left to right). Figure taken from *Transparent Deep Neural Networks and Beyond* [33].

Neural networks are, in general, evaluated by how reliable their predictions are, i.e. how high is the probability that their predictions will be correct. To determine the usefulness of neural networks explanation is more complicated because there is no ground truth in such explanations. Several aspects were proposed to help evaluate if and how big an impact explanation methods have on neural network's performance. Information about the following requirements for explanation methods is obtained from William Swartout and Johanna Moore's conference paper [40] and Transparent Deep Neural Networks and Beyond article [33].

Faithfulness

Faithfulness as a property of methods for neural network explanation is associated with how the explanation is created. An incorrect or confusing explanation of a neural network model is not useful and can provide misleading information about the model, possibly causing more problems than an unexplained model. Explanations must be based on the same knowledge as is the model's decision-making to accurately and faithfully represent its decision structure. *Pixel-flipping* is a method for determining the faithfulness of the explained model. The *Pixel-flipping* method is based on removing the most relevant pixels from an input image and evaluating changes in the model's output. As relevant pixels or features begin to disappear from the input, the model accuracy, i.e. probability of correct prediction, should be decreasing, Figure 3.7.

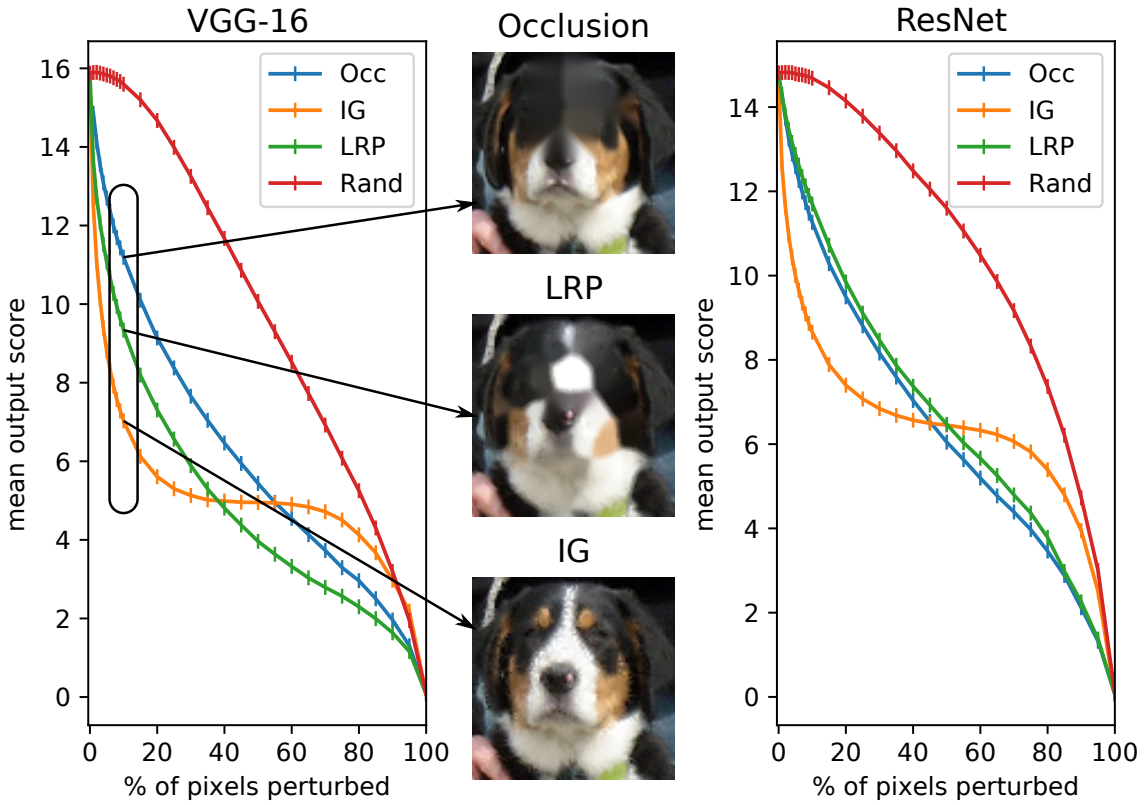

Figure 3.7: Experiment for determining faithfulness of different explanation approaches on image classification model. As we can observe *Integrated Gradient method*(bottom picture of a dog) found pixels on which model depends the most. Although the input image nearly does not change for the human eye, the prediction accuracy drops drastically [33].

Even though faithfulness determines if the explanation highlights relevant and comprehensive features of the model, it does not ensure an easily interpretable explanation for a human observer.

Interpretability

The intention for research and use of interpretation methods is to gain, to some extent, insight and try to understand the black box that neural networks are. For this to be

successful, explanations produced by these methods need to be interpretable to humans. According to Miller [22], "most of the research and practice in this area seems to use the researchers' intuitions of what constitutes a 'good' explanation". It is hard to define what a good explanation is because different people may interpret the same explanation differently, based on their knowledge and capabilities. Miller [22] also highlights findings important for explainable AI. Some of these findings are that:

- referring to causes is, for people, more important than referring to probabilities, or
- people are more likely to ask "why event P happened instead of some event Q"[22] than why event P happened

In [33], the interpretability of different explanation methods for image classification models is measured based on the produced explanation's file size. This comparison shows that occlusion produced the smallest file size, roughly showing where important features are located, therefore, it should be the best for interpretation.

Applicability

Other important characteristics of explanation methods are applicability and runtime. Applicability determines if a method can be applied to a variety of neural network models, including those which are the subject of research, and how easy the implementation of the method is. Runtime determines computing efficiency, how many resources and time the method needs to produce the explanations. According to the results of the comparison between Occlusion analysis, LRP, and Integrated gradients method presented in [33], Occlusion analysis is the easiest to implement and can be obtained for every network. However, it is the slowest among the three. LRP, on the other hand, is the fastest method but assumes that a model has the structure of a neural network consisting of a sequence of layers.

Chapter 4

Used datasets and deep neural network architectures

The usage of LRP for interpretation requires access to the internal structure of a neural network. It is crucial to understand the input data to accurately interpret neural network decisions. This chapter describes datasets used for interpretation of two speech classification models for gender classification using spectrograms (AudioMNIST dataset) and speaker ID classification using audio filter banks (VoxCeleb dataset). Another part of the chapter describes deep neural network models used with these datasets. The AlexNet model is used with AudioMNIST and ResNet with Voxceleb dataset, showing their architecture and produced outputs.

4.1 AudioMNIST audio dataset

To replicate and extend experiments proposed by *S. Becker et al.*[4] the same dataset and model are used. This dataset was originally used for the interpretation of both digit and gender speech classification models. It consists of audio recordings (spoken digits $0-9$) of 60 different speakers of various nationalities and age, where 12 speakers are females and 48 males. Each speaker has 500 recordings, where every digit $(0-9)$ is repeated ten times, producing a total of 30000 audio recordings.

The raw audio samples were recorded with a 48kHz sampling frequency, stored as a *.wav* file. These samples were preprocessed into spectrograms (Figure 4.1) with python script included in the AudioMNIST repository. At first, the recordings were downsampled to 8kHz and zero-padded into 8000-dimensional vectors. Then, spectrograms were created using Short-time Fourier transform with Hann window, 455 samples per segment, and overlapping segments with the size of 420 samples per segment. Produced spectrograms of audio recordings had size of 228×230 ($frequency \times time$). The highest frequency bin and last three time segments were cropped, creating spectrograms with the size of 227×227. The amplitude was converted into decibels using $d = 20 \log_{10} \frac{a}{a_{ref}}$, where d is result in decibels, a is spectrogram amplitude and a_{ref} is reference amplitude ($a_{ref} = \max(a)$).

The preprocessed data were reduced to 24 speakers (12 female and 12 male chosen randomly) and split into three disjoint splits: training, validation, and test split containing 6000, 3000, and 3000 recordings, respectively.

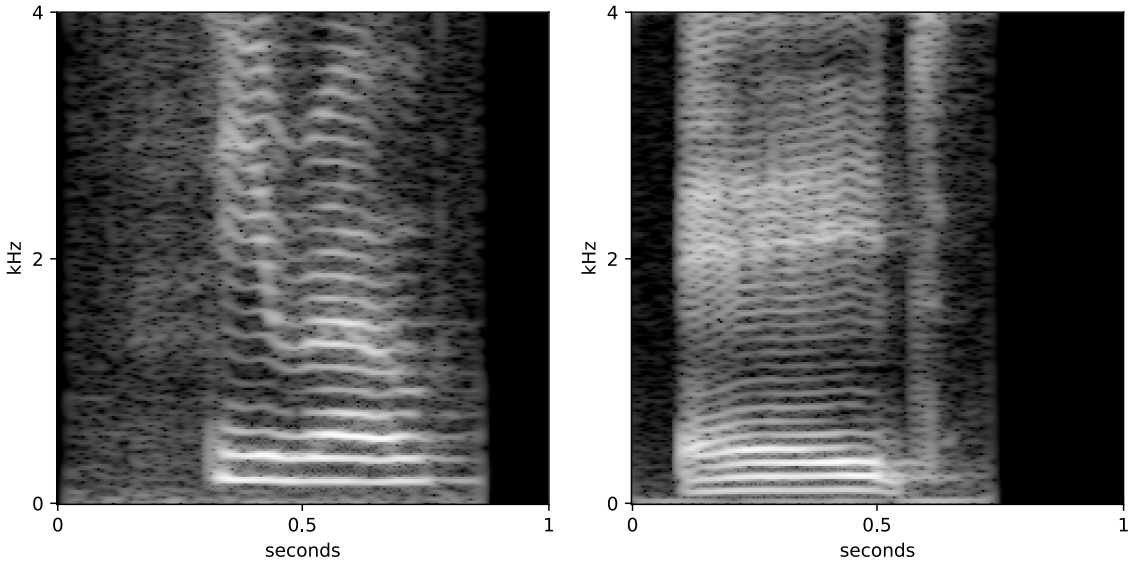

Figure 4.1: Spectrograms of AudioMNIST recordings, female(left) and male(right) speaker.

4.2 VoxCeleb audio dataset

Features from VoxCeleb[25][26][8] audio dataset were used to extend interpretation experiments to a different task such as speaker classification. This dataset was chosen because I obtained a neural network model pre-trained on features from this dataset for a speaker classification task. In particular, the data were from the VoxCeleb2 dev dataset containing

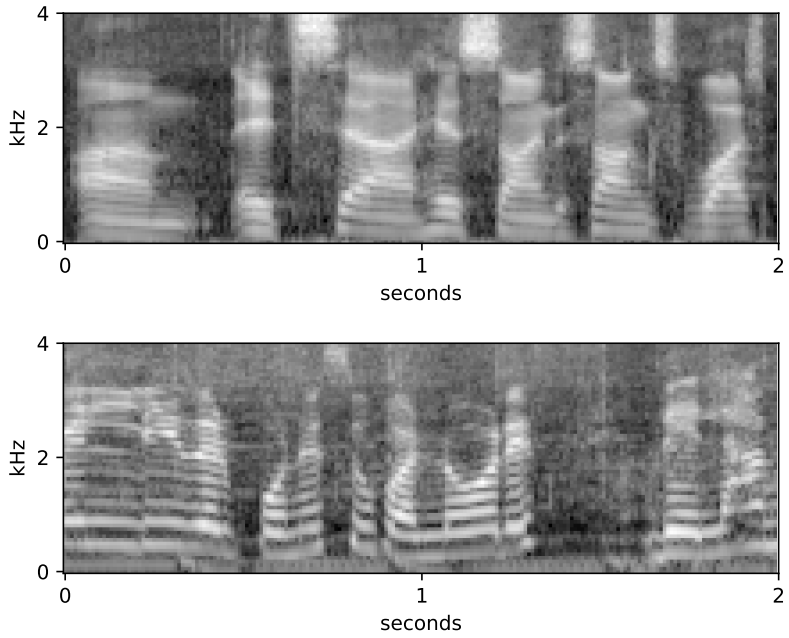

Figure 4.2: Spectrograms of VoxCeleb features augmented with music(top) and noise(bottom).

speech recordings of 5994 individual speakers and its augmentations with music and noise from the MUSAN[37] dataset. Features used for model training and interpretation experiments are 64-dimensional filterbanks showed in Figure 4.2. In every dimension, the mean value is normalized to 0, and each frame represents 25ms of speech with a 15ms overlap. Each file with 64×200 features corresponds to a two-second segment of the recording.

4.3 AlexNet architecture

For the gender classification task, a convolutional neural network model was used, as proposed in the article by S.Ḃecker et al.[4]. The model has AlexNet architecture with adapted parameters for classification from spectrograms (section 4.1). It consists of two main parts: feature extraction block and classification block. The feature extraction block is composed of five convolutional layers, ReLU activation functions, and max-pooling layers. The input data has a size of 227×227 and a single channel as described in the Section 4.1. The five convolutional layers with kernel sizes 11, 5, and 3 respectively, and max-pooling layers down-sample the input (2.4). ReLU activation function (2.2) is very popular and crucial for interpretation with the LRP method. Adaptive average pooling is applied to ensure that size of the feature tensor is 6x6. The output of the average pooling layer is flattened and fed into a classification block. The classification block is composed of three dense layers feeding the output of size $(1, 2)$ into sigmoid activation function. During an evaluation of the model, the prediction is obtained as $y_{\text{gender}} = argmax(\mathbf{out})$, where *out* is output tensor of the model and value of y_{gender} represents gender as follows:

$$y_{\text{gender}} = \begin{cases} male & \text{if } argmax(\mathbf{out}) = 0 \\ female & \text{if } argmax(\mathbf{out}) = 1 \end{cases}$$

4.3.1 Training of the model

AlexNet model was trained for 200 epochs on the 6000 recordings training set described in Section 4.3 with a batch size of 50 recordings, where recordings in each batch were chosen randomly. The learning rate was set to 1e-4 and momentum to 0.9. Binary Cross Entropy was used as a loss function to evaluate the model during training. Stochastic gradient descent was used as an optimizer function for updating the model's weights. After 200 epochs model achieved a gender prediction accuracy of 97.83% on both validation and test sets (described in Section 4.3).

4.4 Speaker ID classification model

Another neural network model chosen for interpretation is trained to classify speakers from an audio recording. This model is a more robust and complex deep neural network than previously used AlexNet. It is based on ResNet34 architecture with some changes to perform well on a designated task. Pretrained model was provided by Phonexia/VUT FIT, with a classification accuracy of 96% on both training and validation datasets. Input data into this network are filterbank features of size 64×200 and a single channel described in Section 4.2.

This architecture is using mainly a combination of 2D convolutional layers (Conv2d) and 2D batch-normalization (BatchNorm2d) layers. Starting with one convolutional and one batch-norm layer, followed by four main parts composed of multiple blocks of Conv2d, BatchNorm2d layers, and a residual connection. The main hidden layers consist of 3, 4,

6, and 3 blocks. Each main layer downsamples the input in half by setting $stride = 2$ in the first convolutional layer of the first block. The blocks consist of two Conv2d and two BatchNorm2d layers, as shown in Figure 4.3 on the right. In addition, the first block in each main layer has a residual connection composed of Conv2d and BatchNorm2d layers, shown in Figure 4.3 on the left. Activation functions used in this architecture were ReLU. Followed by mean and standard deviation pooling, which summarizes the whole utterance into one vector. The network ends with a dense layer for speaker embeddings extraction with $(N, 256)$ output features tensor followed by an 1D batch-normalization layer and a dense layer producing output speaker ID tensor of size $(N, 5994)$, where N represents batch size. The prediction of the model is obtained as $y = argmax(\mathbf{logits})$, where logits represent the value of each speaker.

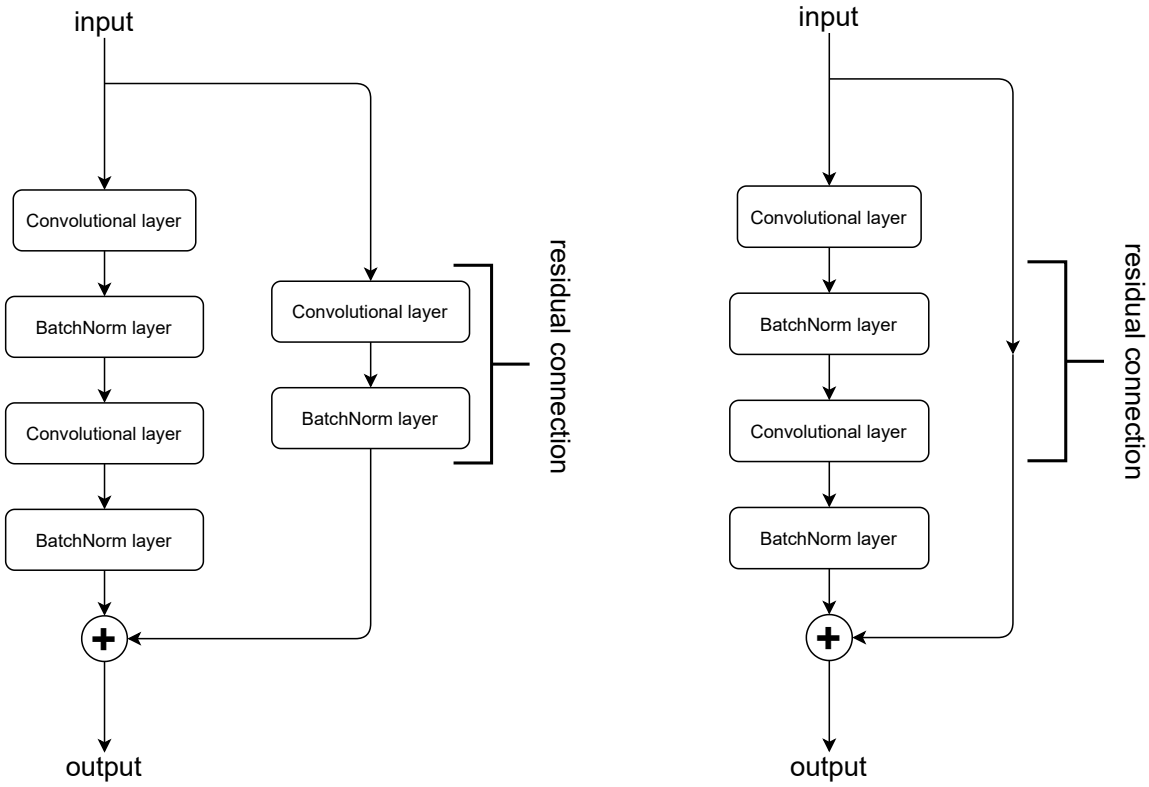

Figure 4.3: This figure shows the architecture of one block in the ResNet model. The first block in the main layer has a convolutional and batch normalization layer in its residual connection(left). Other blocks in the main layer have a residual connection without additional layers in it(right).

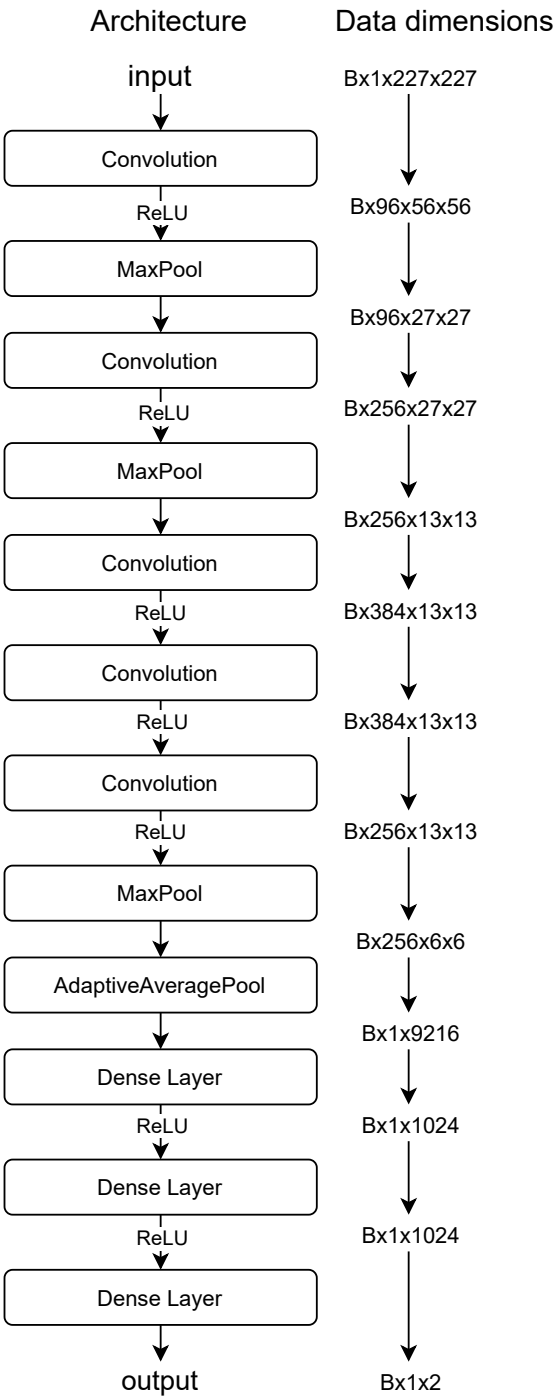

Figure 4.4: AlexNet architecture scheme showing its layers(left) and dimensions of their tensors(right). The First and last size is the dimensions of input and output tensors, respectively. Others are dimensions of the output tensor of the respective layer. B in represents a batch size of a given tensor.

Chapter 5

Solution design and implementation

The Layer-wise relevance propagation can be implemented in different ways depending on neural network architecture and chosen library. This chapter gives a brief overview of different libraries for machine learning, existing LRP implementations, and finally, my implementation used for experiments in this thesis.

5.1 Machine learning libraries

Machine learning libraries described in this section were initially released in 2015 (Keras and Tensorflow) and 2016 (PyTorch). Since then, they have gained on popularity in research, development, real-life applications, and others. They allow easy creation, training, and evaluation of a broad spectrum of machine learning models while also providing some pre-trained models. These are not the only libraries for machine learning but are the most popular.

5.1.1 TensorFlow and Keras

TensorFlow is a free, open-source library for machine learning developed by Google written in Python, C++, and CUDA. Code written in TensorFlow can run on both CPU or GPU, allowing acceleration in computation, especially with high dimensional matrices called tensors which are primary data structures in this library. The big difference between TensorFlow and PyTorch is the implementation of a computational graph. In TensorFlow, the computational graph is static, where the sequence of computation is defined beforehand, allowing the use of placeholders. This approach has great performance, but it is hard to debug. TensorFlow has great production and deployment options in comparison to Pytorch and therefore is very popular among developers.

Keras is an open-source high-level Python API later integrated into TensorFlow. Keras can be used independently of other libraries. However, since it handles only high-level computations, it is convenient to use Keras on top of the other machine learning library that functions as a backend.

5.1.2 PyTorch

PyTorch[30] is also an open-source machine learning library initially released in 2016 and developed by Facebook. It is based on the Torch library and written in Python, C++, and CUDA. Programs written in PyTorch can also run on both CPU and GPU. PyTorch is highly popular in the research field, mainly for its simplicity, flexibility, and it tends to be easier to use than TensorFlow when starting with machine learning. It is a high-performance library that handles low-level computations, has efficient memory usage and great debugging options. It allows to easily build customized neural network models, debug them with, for example, forward and backward hooks, and convert PyTorch tensors into Numpy multi-dimensional arrays. The big difference from TensorFlow is the dynamic computational graph. In PyTorch, the computational graph is built and can be changed during runtime. I chose PyTorch for its simplicity, debugging options, popularity in research, and hooks.

5.2 Existing solutions and proposed solution design

There are existing solutions for interpreting neural networks implementing different methods, mentioned in Chapter 3, to explain various models trained for different tasks. However, only a couple of these projects implement LRP. Each solution takes a different approach to implement explanation methods and models using different Python libraries, like PyTorch, Keras, Caffe, or even NumPy for LRP. In a lot of cases, LRP is implemented as either variations of Python classes with methods computing LRP rules. Or, in the case of PyTorch implementations, whole layers are implemented with a custom forward and backward passes, and LRP is computed in a backward function utilizing PyTorch's dynamic computational graph. These solutions have usually implemented LRP only for a couple of layers depending on the model used for interpretation.Also, all of these implementations were for image classification models.

As I chose PyTorch as a library for my thesis and both models used for the interpretation are also implemented in PyTorch, I decided to utilize easy-to-use forward hooks available in PyTorch in combination with classes computing LRP for each type of layer. I did not want to implement whole layers with forward pass function and backward pass functions as it is already well optimized. In addition to separate classification from LRP computation, I decided to create a class for each type of neural network layer used in the models. Each class implements relevance propagation according to LRP rules and is callable outside the interpreted model. This separation is achieved by storing the weights, inputs, and outputs of layers during a forward pass through the model using registered forward hooks as described in the following Section 5.3.

5.3 Implementation of LRP for AlexNet and ResNet models

The main function for computing LRP is composed of the definition of forward hooks, their registration, forward pass of the model, and loop for computing the LRP through the layers. First, forward hooks are implemented so that for each layer important for LRP (e.g. excluding ReLU) a custom object representing that layer is created with corresponding weights and stored into a list of custom layers ready for LRP as shown in Listing 5.1.

```
def f_hook(module, input, output):
    if isinstance(module, nn.Conv2d):
```

```
        # bias is set to false in ResNet
        layers.append(ConvLayer(input[0], output, type(module),
                    module.weight.data, None, module.kernel_size,
                    module.stride, module.padding, module.groups))
    elif isinstance(module, nn.Linear):
        layers.append(LinearLayer(input[0], output, type(module),
                      module.weight.data, module.bias.data))
```

Listing 5.1: Part of the forward hook function, where convolutional and dense layer are being added into the list for LRP computation in the future.

These hooks are registered before the forward pass of the model. After the forward pass of input data and the creation of custom layer objects, the registered forward hooks are deleted. The relevance tensor is initialized from the model's output tensor and propagated backward through the layers from the output to the input layer as shown in 5.2.

```
    with torch.no_grad():
        layers.reverse()
        R = Rinit
        for layer in layers:
            R = layer.lrp(R)
            if verbose:
                print(f'Current relevance tensor shape: {R.shape}')
    return R
```

Listing 5.2: Computation of LRP through layers, where Rinit is output of the model and R is relevance tensor newly propagated through given layer.

Each type of neural network layer has a corresponding class implementing LRP computation according to the rules and decomposition described in the subsection 3.2.2. Each class has object variables, such as weights input and output tensors of representing layer, needed to compute LRP. LRP implementation of convolutional, linear, and max-pooling layers was inspired by the solution presented in the LRP-toolbox[18] repository but written in PyTorch and further adapted to the layers of AlexNet and ResNet models. The adaptation mainly involves the addition of padding, optional bias, and groups in convolutional layers. Using the LRP method on ResNet architecture required the implementation of batch normalization layers shown in Listing 5.3 and solving relevance propagation through residual connections (Listing 5.4).

```
def lrp(self, R):
        .
        .
        .
    s = torch.true_divide(W_positive,
                          torch.sqrt(self.run_std ** 2 + self.eps))
    Rx = torch.true_divide(self.X * s[..., na, na], self.Y)
    Rx = Rx * R
    return Rx
```

Listing 5.3: Implementation of basic LRP for batch normalization layer. *W_positive* represents learned γ parameter, *self.run_std* is mini-batch mean value, and *self.eps* represents ϵ from Equation 3.12. *self.Y* is output of the layer represented as z in Equation 3.15 and R is relevance tensor propagated from the previous layer.

I implemented a *BasicBlock* class representing blocks of layers with residual connections in the used ResNet model, enabling to backpropagate relevance correctly through these connections. *BasicBlock* class has two lists of layer objects, one for layers of represented part of neural network and the other for layers used in residual connection. When a residual connection is present, relevance from the lower layer is propagated as if there was a linear layer with two inputs and weights set to 1. Then the relevance is propagated further simultaneously, through usual neural network layers, and layers in residual connection. Sum of relevances produced by these two ways is propagated into the upper layers.

```
# compute LRP for hidden layers
for layer in self.layers:
    self.R = layer.lrp(self.R, rule, eps=eps, gamma=gamma)

# compute LRP for residual connection
for layer_short in self.res_connection:
    self.R_residual = layer_short.lrp(self.R_residual, rule,
    eps=eps, gamma=gamma)

return self.R + self.R_residual
```

Listing 5.4: Implementation of LRP computation through residual connection, where *self.layers* is list of hidden layers in the block. Layers inside residual connection are in the *self.res_connection* list, *eps* and *gamma* are constants (ϵ, γ) set for LRP rules. Relevance tensor propagated from previous layer is divided in proportion to output of hidden layers and residual connection into *self.R* (relevance that came from hidden layers) and *self.R_residual* (relevance that came from residual connection)

Chapter 6

Proposed experiments for audio signal interpretation

One of the goals of this thesis is to train and interpret a gender classification model. With the trained model and implemented interpretation method, I try to replicate results made by S. Becker et al. [4]. Further goal is to extend these experiments further based on results and use information obtained by LRP to improve the model. In the second part of the experiments, the challenge is to interpret a more complex model, trained for a more complex task, such as the mentioned ResNet model. This model is trained on a much bigger dataset with various data augmentations making it more robust. Another difference is the input data. While in the case of AlexNet, it is spectrograms, ResNet uses filter banks, which are more difficult to interpret.

6.1 Explanation of audio spectrogram gender classification

I used the previously proposed method — LRP, to find out based on what the AlexNet model is making its decisions. The LRP used for this experiment utilizes LRP-0 to propagate relevance values through linear layers and LRP-eps for hidden and input layers with epsilon value 0.8. This value was chosen based on preliminary experiments. Figure 6.1 shows heatmaps overlayed on top of the corresponding spectrograms of audio recordings. The obtained heatmaps look similar to the ones presented in [4], where the neural network makes decisions based on the lower frequencies.

In Figure 6.2, spectrograms of female and male recording are modified by setting 1% of the bins with the highest positive contribution towards correct prediction to zero w.r.t. LRP heatmaps shown in Figure 6.1. This 1% shows that even though the Figure 6.1 shows relevant bins in higher frequencies, the most relevant bins are located in lower frequencies. In case of male speakers, the model learned to look at parts of spectrograms that were zero-padded during the pre-processing but had a value of -80 in spectrograms. The male spectrograms have padded areas that are usually bigger than those in female spectrograms. In male spectrograms, on average, 75 dimensions out of 227 have padded values (-80). Female spectrograms have 52 out of 227 dimensions, on average, with a padded value (-80). This difference in male and female data may be one of the causes of such a flaw of the model. It is possible that the dependency on the padded area can be reduced by augmentation of the training dataset, e.g. choosing a different approach on how to pad

the data. However, this shows behavior similar to Clever Hans predictors, where a model performs very well but has learned patterns that can be considered wrong.

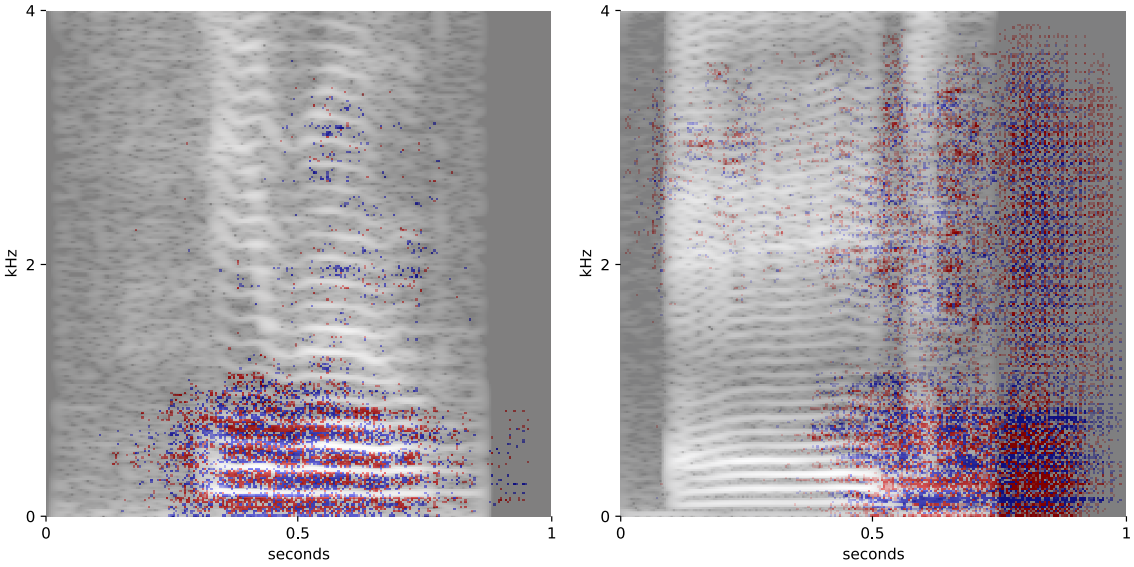

Figure 6.1: Heatmaps produced by LRP on female(left) and male(right) spectrograms. Red represents time-frequency bins with positive relevance scores and blue with negative relevance scores.

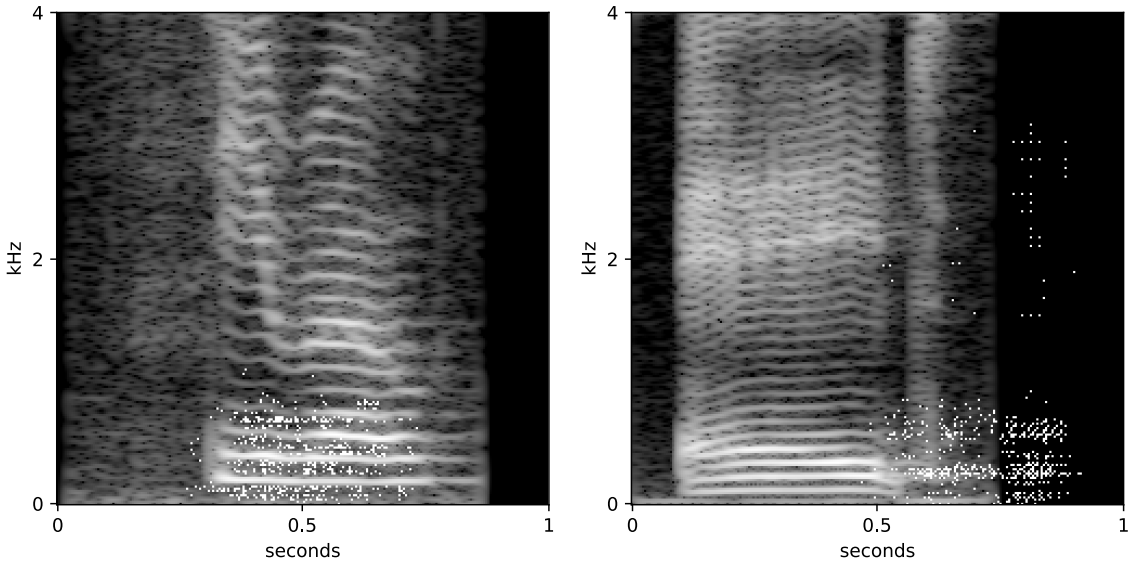

Figure 6.2: AudioMnNIST spectrograms, female(left) and male(right), where 1% of the most relevant time-frequency bins were set to zero.

Heatmaps produced by LRP are interpretable for humans but do not guarantee that these explanations are also faithful. To determine the faithfulness of the used LRP method and rules, I used the same method as proposed by S. Becker et al. [4] called pixel-flipping. This method is based on setting specific pixels, in this case, time-frequency bins, of input

data to zero before classification and evaluating the model's performance. In this experiment, from 0% to 100% of spectrogram bins were set to zero with three different strategies. First, from the most relevant spectrogram bins with the highest positive contribution according to LRP, referred to as *lrp*. The second strategy chose spectrogram bins randomly, referred to as *random*. The third strategy is the reverse of the *lrp*, where the first bins set to zero were the ones with the highest negative contribution, referred to as *lrp_rev*.

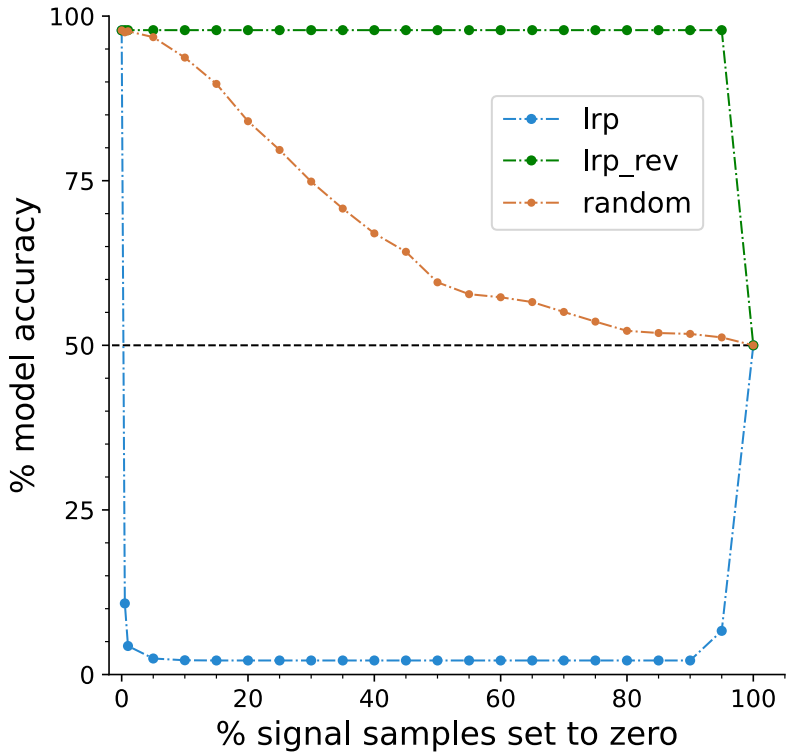

Figure 6.3: Accuracy of AlexNet model w.r.t. pixel-flipping method. Lrp represents setting time-frequency bins to zero from the ones with highest positive contribution, lrp_rev with highest negative contribution, and random sets bins at random.

Results of the pixel-flipping evaluation are shown in Figure 6.3. In case of pixel-flipping using *lrp*, AlexNet's accuracy dropped rapidly right at the beginning, where only 0.5% to 1% time-frequency bins were set to zero. Evaluation with the pixel-flipping method shows two things. First, it proves the faithfulness of LRP method, i.e. time-frequency bins highlighted by heatmaps are indeed the most relevant. The second is that this model's predictions heavily depend on a small fraction of time-frequency bins. The faithfulness is also supported by the fact that setting bins, with high negative contribution towards correct prediction, to zero, slightly increases the model's accuracy by 0.03%. Furthermore, the accuracy of the model does not decrease even when 95% of the spectrogram's time-frequency bins are set to zero, such spectrogram is shown in Figure 6.4.

Individual values of the model's accuracy w.r.t pixel-flipping are shown in Table 6.1. With only 1% of time-frequency bins changed to zero, the accuracy dropped from 97.83% to 4.33%, which uncovers the low robustness of the model. Because of a drastic drop in accuracy, I used the pixel-flipping method for only 1% of pixels to get more insight into such a sudden drop. In this experiment, N spectrogram bins were set to zero, where N

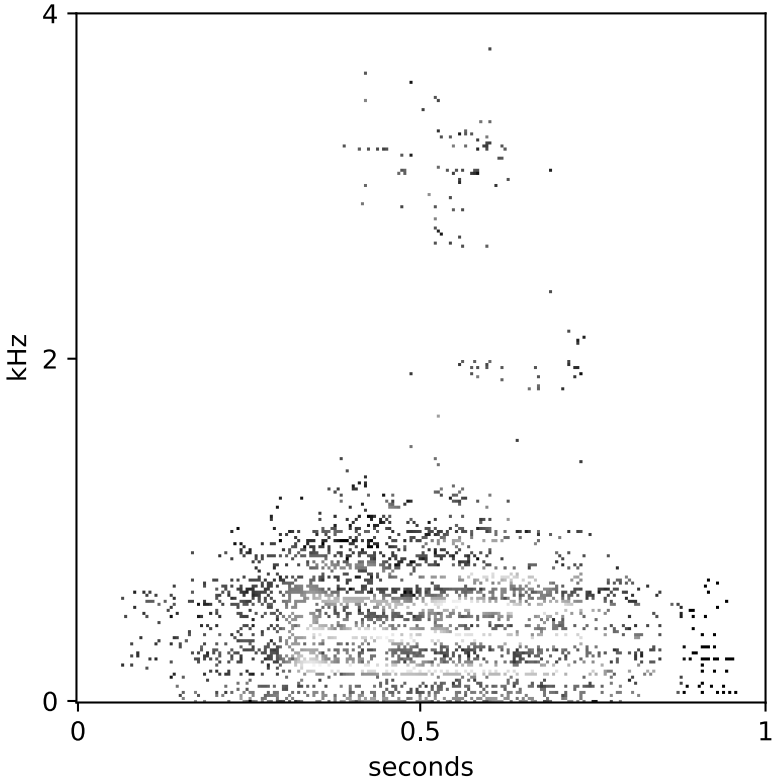

Figure 6.4: Spectrogram of female voice recording, where 95% of time-frequency bins with the lowest positive contribution towards correct prediction is set to zero w.r.t. hetmap produced by LRP. This spectrogram is still classified correctly as female voice.

takes values from 0 to 515 and is increased by 5. As shown in Figure 6.5, at only 60 to 70 flipped time-frequency bins model's accuracy drops to around 50%.

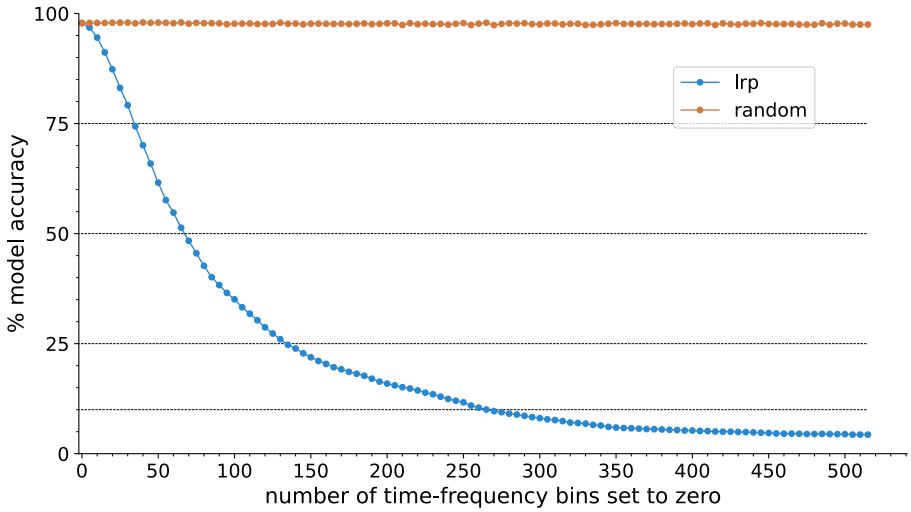

Figure 6.5: Accuracy of AlexNet model w.r.t. pixel-flipping method. Showing only first one percent (515 bins) of time-frequency bins changed.

The results of the pixel-flipping evaluation differ from the results in the original by S. Becker et al. [4]. In the original paper, the curve representing accuracy drop achieved by pixel-flipping was less steep, and in the case of pixel-flipping w.r.t. lrp, the accuracy dropped only slightly under 50%. Also, the random curve was similar to the *lrp_rev* curve in the evaluation I achieved (Figure 6.3). In this case, the drop under the chance level is caused by the low robustness of the model and the fact that the model is highly dependent on time-frequency bins with a high positive contribution towards correct prediction. When these bins are removed with the pixel-flipping method, a high portion of the positive contribution is removed as well. This results in predominantly negative contributions contributions towards correct classification, and therefore the opposite gender is classified. I do not know what exactly caused such a big difference in accuracy w.r.t. LRP but it can be caused by several factors such as a differently trained model and/or differences in LRP implementation.

Table 6.1: Numeric representation of model's accuracy showed in Figure 6.3. Rows show different approaches to pixel-flipping time-frequency bins, and columns show the model's accuracy w.r.t. percentage of the bins set to zero.

	Percentage of bins changed to zero								
Method	0	0.5	1	5	10	15	20	25	30
lrp	97.83	10.8	4.33	2.43	2.17	2.13	2.13	2.13	2.13
random	97.83	97.53	97.70	96.80	93.70	89.70	84.03	79.67	74.87
lrp_reverese	97.83	97.86	97.86	97.86	97.86	97.86	97.86	97.86	97.86

6.2 Increasing robustness of model for gender classification

I decided to use the information obtained by the explanation method in previous experiments to improve the trained model. The aim is to lower the model's high dependency on such a small number of time-frequency bins and potentially increase its performance on a validation set.

First, I created an augmented training set using the pixel-flipping method described in the previous Section 6.1 by setting 1% of the most relevant time-frequency bins to 0. Besides the bins set to zero this augmented dataset has the same features as the original AudioMNIST training dataset described in Chapter 4. The pre-trained model reaching an accuracy of 97.83% was re-trained on the augmented dataset for 100 epochs with the same hyperparameters described in Section 4.3. This new model was evaluated with the pixel-flipping method on the original AudioMNIST validation set. Results of pixel-flipping evaluation and spectrograms of this re-trained model are shown in Figure 6.6 and Figure 6.7, respectively. Figure 6.6 shows that the model accuracy is around 50%, and the *lrp* or *random* pixel-flipping method has almost no effect on it. Figure 6.7 shows the heatmap (left) produced by LRP for the re-trained model and spectrogram of the same recording with 1% of, according to LRP, the most relevant time-frequency bins set to zero (right). The spectrogram (right) in the Figure 6.7 shows that there is a shift of the most relevant bins to the higher frequency range in comparison to the spectrogram (left) in the Figure 6.2. The accuracies for different models in the Table 6.2 show that the pixel-flipping method where time-frequency bins are set to zero is not a suitable method for augmentation of a training set.

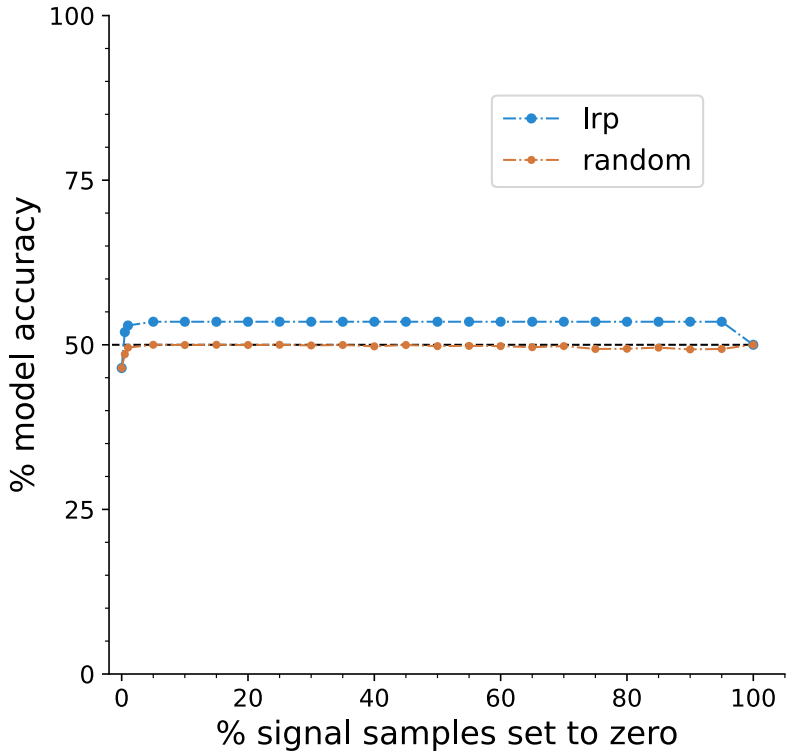

Figure 6.6: Accuracy of AlexNet model w.r.t. pixel-flipping method. Lrp represents setting time-frequency bins to zero from the ones with the highest positive contribution and random sets bins at random. This model was trained on a dataset augmented with the pixel-flipping method.

The second approach is again to change the values of the most relevant bins to lower high dependency on them. But the new values should be more less obtrusive, so they blend in, and the model eventually learns to make predictions based on more time-frequency bins. To achieve this, I propose to set the values of the 1% bins with the highest positive relevance to an average value of their Moore neighborhood. Using this method, I created augmented training set from the original AudioMNIST. I repeat the process by taking the originally trained model and train it again on the augmented dataset for 100 epochs with the same hyper-parameters as described in Section 4.3. Heatmap and spectrogram with 1% of the most relevant bins highlighted produced by the LRP method in Figure 6.8 shows that model trained on this augmented dataset makes predictions based on the lower frequencies as it should. When compared to female spectrogram in Figure 6.2, there is a slight shift in the most relevant bins, but they stayed in a low-frequency range, which is a good sign. However, the heatmaps of the new model cannot determine if it is more robust or on how many spectrograms' bins the predictions depend. As before, the pixel-flipping method is used to evaluate how the new model would respond to augmented spectrograms.

The evaluation of the improved model and comparison to the previously trained model and the original one is in Figure 6.9 and Table 6.2. The results show increased robustness of the model trained on dataset augmented by average values of Moore neighborhood. The accuracy of the improved model on the non-augmented validation dataset was increased by 1%. Because the validation set is composed of 3000 spectrograms this means that 30

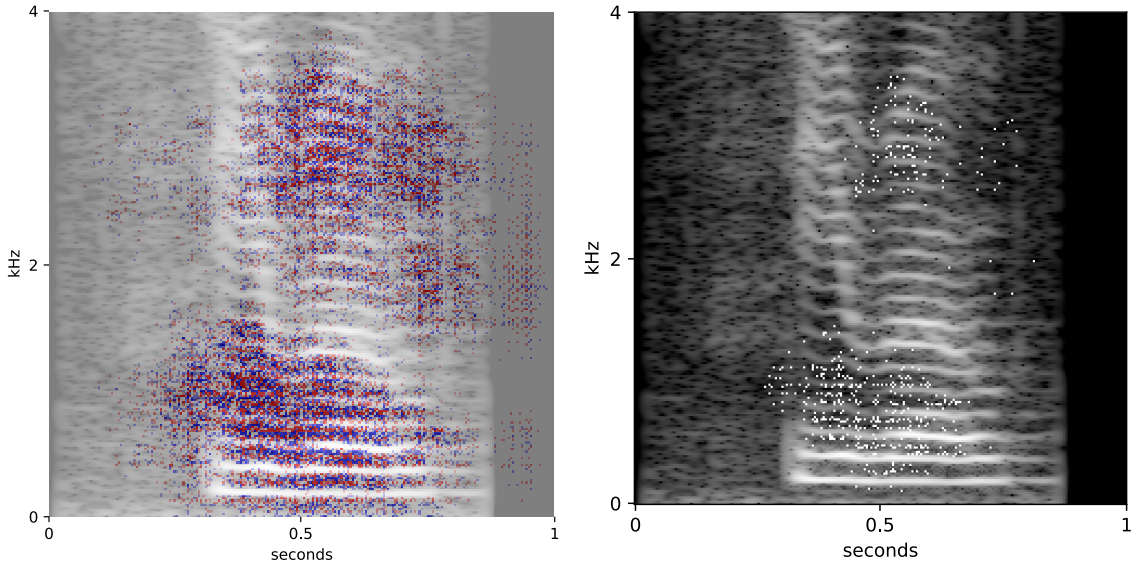

Figure 6.7: Heatmap produced by LRP of AlexNet model trained on a dataset augmented with the pixel-flipping method. Spectrogram shows 1% of the most relevant time-frequency bins and their shift to higher frequencies in comparison to the female spectrogram in Figure 6.2.

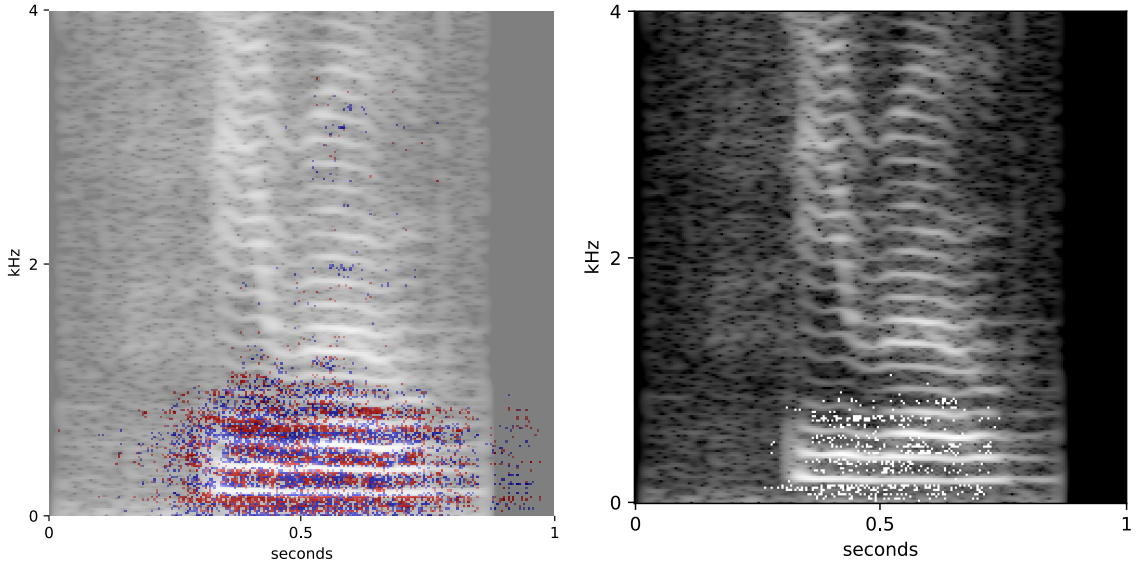

Figure 6.8: Heatmap produced by LRP of AlexNet model trained on a dataset augmented by setting the 1% of the most relevant bins to average value of their Moore neighborhood. The spectrogram on the right shows that the most relevant time-frequency bins remained in the low frequency range, similar to Figure 6.2

more data samples were classified correctly. However, the main reason for this experiment was to make the model more robust, therefore lowering the dependency on a small number of time-frequency bins. With 0.5% of the most relevant bins changed to zero, the model

achieved an accuracy of 28.6%, and with 1% of changed bins, the accuracy was 22%. This means an increase in almost 18% in both cases.

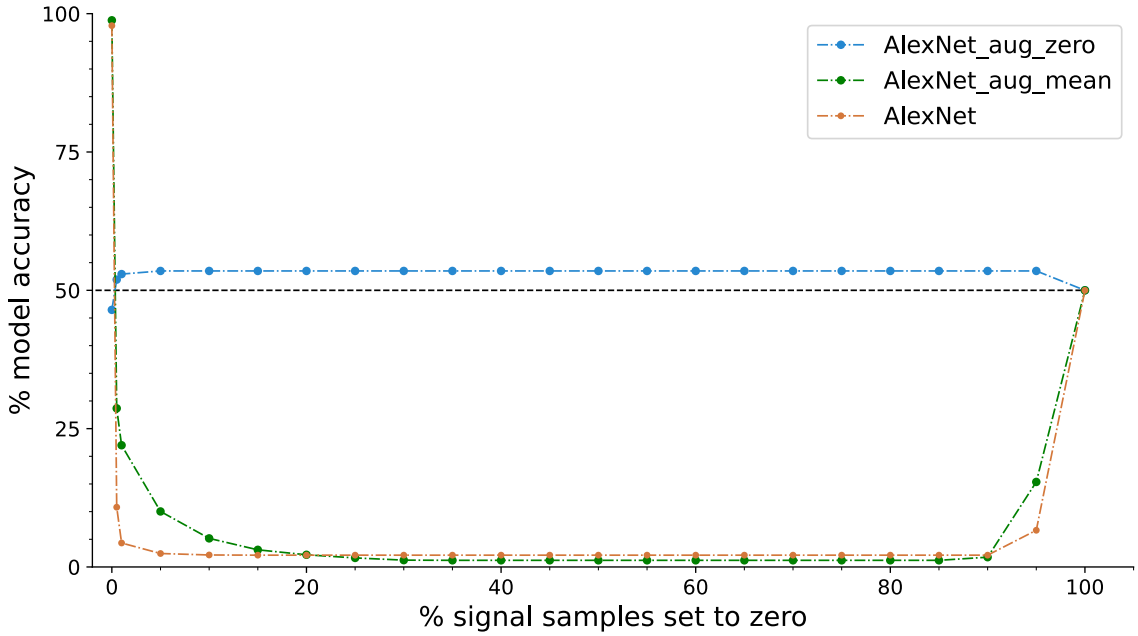

Figure 6.9: Accuracy of differently trained AlexNet models w.r.t. pixel-flipping method, where time-frequency bins were set to zero from the ones with the highest positive contribution. AlexNet represents a model trained on the original dataset. AlexNet_aug_zero represents a model trained on a dataset augmented with pixel-flipping. AlexNet_aug_mean represents a model trained on a dataset augmented by setting the most relevant pixels to an average value of their Moore neighborhood instead of zero.

Table 6.2: Numeric representation of accuracies for different models showed in Figure 6.9. Rows show different models, whereas columns show their accuracy w.r.t. percentage of relevant time-frequency bins set to zero.

	Percentage of bins changed to zero							
Model	0	0.5	1	5	10	15	20	25
AlexNet	97.83	10.8	4.33	2.43	2.17	2.13	2.13	2.13
AlexNet_aug_zero	46.67	51.93	52.93	53.5	93.70	89.70	84.03	79.67
AlexNet_aug_mean	98.80	28.67	22.00	10.03	5.17	2.2	1.63	1.23

Even though the gender classification on spectrograms is a simple task, these experiments demonstrated the usefulness of neural network interpretation in different ways:

- gaining insight into neural network predictions by creating heatmaps, in this case showing that decisions are based on lower frequencies,
- uncovering robustness and vulnerability to a small number of highly important data points,

- potentially improving models based on the obtained information,
- allowing to "debug" the neural networks as shown in Figure 6.7, where we can observe a shift of time-frequency bins important to decision making, explaining the model's drop of accuracy after training.

6.3 Interpretation of speaker ID classification with ResNet model

This experiment aims to explain a deeper neural network model, described in Section 4.4, with a similar approach to previous AlexNet explanations. The main difference in explaining this network over Alexnet is in ResNet's utilization of the batch normalization layers, pooling through time, and residual connections.

Heatmaps shown in Figure 6.10 are produced by LRP utilizing only the *LRP-0* rule on all layers. The input data are filterbanks of speech recording augmented by music (top) and noise (bottom). Time-frequency (TF) bins with positive relevance attribution are highlighted in red, whereas bins with negative attribution are blue. The heatmaps are noisy as expected when using only the *LRP-0* rule. The most relevant time-frequency bins are mostly located around lower and fundamental frequencies and spread through time, which can be considered as expected behavior. However, the *LRP-0* tends to create chunks of relevant bins that are less spread throughout the time than the combination of other LRP rules.

The unexpected behavior present in these heatmaps is the presence of positive and negative time-frequency bins in the same areas creating a checkerboard effect [27]. This effect could be a product of deconvolution created during the relevance back-propagation.

To evaluate the faithfulness of the LRP method on this model, I chose the pixel-flipping method the same way as in Sections 6.1 and 6.2. Figure 6.11 shows the accuracy of the ResNet model, w.r.t. percentage of time-frequency bins set to zero, on the training set modified with the pixel-flipping. *Random* curve represents a random choice of bins changed to zero. *Lrp* curve represents an evaluation where bins were set to zero in ascending order from the ones with the most positive relevance. *Lrp_rev* represents evaluation similar to the *lrp* but with descending order, i.e. from bins with the highest negative relevance attribution. The *lrp* and *lrp_rev* curves have an almost identical course that is steeper than the *rand* curve.

This can be interpreted as two things. First, the LRP method produces heatmaps that, to some extent, correctly show the most relevant parts in neural network decision making. Second, according to LRP, time-frequency bins with positive relevance attribution are roughly equally important as bins with negative relevance attribution in terms of correct speaker classification. The equal importance of both negative and positive bins, which is unexpected behavior, is the main difference from AlexNet interpretation experiments in Section 6.1. Another difference from AlexNet experiments is the drop in the model accuracy between lrp and random curve. In the case of ResNet, the lrp (blue) curve has a similar shape to the random (orange) curve, and it is much closer to it. This is probably caused by the difference in robustness of the models. The ResNet model is a lot more robust; therefore, changing the most important time-frequency bins to zero has not as significant an impact as in the case of AlexNet. The result of the pixel-flipping experiment can be caused by the noisy gradient in such a deep neural network as this ResNet model is. Also,

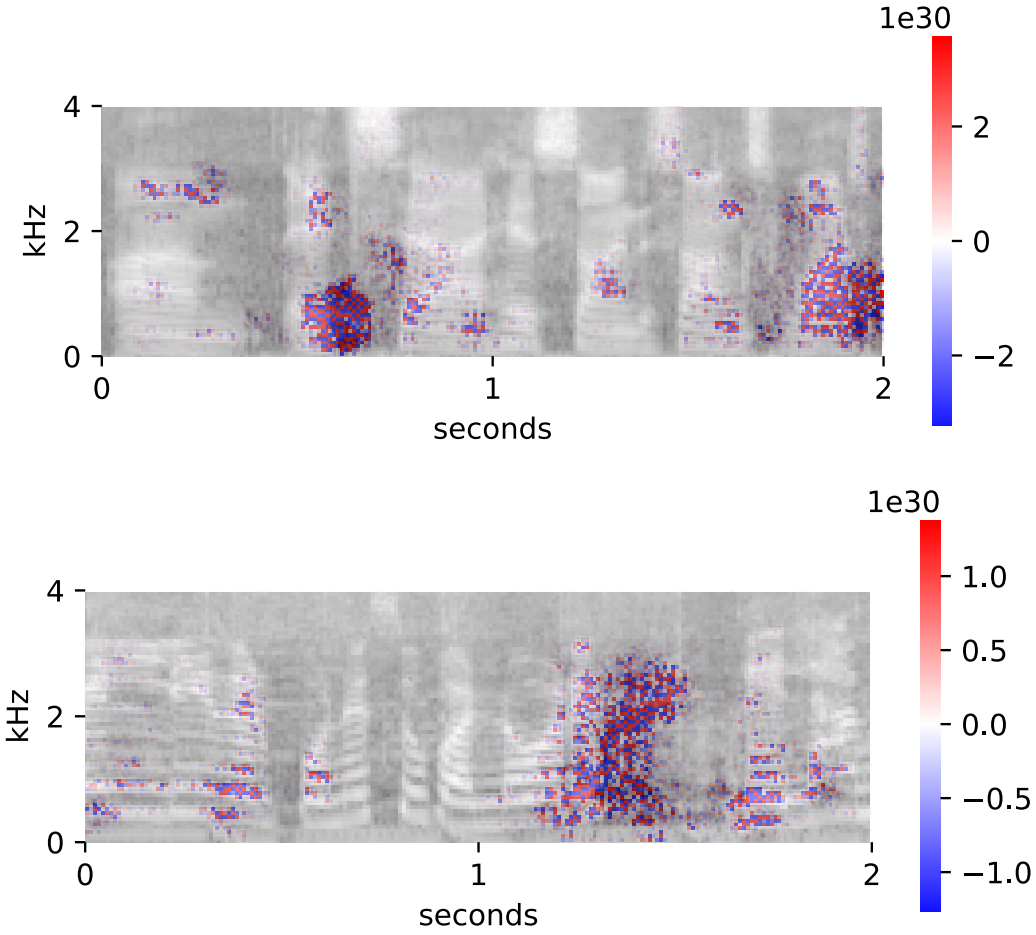

Figure 6.10: This figure shows heatmaps produced by LRP utilizing only the LRP-0 rule on top of the VoxCeleb filterbank spectrograms augmented with music(top) and noise(bottom). The red-colored time-frequency bins represent bins with positive relevance values (positive contribution towards correct prediction). The blue-colored time-frequency bins represent bins with negative relevance values (negative contribution towards correct prediction). These heatmaps show that uniform LRP-0 creates noisy heatmaps with a checkerboard effect[27], therefore, it is not a sufficient rule for interpretation of such a deep model as is Resnet.

this shows the importance of more robust LRP rules when LRP is used as an explanation method in deep neural networks.

In the previous experiments with the AlexNet (Section 6.1), even heatmaps produced only by the LRP-0 rule were interpretable, and features with positive and negative attributes were easily distinguished. But, as the Figure 6.10 shows, this does not apply to ResNet's heatmaps, and therefore using more robust rules such as LRP-γ and LRP-ϵ, for the relevance back-propagation, should bring improvements.

As the first improvement, I used LRP-ϵ, which, in theory, should make heatmaps less noisy and perhaps easier to interpret. Based on preliminary experiments, I chose $\epsilon = 0.5$. LRP-ϵ was used for most of the hidden layers consisting of convolutional and 2D batch normalization layers and input layer. For linear layers, 1D batch normalization layer and

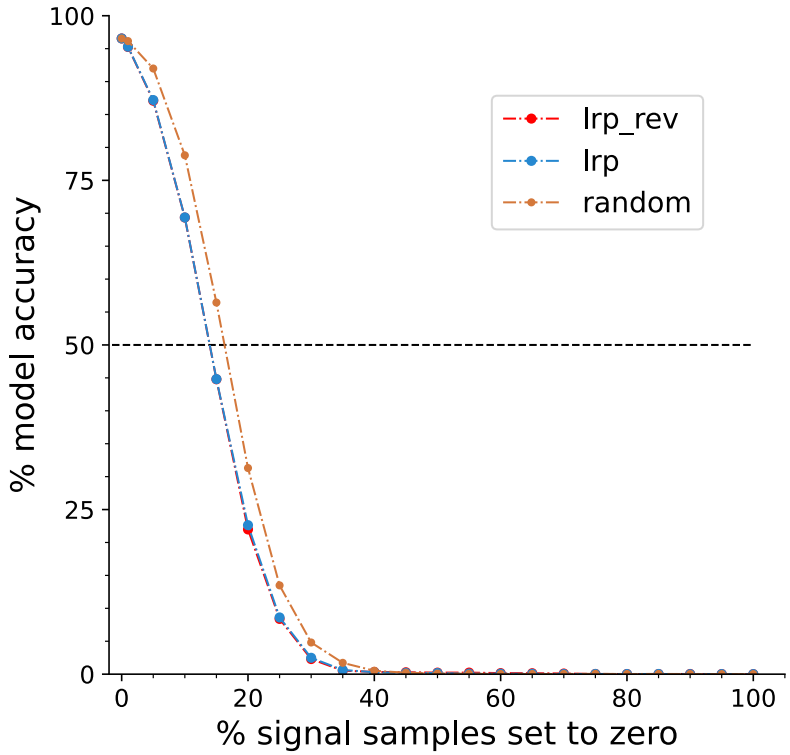

Figure 6.11: Results of pixel-flipping evaluation of LRP on ResNet model, utilizing only LRP-0 rule, made on 20000 data samples. *Lrp*(blue) curve represents a drop in model accuracy w.r.t. percentage of time-frequency bins set to zero from the bins with the highest positive contribution to correct prediction. *Lrp_rev*(red) curve represents a drop in model accuracy w.r.t. percentage of time-frequency bins set to zero, from the bins with the highest negative contribution to correct prediction. Accuracy, when time-frequency bins are randomly set to zero, is represented by a *random*(orange) curve. The fact that the lrp and lrp_rev curves have the same shape and close to the random curve shows that the LRP-0 rule fails at the interpretation of such a deep neural network as is ResNet. Also, the LRP-0 rule cannot produce heatmaps that distinguish between features with a positive and negative contribution to the correct prediction.

pooling layers before the linear ones LRP-0 remained used. This combination of the rules selection was based on the results presented by Montavon et al. [23].

Heatmaps of the same recordings as in Figure 6.10, produced after the addition of the LRP-ϵ rule shown in Figure 6.12 are significantly less noisy. The time-frequency bins with positive (red) and negative (blue) relevances are now more separated, visibly aligned along with lower frequencies, and therefore easier to interpret. Based on these heatmaps I assume, that the model's decisions are mostly based on the features in the lower frequency range.

To verify this assumption, I made an evaluation with the pixel-flipping method, as in previous experiments. The results in Figure 6.13 show a faster decrease in accuracy after setting the values of time-frequency bins to zero w.r.t. their positive relevance (blue curve) than with LRP-0 for all layers shown in Figure 6.11. In addition, the slower decrease in accuracy when pixel-flipping the bins w.r.t. their negative relevance (red curve) shows an improvement. Even though the time-frequency bins with negative relevance values have still

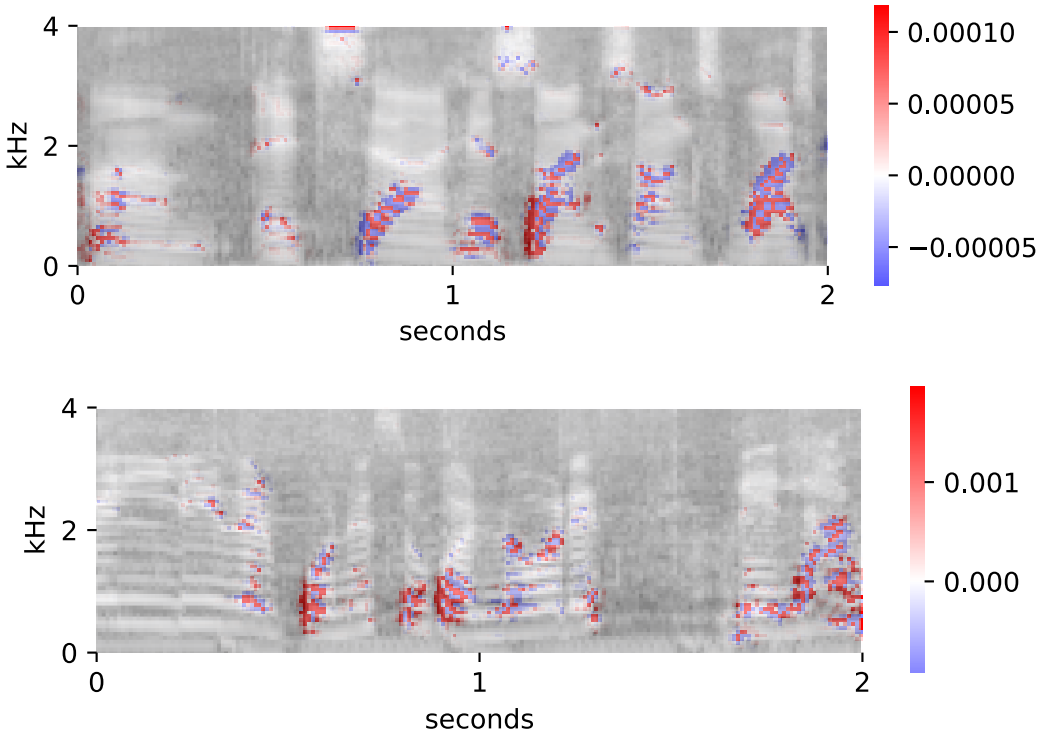

Figure 6.12: This figure shows heatmaps produced by LRP utilizing LRP-0 with LRP-eps rules, where $\epsilon = 0.8$, on top of the VoxCeleb filterbank spectrograms augmented with music(top) and noise(bottom). The red-colored time-frequency bins represent bins with positive relevance values (positive contribution towards correct prediction). The blue-colored time-frequency bins represent bins with negative relevance values (negative contribution towards correct prediction). These heatmaps show that adding a more robust rule such as LRP-eps causes less noisy features. Using LRP-eps creates heatmaps easier for interpretation by reducing noise and checkerboard effect [27].

a big impact on the model's prediction, a slower decrease in accuracy means that features contributing to an incorrect prediction are slightly better identified using LRP-ϵ.

Even though the LRP-ϵ rule in combination with the LRP-0 rule produced sufficient results, adding LRP-γ should improve the results even more in terms of heatmap interpretability. In the following experiment, the LRP-γ rule was added for relevance propagation in approximately 1/3 of the layers (from the input layer), creating a similar chain of LRP rules as shown in Figure 3.5. The ϵ value remained 0.5, and for the LRP-γ rules, the constant value was set to $\gamma = 5$. The γ value was chosen on preliminary experiments, where $\gamma \leq 1$ created sparse and noisy heatmaps that were hard to interpret and highlighted features that were not nearly the most important for the model's predictions. Heatmaps produced by a combination of all three rules (LRP-0, -ϵ, -γ) are mostly less noisy, as shown in Figure 6.14. Features highlighted in these heatmaps are more separated based on their contribution to the correct prediction (positive,i.e. red or negative, i.e. blue) relevance values). These highlighted features suggest that a speaker is classified based on found features in his voice throughout the time and mostly in a lower frequency range. The presence of

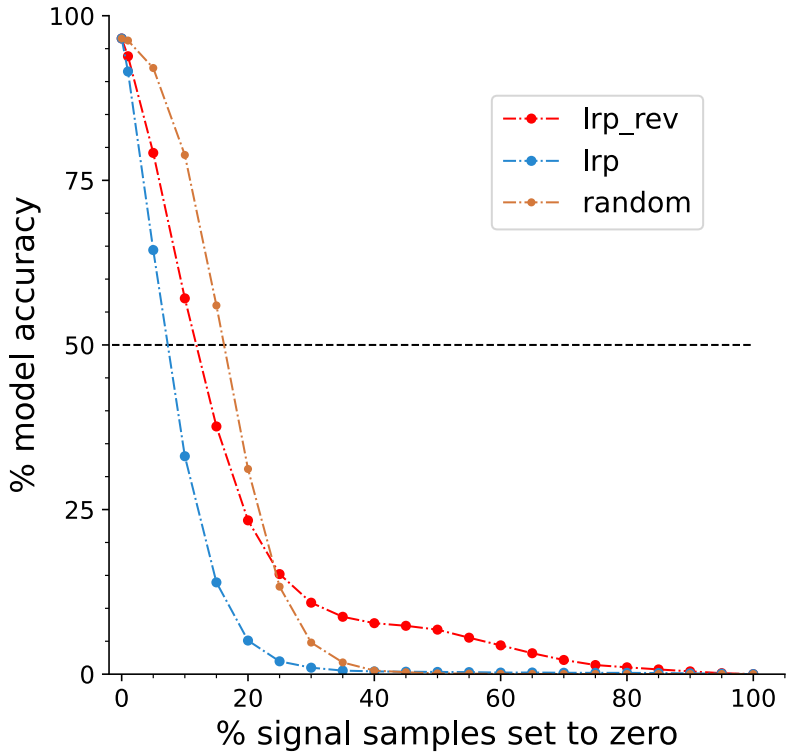

Figure 6.13: Results of pixel-flipping evaluation of LRP on ResNet model, utilizing LRP-0 and LRP-ϵ rules, where $\epsilon = 0.8$, made on 20000 data samples. *Lrp*(blue) curve represents a drop in model accuracy w.r.t. percentage of time-frequency bins set to zero from the bins with the highest positive contribution to correct prediction. *Lrp_rev*(red) curve represents a drop in model accuracy w.r.t. percentage of time-frequency bins set to zero, from the bins with the highest negative contribution to correct prediction. Accuracy, when time-frequency bins are randomly set to zero, is represented by a *random*(orange) curve. The shape of *Lrp*, *lrp_rev* and the distance between them show that LRP successfully finds features positively contributing to correct prediction but is less successful in finding the most important features that negatively contribute to correct prediction.

features in lower frequencies is similar to results obtained by previous experiments with AlexNet in Section 6.1.

The faithfulness evaluation of the relevance heatmaps, produced by the combination of three rules, can be seen in Figure 6.15. The method for this evaluation was again pixel-flipping with the same meaning of individual curves as in previous experiments. The *lrp* curve, where time-frequency bins with the most positive relevance values are set to zero, is less steep in comparison to the previous evaluation (Figure 6.13), where only LRP-0 and LRP-ϵ rules were used. This change can be caused by the fact that LRP-γ favors features with positive relevance values over the features with negative relevance values. Therefore, some time-frequency bins that are less important may have higher positive relevance when using LRP-γ over LRP-ϵ. On the other hand, the *lrp_rev* curve, where time-frequency bins with the highest negative value are set zero first, is a lot less steep compared to using only LRP-ϵ and LRP-0 rules. This behavior suggests that the addition of the LRP-γ rule better separates and highlights features with a negative contribution towards correct prediction.

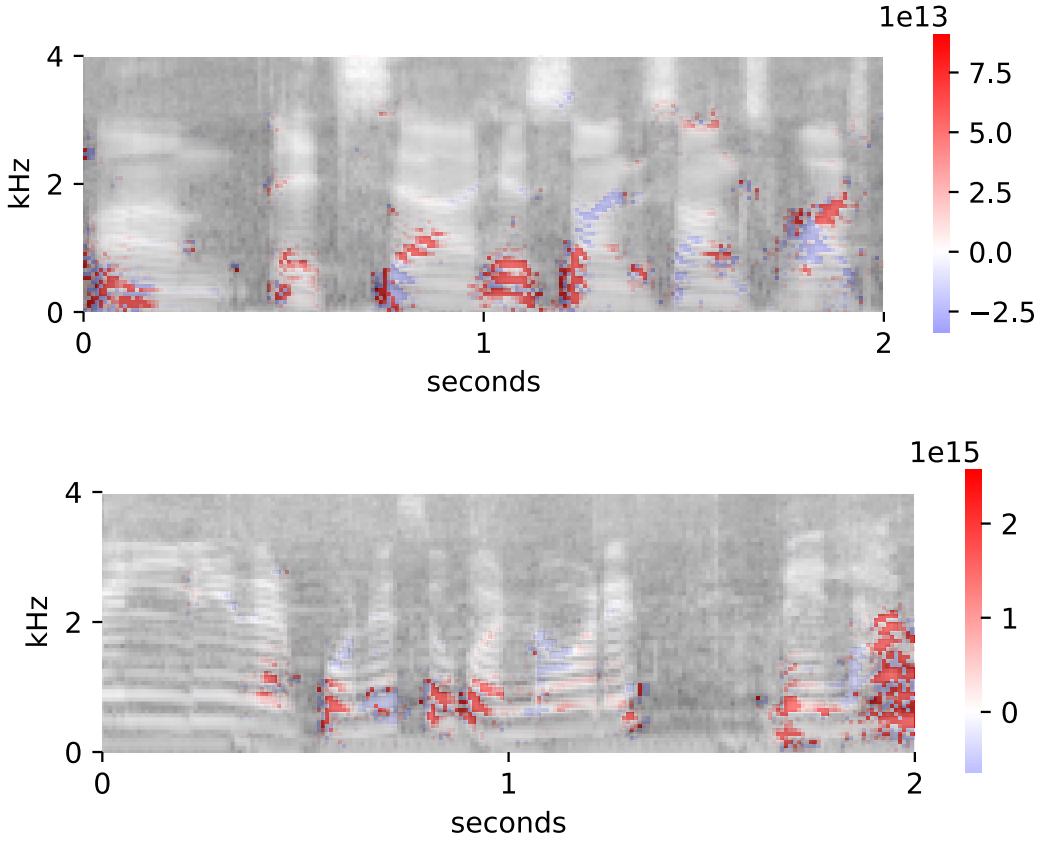

Figure 6.14: This figure shows heatmaps produced by LRP utilizing a combination of LRP-0, LRP-ϵ, and LRP-γ rules, where $\epsilon = 0.8$ and $\gamma = 5$, on top of the VoxCeleb filterbank spectrograms augmented with music(top) and noise(bottom). The red-colored time-frequency bins represent bins with positive relevance values (positive contribution towards correct prediction). The blue-colored time-frequency bins represent bins with negative relevance values (negative contribution towards correct prediction). In comparison to Figure 6.12, the addition of the LRP-γ rule caused even less noisy heatmaps. Also, it visibly divided features with the positive and negative contribution to the correct prediction, which makes these heatmaps even better and easier to interpret.

In conclusion, both combinations of rules, LRP-0 + LRP-ϵ and LRP-0 + LRP-ϵ + LRP-γ, could highlight features important for correct speaker classification. According to the results of pixel-flipping evaluations LRP-0 + LRP-ϵ could find individual time-frequency bins or smaller areas of important features better. On the other hand, the combination of LRP-0 + LRP-ϵ + LRP-γ could better distinguish between features that contribute to the correct prediction and the ones that do not. Additionally, this combination of rules produced nicer heatmaps better for interpretation with decent faithfulness.

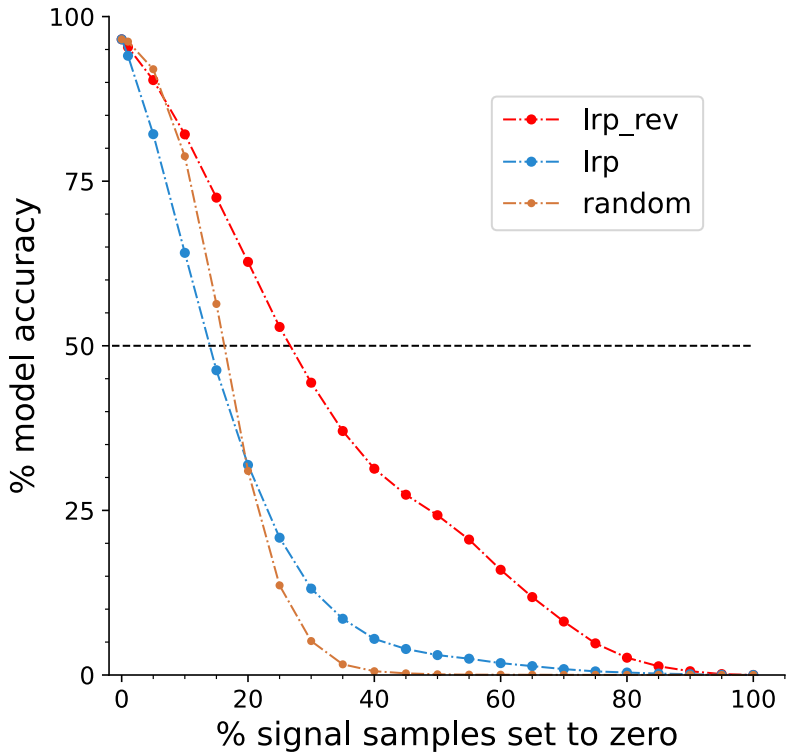

Figure 6.15: Results of pixel-flipping evaluation of LRP utilizing LRP-0, LRP-ϵ, and LRP-γ, where $\epsilon = 0.8$ and $\gamma = 5$, on ResNet model made on 20000 data samples. *Lrp*(blue) curve represents a drop in model accuracy w.r.t. percentage of time-frequency bins set to zero from the bins with the highest positive contribution to correct prediction. *Lrp_rev*(red) curve represents a drop in model accuracy w.r.t. percentage of time-frequency bins set to zero, from the bins with the highest negative contribution to correct prediction. Accuracy, when time-frequency bins are randomly set to zero, is represented by a *random*(orange) curve. The shape of lrp, lrp_rev and the distance between them show that LRP can be considered faithful because it correctly highlights features important for correct or incorrect prediction.

Chapter 7

Conclusion

The goal of this thesis was an interpretation of the deep neural network used for audio classification, i.e., find features in data with high contribution to made prediction and try to replicate results of gender classification from spectrograms originally presented by S. Becker et al. [4]. Then with information obtained from previous results extend the interpretation experiments further.

I successfully produced heatmaps similar to the ones presented in the original paper. According to these heatmaps, used convolutional neural network made gender predictions from spectrograms based on lower frequencies. Evaluation of these heatmaps with the pixel-flipping method showed their faithfulness and uncovered low robustness of the trained model. Even though this model achieved 97.8% accuracy in predicting correct gender, using LRP, I found out that the predictions were based on a small number of spectrogram's time-frequency (TF) bins. Using the pixel-flipping method w.r.t. LRP, the model's accuracy of predicting correct gender dropped from 97.8% to only 10.8% when only 0.5% of TF bins were set to zero. To extend these experiments and use information obtained from heatmaps, I create an augmented training dataset w.r.t. heatmaps and re-trained the model. I manage to make the re-trained model more robust, i.e. less dependant on such a small number of TF bins, boosting its accuracy when 0.5% of TF bins were set to zero w.r.t. LRP from 10.8% to 28.67%. I further extended the experiments using LRP to interpret a more complex ResNet model trained for the speaker ID classification task. Interpretation of this model showed some negative effects caused by deep convolutional networks when using LRP-0 rule, such as noisy heatmaps or checkerboard effect presented in heatmaps. I overcame these problems using more robust LRP rules during propagation, creating less noisy and relatively easier to interpret heatmaps.

For future work, I propose implementing LRP, or even other interpretation methods, for more neural network layers and change the way of creating these layers during computation so the computation of LRP can be modified and easily used for different models. Because LRP is heavily dependent on the model's architecture and needs to have access to its layer, such generalized implementation could potentially make usage of interpretation methods more common not only in the field of research. For example, LRP could be used as a form of debugging for neural networks, i.e. by looking on heatmaps that highlight important features in data, one could determine if these features represent desired behavior or not. Another example is using interpretation methods as a tool to increase deep neural network credibility among people, e.g. when a company creates a product using machine learning.

Bibliography

[1] Aggarwal, C. *Neural Networks and Deep Learning: A Textbook*. January 2018. ISBN 978-3-319-94462-3.

[2] Arras, L., Osman, A., Müller, K. and Samek, W. Evaluating Recurrent Neural Network Explanations. *CoRR*. 2019, abs/1904.11829. Available at: http://arxiv.org/abs/1904.11829.

[3] Balduzzi, D., Frean, M., Leary, L., Lewis, J. P., Ma, K. W.-D. et al. The Shattered Gradients Problem: If resnets are the answer, then what is the question? In: Precup, D. and Teh, Y. W., ed. *Proceedings of the 34th International Conference on Machine Learning*. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, vol. 70, p. 342–350. Proceedings of Machine Learning Research. Available at: http://proceedings.mlr.press/v70/balduzzi17b.html.

[4] Becker, S., Ackermann, M., Lapuschkin, S., Müller, K.-R. and Samek, W. Interpreting and Explaining Deep Neural Networks for Classification of Audio Signals. *CoRR*. 2018, abs/1807.03418.

[5] Bishop, C. *Pattern Recognition and Machine Learning*. Springer, January 2006. Available at: https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/.

[6] Brownlee, J. *Loss and Loss Functions for Training Deep Learning Neural Networks* [online]. Machine Learning Mastery Pty. Ltd., october 2019 [cit. 2021-03-02]. Available at: https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/.

[7] Chollet, F. et al. *Deep learning with Python*. Manning New York, 2018.

[8] Chung, J. S., Nagrani, A. and Zisserman, A. VoxCeleb2: Deep Speaker Recognition. In: *INTERSPEECH*. 2018.

[9] Ding, B., Qian, H. and Zhou, J. Activation functions and their characteristics in deep neural networks. In: *2018 Chinese Control And Decision Conference (CCDC)*. 2018, p. 1836–1841. DOI: 10.1109/CCDC.2018.8407425.

[10] Godin, F., Degrave, J., Dambre, J. and De Neve, W. Dual Rectified Linear Units (DReLUs): A Replacement for Tanh Activation Functions in Quasi-Recurrent Neural Networks. *Pattern Recognition Letters*. september 2018, vol. 116. DOI: 10.1016/j.patrec.2018.09.006.

[11] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[12] *ML Practicum: Image Classification* [online]. Google, september 2020 [cit. 2021-02-23]. Available at: https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks.

[13] He, K., Zhang, X., Ren, S. and Sun, J. Deep Residual Learning for Image Recognition. *CoRR*. 2015, abs/1512.03385. Available at: http://arxiv.org/abs/1512.03385.

[14] Hui, L. Y. W. and Binder, A. BatchNorm Decomposition for Deep Neural Network Interpretation. In: Rojas, I., Joya, G. and Catala, A., ed. *Advances in Computational Intelligence*. Cham: Springer International Publishing, 2019, p. 280–291. ISBN 978-3-030-20518-8.

[15] Karayiannis, N. and Venetsanopoulos, A. *Artificial Neural Networks: Learning Algorithms, Performance Evaluation, and Applications*. January 1993. ISBN 978-1-4419-5132-8.

[16] Kohlbrenner, M., Bauer, A., Nakajima, S., Binder, A., Samek, W. et al. Towards Best Practice in Explaining Neural Network Decisions with LRP. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. 2020, p. 1–7. DOI: 10.1109/IJCNN48605.2020.9206975.

[17] Lapuschkin, S. *Opening the machine learning black box with Layer-wise Relevance Propagation*. Berlin, 2019. Doctoral Thesis. Technische Universität Berlin. Available at: http://dx.doi.org/10.14279/depositonce-7942.

[18] Lapuschkin, S., Binder, A., Montavon, G., Müller, K.-R. and Samek, W. The LRP Toolbox for Artificial Neural Networks. *Journal of Machine Learning Research*. 2016, vol. 17, no. 114, p. 1–5. Available at: http://jmlr.org/papers/v17/15-618.html.

[19] Lapuschkin, S., Binder, A., Montavon, G., Müller, K.-R. and Samek, W. Analyzing Classifiers: Fisher Vectors and Deep Neural Networks. In:. June 2016, p. 2912–2920. DOI: 10.1109/CVPR.2016.318.

[20] Lapuschkin, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R. et al. On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PLoS ONE*. july 2015, vol. 10, p. e0130140. DOI: 10.1371/journal.pone.0130140.

[21] Mehlig, B. Artificial Neural Networks. *CoRR*. 2019, abs/1901.05639. Available at: http://arxiv.org/abs/1901.05639.

[22] Miller, T. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*. 2019, vol. 267, p. 1–38. DOI: https://doi.org/10.1016/j.artint.2018.07.007. ISSN 0004-3702. Available at: https://www.sciencedirect.com/science/article/pii/S0004370218305988.

[23] Montavon, G., Binder, A., Lapuschkin, S., Samek, W. and Müller, K.-R. Layer-Wise Relevance Propagation: An Overview. In: Samek, W., Montavon, G.,

Vedaldi, A., Hansen, L. K. and Müller, K.-R., ed. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Cham: Springer International Publishing, 2019, p. 193–209. DOI: 10.1007/978-3-030-28954-6_10. ISBN 978-3-030-28954-6. Available at: https://doi.org/10.1007/978-3-030-28954-6_10.

[24] Montavon, G., Samek, W. and Müller, K.-R. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*. 2018, vol. 73, p. 1–15. DOI: https://doi.org/10.1016/j.dsp.2017.10.011. ISSN 1051-2004. Available at: https://www.sciencedirect.com/science/article/pii/S1051200417302385.

[25] Nagrani, A., Chung, J. S. and Zisserman, A. VoxCeleb: a large-scale speaker identification dataset. In: *INTERSPEECH*. 2017.

[26] Nagrani, A., Chung, J. S., Xie, W. and Zisserman, A. Voxceleb: Large-scale speaker verification in the wild. *Computer Science and Language*. Elsevier. 2019.

[27] Odena, A., Dumoulin, V. and Olah, C. Deconvolution and Checkerboard Artifacts. *Distill*. 2016. DOI: 10.23915/distill.00003. Available at: http://distill.pub/2016/deconv-checkerboard.

[28] O'Shea, K. and Nash, R. An Introduction to Convolutional Neural Networks. *ArXiv e-prints*. november 2015.

[29] Park, Y.-S. and Lek, S. Chapter 7 - Artificial Neural Networks: Multilayer Perceptron for Ecological Modeling. In: Jørgensen, S. E., ed. *Ecological Model Types*. Elsevier, 2016, vol. 28, p. 123–140. Developments in Environmental Modelling. DOI: https://doi.org/10.1016/B978-0-444-63623-2.00007-4. ISSN 0167-8892. Available at: https://www.sciencedirect.com/science/article/pii/B9780444636232000074.

[30] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché Buc, F. d', Fox, E. et al., ed. *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, p. 8024–8035. Available at: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[31] Ribeiro, M., Singh, S. and Guestrin, C. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In:. February 2016, p. 97–101. DOI: 10.18653/v1/N16-3020.

[32] Rosenblatt, F. The Perceptron: A Probabilistic Model for Information Storage and Organization (1958). In:. February 2021, p. 183–190. DOI: 10.7551/mitpress/12274.003.0020. ISBN 9780262363174.

[33] Samek, W., Montavon, G., Lapuschkin, S., Anders, C. and Müller, K.-R. *Toward Interpretable Machine Learning: Transparent Deep Neural Networks and Beyond*. March 2020.

[34] Samek, W., Montavon, G., Vedaldi, A., Hansen, L. and Müller, K.-R. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. January 2019. ISBN 978-3-030-28953-9.

[35] Sathya, R. and Abraham, A. Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. *International Journal of Advanced Research in Artificial Intelligence*. february 2013, vol. 2. DOI: 10.14569/IJARAI.2013.020206.

[36] Sharma, S., Sharma, S. and Athaiya, A. ACTIVATION FUNCTIONS IN NEURAL NETWORKS. *International Journal of Engineering Applied Sciences and Technology*. may 2020, vol. 04, p. 310–316. DOI: 10.33564/IJEAST.2020.v04i12.054.

[37] Snyder, D., Chen, G. and Povey, D. *MUSAN: A Music, Speech, and Noise Corpus*. 2015.

[38] Soni, D. *The Vanishing Gradient Problem* [online]. Towards Data Science Inc., march 2018 [cit. 2021-03-02]. Available at: `https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d`.

[39] Suzuki, K. *Artificial Neural Networks - Methodological Advances and Biomedical Applications*. April 2011. ISBN 978-953-307-243-2.

[40] Swartout, W. and Moore, J. Explanation in Second Generation Expert Systems. In:. January 1993, p. 543–585. DOI: 10.1007/978-3-642-77927-5_24.

[41] Theodoridis, S. and Koutroumbas, K. *Pattern Recognition, Fourth Edition*. November 2008. ISBN 1597492728.

[42] Traunmüller, H. and Eriksson, A. The frequency range of the voice fundamental in the speech of male and female adults. january 1995, vol. 2.

[43] Wang, C.-F. *The Vanishing Gradient Problem* [online]. Towards Data Science Inc., january 2019 [cit. 2021-03-02]. Available at: `https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484`.

Appendix A

Contents of the included storage media

```
/
├── augmented_datasets/
├── data/
├── eval_data/
│   ├── AlexNet/
│   └── speakerID/
├── figures/
├── interpretation/
│   ├── __init__.py
│   ├── interpret_methods.py
│   └── lrp_class.py
├── models/
│   ├── AlexNet/
│   ├── AudioNet/
│   ├── customDataset.py
│   ├── __init__.py
│   └── wrapper.py
├── SpeakerID/
├── thesis/
├── AlexNet_experiments.ipynb
├── AlexNet_pf_video.py
├── AlexNet_wrong.txt
├── cnn_lrp_demo.py
├── create_dataset.py
├── dataset_check.py
├── lrp_eval.py
├── README.md
├── relu.py
├── speakerID_eval.py
├── speakerID_experiments.ipynb
├── speakerID_heatmaps.py
└── speaker_lrp_demo.py
```

```
├── utils.py
├── enviroment.yml
├── LICENSE
└── thesis.pdf
```

Source files (.py and .ipynb) and how to setup an enviroment are described in **README.md** in included storage media.

- **augmented_datasets/** — datasets created using pixel-flipping method w.r.t. heatmaps produced by LRP
- **data/** — AlexNet spectrograms in .hdf5 format for demo
- **figures/** — experiment figures used in thesis
- **interpretation/** — source codes for Layer-wise Relevance Propagation
- **models/** — AlexNet architecture, trained models and utilities for model loading, training, evaluating and etc.
- **SpeakerID/** — ResNet model and VoxCeleb data used for experiments
- **thesis/** — LaTeX source code for thesis
- **README.md** — manual for running the code, loading enviroment and description of .py and .ipynb scripts
- **enviroment.yml** — anaconda enviroment with dependencies
- **LICENSE** — code license
- **thesis.pdf** — Bachelor thesis in pdf