Radboud University Nijmegen

Faculty of Science

Convolutional Neural Networks applied to Keyword Spotting using Transfer Learning

Thesis in Automatic Speech Recognition
(LET-REMA-LCEX10)

Author:

Christoph Schmidl
s4226887
c.schmidl@student.ru.nl

Supervisor:

dr. L.F.M. ten Bosch

August 15, 2019

Contents

1 Introduction

The task of keyword spotting (KWS) is interesting to different domains where a hands-free interaction experience is required or desired like Google's feature of interacting with mobile devices (include "OK Google" reference).

Different approaches to keyword spotting like:

- Deep Neural Networks (DNNs)
- Convolutional Neural Networks (CNNs)
- (Keyword/Filler) Hiddem Markov Models (HMMs)

1. Problem
2. Background (literature overview)
3. Research Question, Hypotheses, intro to experiment

1.1 Literature review

This section contains the most prominent approaches to the KWS task which have been successfully applied in the past and serve as baseline models or inspirations for the proposed model in this thesis.

1.1.1 Raw waveform-based audio classification using sample-level CNN architectures

- Raw waveform-based audio classification using sample-level CNN architectures [13]

1.1.2 Transferable deep features for keyword spotting

- Transferable deep features for keyword spotting [15]

1.1.3 Imagenet: A large-scale hierarchical image database

- Imagenet: A large-scale hierarchical image database [5]

1.1.4 Imagenet classification with deep convolutional neural networks

- Imagenet classification with deep convolutional neural networks [11]

1.1.5 Speech Recognition: Keyword Spotting Through Image Recognition.

The authors of the paper "Speech Recognition: Keyword Spotting Through Image Recognition" [10] transformed the KWS task which incoporates audio data into the domain of image classification. They used the Speech Commands Dataset [20] which contains spoken words of the length of one second in order to train and evalutate their model. According to [20], the Speech Commands Dataset V2 [9] comprises one-second audio clips which were sampled at 16KHz and containing ten words, namely "Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop", and "Go", and have one additional special label for "Unknown Word", and another for "Silence" (no speech detected). A vector representation of these one-second audio clips would therefore be of the form $\mathbb{R}^{16000}$.

The authors used three different models, namely:

- A Low Latency Convolutional Neural Network which is designed to reduce its memory footprint by limiting the number of model parameters. This model is similar to the model called ”cnn-one-fstride4” which is used in [16] but differs in terms of filter size, stride, channels, dense and params. The model has been trained using Stochastic Gradient Descent and Xavier Initialization has been used in order to initialize the model weights.

- The MNIST TensorFlow CNN / Basic CNN where some tweaks have been performed to the first layer in order to fix dimension mismatches. A baseline architecture is described in [16] (3 module setup?).

  - `https://github.com/tensorflow/docs/blob/master/site/en/tutorials/estimators/cnn.ipynb`
  - `https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/tutorials/mnist`

- Adversarially trained CNN which is inspired by MCDNN [4] and AlexNet [11]. One shallow CNN which has been used for prototyping and hyperparameter tuning. Dropout was counter-productive and therefore Virtual Adversarial Training was used, inspired by [7].

Evaluated parameters in this paper:

- Adversarial Training Results and Comparison with Vanilla CNN
- increase in training and validation accuracy over the first ten epochs for the low-latency convolution and VAT models: a zoomed-in version of Figure 12
- decrease of costs over 500 epochs for the lowlatency convolution and VAT models: a zoomed-out version of Figure 13
- increase of training and validation accuracy over 500 epochs for the low-latency convolution and VAT models: a zoomed-out version of Figure 10
- reduction of cost over the first ten epochs for the low-latency convolution and VAT models: a zoomed-in version of Figure 11
- the evolution of training cross-entropy loss (blue and green) and validation accuracy (red and orange) compared between Xavier and truncated normal initialization; Xavier converges much faster and may attain better results
- the evolution of training cross-entropy loss (blue and green) and validation accuracy (red and orange) compared between Adam and SGD optimization; Adam converges faster than SGD but reaches the same results
- effect of the number of frequency-counting buckets on the accuracy of the low-latency convolution model. The model did not benefit from the increase in available data caused by increasing the number of buckets.
- effect of the spectrogram window size on the accuracy of the low-latency convolution model. There is a local optimum, as there was for stride in Figure 18.
- effect of added background noise on the final accuracy of the low-latency convolution model. The horizontal axis is signal-noise ratio in linear units.

(a) (b)

Figure 1: A comparison of the spectrogram (a) and the amplitude-vs.-time plot (b) for the same audio recording of a person saying the word "bed".

- effect of the spectrogram window stride on the accuracy of the low-latency convolution model. For low values of stride, there is too much redundancy, while larger values result in lost information.

Conlusion

In this project we tackled the speech recognition problem by applying different CNN models on image data formed using log spectrograms of the audio clips. We also successfully implemented a regularization method "Virtual Adversarial Training" that achieved a maximum of 92% validation accuracy on 20% random sample of the input data.
The significant work done in this project was the demonstration of how to convert a problem in audio recognition into the better-studied domain of image classification, where the powerful techniques of convolutional neural networks are fully developed. We also saw, particularly in the case of the low-latency convolution model, how crucial good hyperparameter tuning is to the accuracy of the model. A great number of hyperparameters must be tuned, including the many choices that go into network architecture, and any of the hyperparameters, poorly chosen, can make or break the overall performance of the model. Another contribution was the use of adversarial training to provide a regularization effect in audio recognition; this technique improved results relative even to well-established techniques such as dropout, and therefore has promising applications in the future.

Include summary about the approach of converting the long, one dimensional vector of audio data into a spectrograms and therefore making it a image classification problem.

1.1.6 Convolutional neural networks for small-footprint keyword spotting

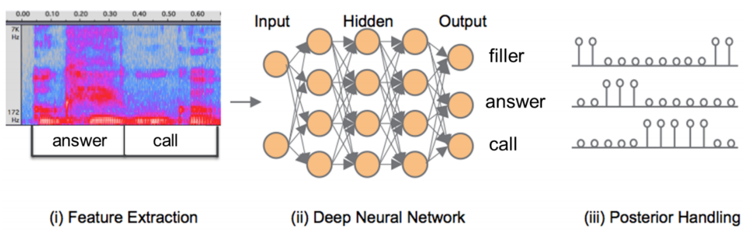


Figure 2: Framework of Deep KWS system, components from left to right: (i) Feature Extraction (ii) Deep Neural Network (iii) Posterior Handling

This framework originally comes from [3]. The only difference is the exchange of the DNN for a CNN.

- Convolutional neural networks for small-footprint keyword spotting [16]

Include summary about the different CNNs approaches which have been put into the 3 module framework of the below framework where the DNN has been exchanged for a CNN. How do the authors handle the long, one dimensional vector?

1.1.7 Small-footprint keyword spotting using deep neural networks

- Small-footprint keyword spotting using deep neural networks [3]

Include summary about the comparison between DNNs and HMMs and the general 3 module approach here: 1. Feature extraction. 2. Deep Neural Network 3. Posterior Handling. DNNs do not need a decoding algorithm like HMMs with Viterbi which makes it low latency.

1.1.8 Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition

- Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition [20]

Include summary and say why the Speech Commands Dataset is a good fit for this thesis. You probably do not need a Voice-activity detection (VAD) system here.

Properties

> The final dataset consisted of 105,829 utterances of 35 words[...].
> Each utterance is stored as a one-second (or less) WAVE format file, with the sample data encoded as linear 16-bit single-channel PCM values, at a 16 KHz rate. There are 2,618 speaker recorded, each with a unique eight-digit hexadecimal identifier assigned as described above. The uncompressed files take up approximately 3.8 GB on disk, and can be stored as a 2.7 GB gzip-compressed tar archive.

Top-One Error

> The standard chosen for the TensorFlow speech commands example code is to look for the ten words "Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop", and "Go", and have one additional special label for "Unknown Word", and another for "Silence" (no speech detected). The testing is then done by providing equal numbers of examples for each of the twelve categories, which means each class accounts for approximately 8.3% of the total. The "Unknown Word" category contains words randomly samples from classes that are part of the target set. The "Silence" category has one-second clips extracted randomly from the background noise audio files.

Applications

> The TensorFlow tutorial gives a variety of baseline models, but one of the goals of the dataset is to enable the creation and comparison of a wide range of models on a lot of different platforms, and version one of has enabled some interesting applications. CMSISNN [21] covers a new optimized implementation of neural network operations for ARM microcontrollers, and uses Soeech Commands to train and evaluate the results. Listening to the World [22] demonstrates how combining the dataset and UrbanSounds [23] can improve the noise tolerance of recognition models. Did you Hear That [24] uses the dataset to test adversarial attacks on voice interfaces. Deep Residual Learning for Small Footprint Keyword Spotting [18] shows how approaches learned from ResNet can produce more efficient and accurate models. Raw Waveformbased Audio Classification [13] investigates alternatives to traditional feature extraction for speech and music models. Keyword Spotting

Data	V1 Training	V2 Training
V1 Test	85.4%	89.7%
V2 Test	82.7%	88.2%

Table 1: Top-One accuracy evaluations using different training data

through Image Recognition [10] looks at the effect of virtual adversarial training on the keyword task.

Evaluation

One of this dataset's primary goals is to enable meaningful comparisons between different models' results, so it's important to suggest some precise testing protocols. As a starting point, it's useful to specify exactly which utterances can be used for training, and which must be reserved for testing, to avoid overfitting. The dataset download includes a text file called `validation_list.txt`, which contains a list of files that are expected to be used for validating results during training, and so can be used frequently to help adjust hyperparameters and make other model changes. The `testing_list.txt` file contains the names of audio clips that should only be used for measuring the results of trained models, not for training or validation. The set that a file belongs to is chosen using a hash function on its name. This is to ensure that files remain in the same set across releases, even as the total number in the same set across releases, even as the total number changes, so avoid set corss-contaimination when trying old models on the more recent test data. The Python implementation of the set assignment algorithm is given in the TensorFlow tutorial code [12] that is a companion to the dataset.

Historical Evaluations

Version 1 of the dataset [8] was released August 3rd 2017, and contained 64,727 utterances from 1,881 speakers. Training the default convolution model from the TensorFlow tutorial (based on Convolutional Neural Networks for Small-footprint Keyword Spotting [16]) using the V1 training data gave a Top-One score of 85.4%, when evaluated against the test set from V1. Training the same model against version 2 of the data set [9], documented in this paper, produces a model that scores 88.2% Top-One on the training set extracted from the V2 data. A model trained on V2 data, but evaluated against the V1 test set gives 89.7% Top-One, which indicates that the V2 training data is responsible for a substantial improvement in accuracy over V1. The full set of results are shown in Table **??**

- Convolutional recurrent neural networks for small-footprint keyword spotting [2]
- Honk: A PyTorch reimplementation of convolutional neural networks for keyword spotting [17]
- An experimental analysis of the power consumption of convolutional neural networks for keyword spotting [19]
- Transfer learning for speech recognition on a budget [12]
- Learning and transferring mid-level image representations using convolutional neural networks [14]
- Deep residual learning for small-footprint keyword spotting [18]

2 Method

The method is inspired by [10] where three different models have been evaluated on their capability to handle audio data transformed to images. One of the baseline models is the MNIST model which is also used in the Tensorflow Speech Recognition tutorial [1]

1. methodology, types of analyses, selection of the method

Xavier Glorot initialization [6]

Taken from [10]: For an $m \times x$ dimensional matrix M, $M_{i,j}$ is assigned values selected uniformly from the distribution $[-\epsilon, \epsilon]$, where

$$\epsilon = \frac{\sqrt{6}}{\sqrt{m+n}} \tag{1}$$

Xavier initialization is shown in equation 1

3 Set-up

3.1 Dataset

Write something about the Speech Recogntion Dataset and the class distribution before and after merging for the competition.

3.2 Preprocessing

The preprocessing pipeline is described in further detail in the following subsections. The pipeline is rather simple at the moment and provides room for improvement which is described in section 8. The overall preprocessing approach is depicted in figure 3.

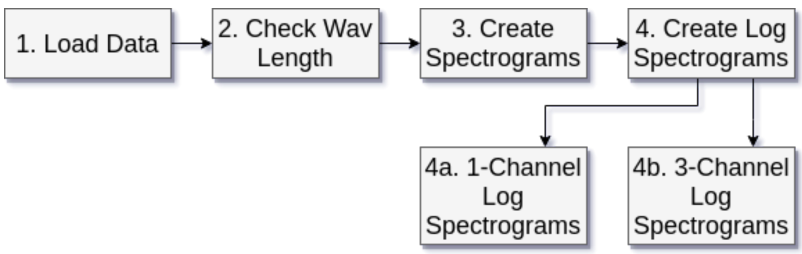


Figure 3: Preprocessing

3.2.1 Load Data

Based on the fact that there is no index file in a common format like csv which maps data entries to labels, the approach to loading the training data involves iterating labeled folders. The training folder contains one folder for each label which then contains the actual wav files for training purposes. By iterating through the different label folders and then copying the paths of the training files, a more conventient numpy array for training purposes is used. A subset of this array is depicted in table 2.

The different paths were iterated and the wav files were load into memory by using the `scipy.io.wavfile` package. The sample rate of the wav files remains at 160000 KHz.

Path	Label
'../data/train/audio/down/fad7a69a_nohash_1.wav'	'down'
'../data/train/audio/go/fa7895de_nohash_0.wav'	'go'
'../data/train/audio/left/fa7895de_nohash_0.wav'	'left'
'../data/train/audio/no/a1c63f25_nohash_2.wav'	'no'
'../data/train/audio/off/4a1e736b_nohash_4.wav'	'off'
'../data/train/audio/on/4a1e736b_nohash_4.wav'	'on'
'../data/train/audio/right/b71ebf79_nohash_0.wav'	'right'
'../data/train/audio/_background_noise_/doing_the_dishes.wav'	'silence'
'../data/train/audio/stop/fa7895de_nohash_0.wav'	'stop'
'../data/train/audio/two/fa7895de_nohash_0.wav'	'unknown'
'../data/train/audio/up/4c841771_nohash_2.wav'	'up'
'../data/train/audio/yes/b71ebf79_nohash_0.wav'	'yes'

Table 2: Training data - Numpy array representation

3.2.2 Check Wav Length

In order to have a consistent dataset with clips of one second length, the length of each wav file has been checked. Two approaches to guarantee one second clips have been applied:

- *length* < 1 *second*: Pad the clip with constant zeros
- *length* > 1 *second*: Cut the clip from the beginning to the one second mark

3.2.3 Create Spectrograms

To transform audio data into images, the $\mathbb{R}^{16000}$ audio vectors have been transformed into spectrograms. The code for the transformation is given in listing 1.

```python
from scipy import signal
from scipy.io import wavfile
import numpy as np

def get_spectrogram(audio_path, num_channels=1):
    (sample_rate, sig) = wavfile.read(audio_path)

    if sig.size < sample_rate:
        sig = np.pad(sig, (sample_rate - sig.size, 0), mode='constant')
    else:
        sig = sig[0:sample_rate]

    # f = array of sample frequencies
    # t = array of segment times
    # Sxx = Spectrogram of x. By default, the last axis of Sxx corresponds
     to the segment times.
    f, t, Sxx = signal.spectrogram(sig, nperseg=256, noverlap=128)
    Sxx = (np.dstack([Sxx] * num_channels)).reshape(129, 124, -1)

    return f, t, Sxx
```

Listing 1: Get spectrogram code

After a first inspection of the spectrograms given in figure 4, it is obvious that the spectrograms do not contain as much visible features as expected. Previous research [10] also suggested that using log spectrograms is more beneficial than using simple spectrograms.

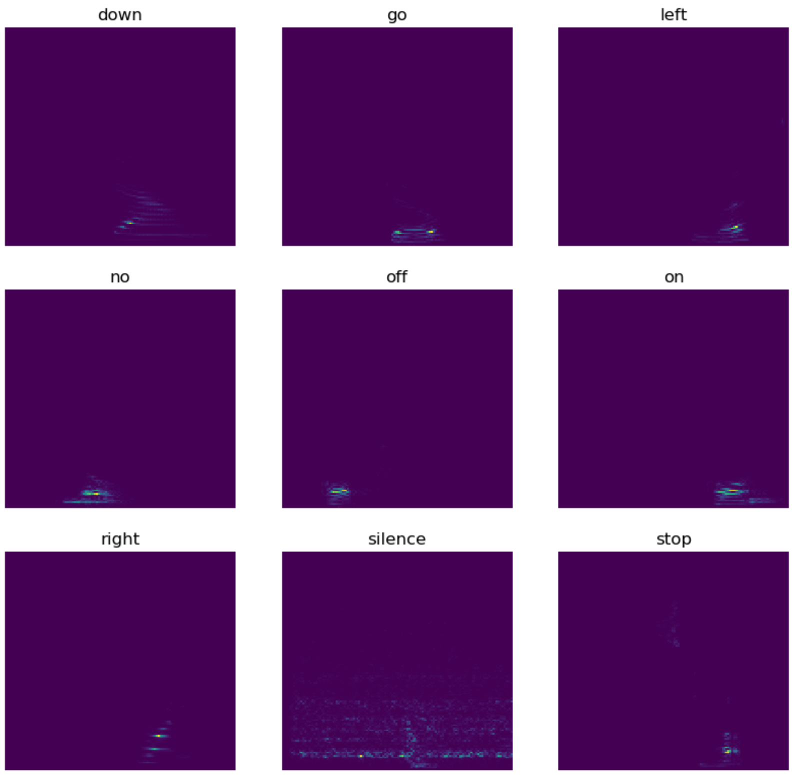


Figure 4: Spectrogram samples of nine different classes

3.2.4 Create Log Spectrograms

In contrast to figure 4, the code depicted in listing 2 produces log spectrograms which contain more visual features as seen in figure 5 which should be beneficial for the model training. One potential problem however is shown in figure 5 at the "silence" class. The padding with zeroes presents itself with a dark bar at the beginning which might introduce more noise into the dataset. Nevertheless, for this experiment this potential problem is ignored and could be tackled in future work.

```
from scipy import signal
from scipy.io import wavfile
import numpy as np

def get_log_spectrogram(audio_path, window_size=20, step_size=10, eps=1e-10, num_channels=1):
    (sample_rate, sig) = wavfile.read(audio_path)

    if sig.size < 16000:
        sig = np.pad(sig, (sample_rate - sig.size, 0), mode='constant')
    else:
        sig = sig[0:sample_rate]

```

```
    nperseg = int(round(window_size * sample_rate / 1e3))
    noverlap = int(round(step_size * sample_rate / 1e3))

    # f = array of sample frequencies
    # t = array of segment times
    # Sxx = Spectrogram of x. By default, the last axis of Sxx corresponds
     to the segment times.
    f, t, Sxx = signal.spectrogram(sig,
                                   fs=sample_rate,
                                   window='hann',
                                   nperseg=nperseg,
                                   noverlap=noverlap,
                                   detrend=False)
    log_spectrogram = np.log(Sxx.T.astype(np.float32) + eps)
    log_spectrogram = (np.dstack([log_spectrogram] * num_channels)).
    reshape(99, 161, -1)

    return f, t, log_spectrogram
```

Listing 2: Get log spectrogram code

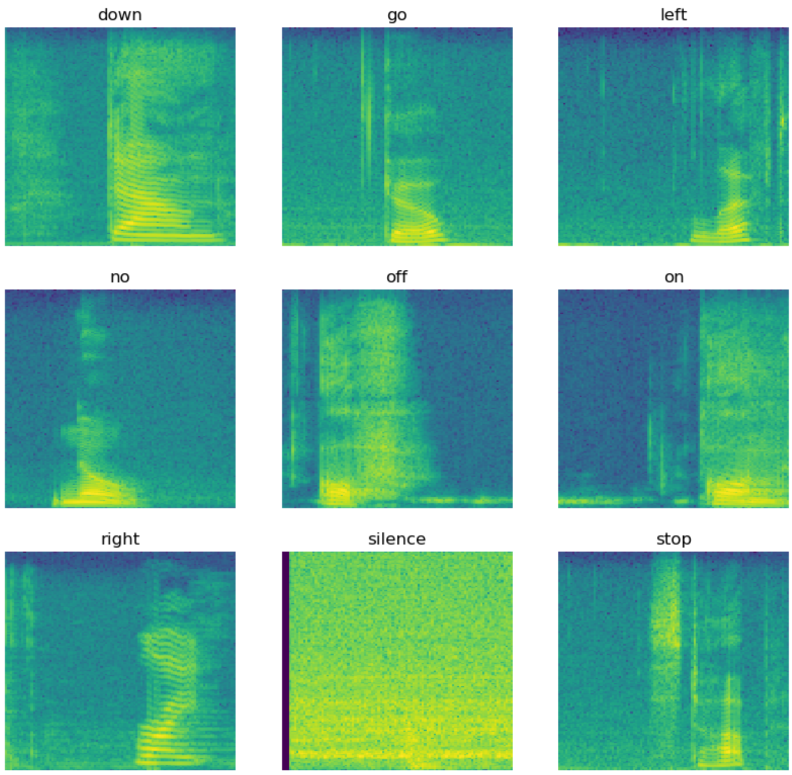
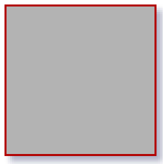
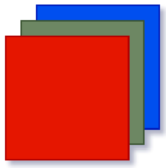


Figure 5: Log spectrogram samples of nine different classes

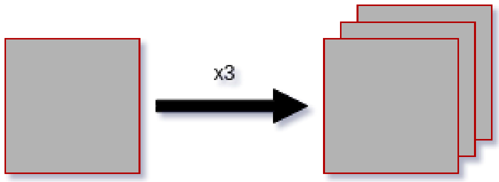
A distinction between 1- and 3-channel log spectrograms has been made because this project uses pre-trained CNN models which are trained on ImageNet data. ImageNet data is in its core based on 3-dimensional RGB data/images while (log) spectrograms are 1-dimensional images and are only depicted in green colors in figure 5 based on the settings in matplotlib which shows grayscale images in a green spectrum. A quick fix to this problem is the duplication of the 1-dimensional spectrogram data and therefore mimicking 3-dimensional RGB data by having the grayscale data copied over three channels as depicted in figure 6.



(a) 1-channel log spectrogram



(b) 3-channel RGB image



(c) Duplication of 1-channel log spectrogram

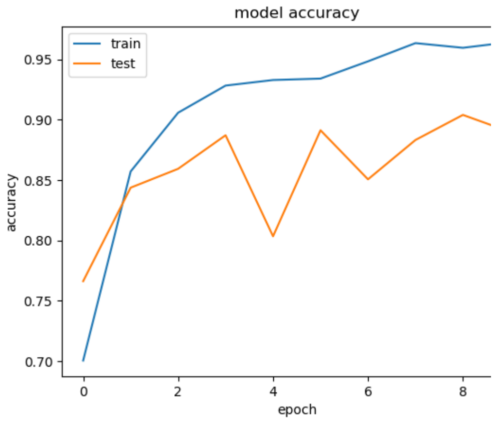
Figure 6: Conversion between 1-channel and 3-channel log spectrograms

3.3 Models

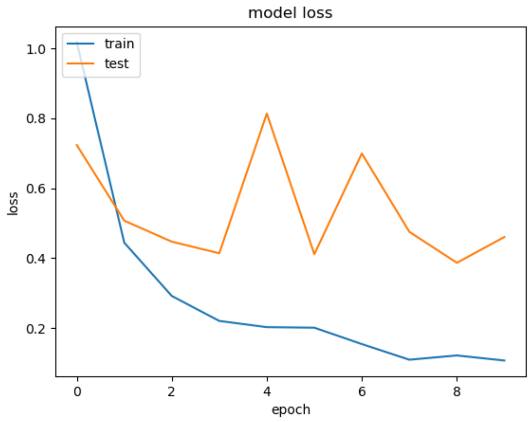
1. tuning/adaptation model parameters
2. types of experiments (generalizations to which unseen conditions, etc.)

Model	**Total params**	**Trainable params**	**Non-trainable params**
Leightweight CNN	723,968	723,454	514
VGG16	23,676,748	23,659,340	17,408
Inception V3	29,185,836	29,137,068	48,768
MNIST	55,038,988	54,929,868	109,120
ResNet50	75,182,988	75,029,516	153,472

Table 3: Model complexity ordered by amount of parameters

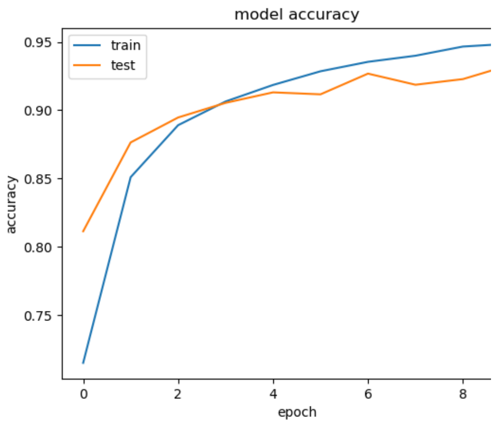


(a) Accuracy

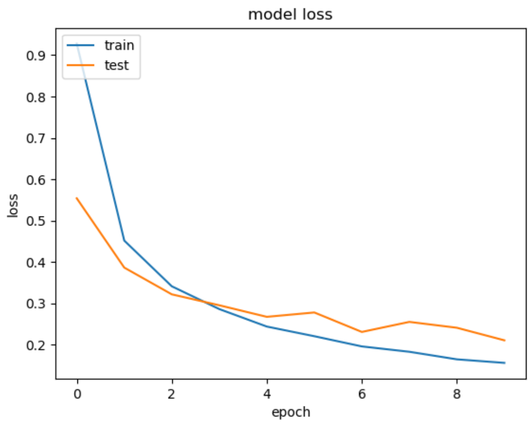


(b) Loss

Figure 7: Accuracy and loss after 10 epochs for the MNIST model with Xavier Glorot initialization



(a) Accuracy



(b) Loss

Figure 8: Accuracy and loss after 10 epochs for the Lightweight CNN model with Xavier Glorot initialization

Model	**Train Acc**	**Train Loss**	**Val Acc**	**Val Loss**	**Time (sec)**
MNIST	0.9595	0.1213	0.9039	0.3866	1064.72
Leightweight CNN	0.9487	0.1564	0.9332	0.2109	532.73

Table 4: Final baseline results with Xavier Glorot initialization

Model	**Train Acc**	**Train Loss**	**Val Acc**	**Val Loss**
MNIST	0.7005	1.0187	0.7662	0.7237
Leight CNN	0.7150	0.9276	0.8114	0.5540

Table 5: Baseline results after one epoch with Xavier Glorot initialization

4 Experiments

4.1 Baseline

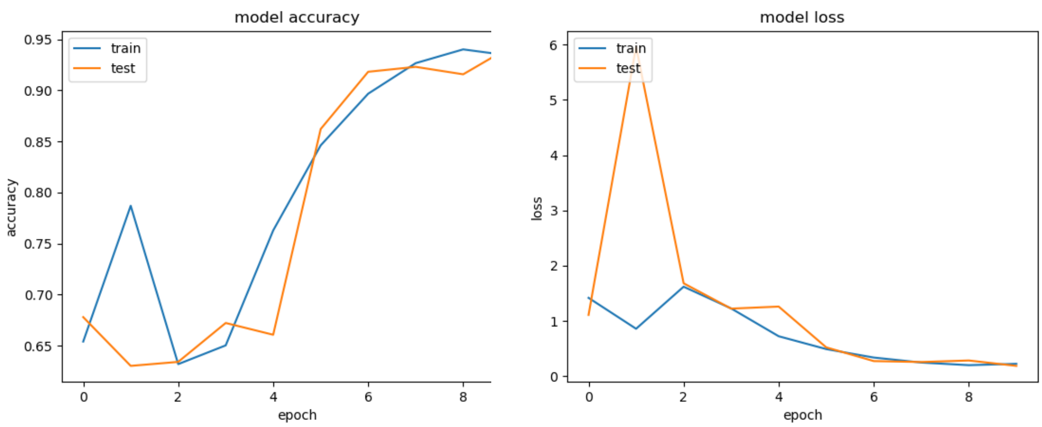
4.2 CNNs

5 Analysis and Results

5.1 Baseline

5.2 CNNs

5.2.1 Xavier initialization



(a) Accuracy

(b) Loss

Figure 9: Accuracy and loss after 10 epochs for the Inception V3 model with Xavier Glorot initialization

Model	Train Acc	Train Loss	Val Acc	Val Loss	Time (sec)
Inception V3	0.9338	0.2238	0.9427	0.1868	2332.23
VGG16	0.9723	0.0900	0.9583	0.2011	2530.47
ResNet50	0.9548	0.1421	0.9445	0.1873	3807.93

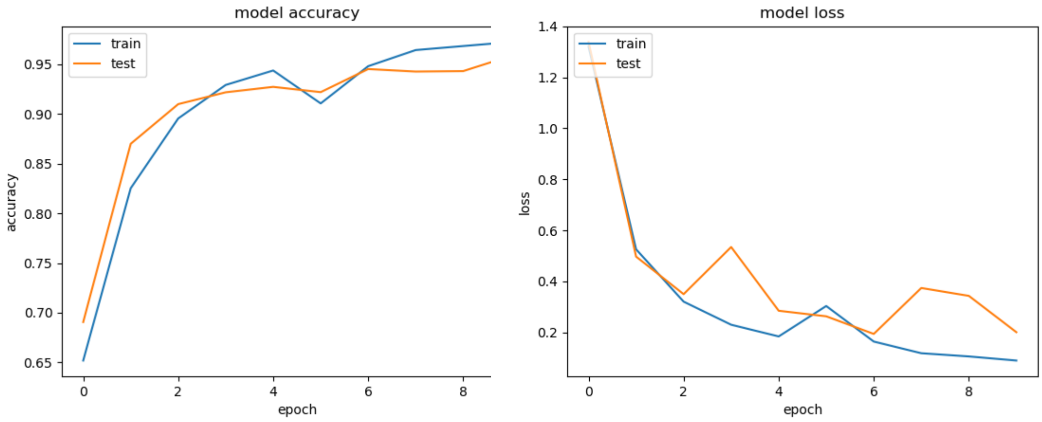
Table 6: Final CNN results with with Xavier Glorot initialization

Model	Train Acc	Train Loss	Val Acc	Val Loss
Inception V3	0.6539	1.4176	0.6778	1.1090
VGG16	0.6519	1.3202	0.6906	1.3379
ResNet50	0.6916	1.2478	0.6351	5.7467

Table 7: CNN results after one epoch with Xavier Glorot initialization

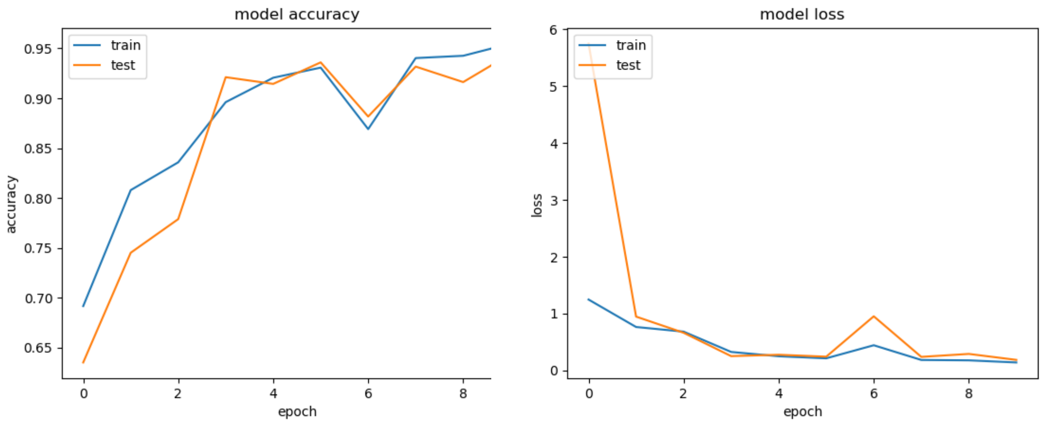
5.2.2 Imagenet weight initialization

Test



(a) Accuracy (b) Loss

Figure 10: Accuracy and loss after 10 epochs for the VGG16 model with Xavier Glorot initialization

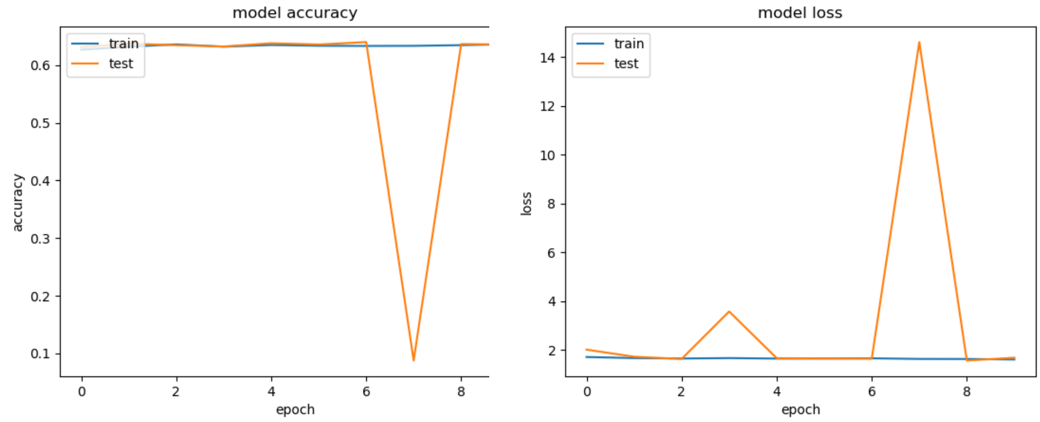


(a) Accuracy (b) Loss

Figure 11: Accuracy and loss after 10 epochs for the ResNet50 model with Xavier Glorot initialization

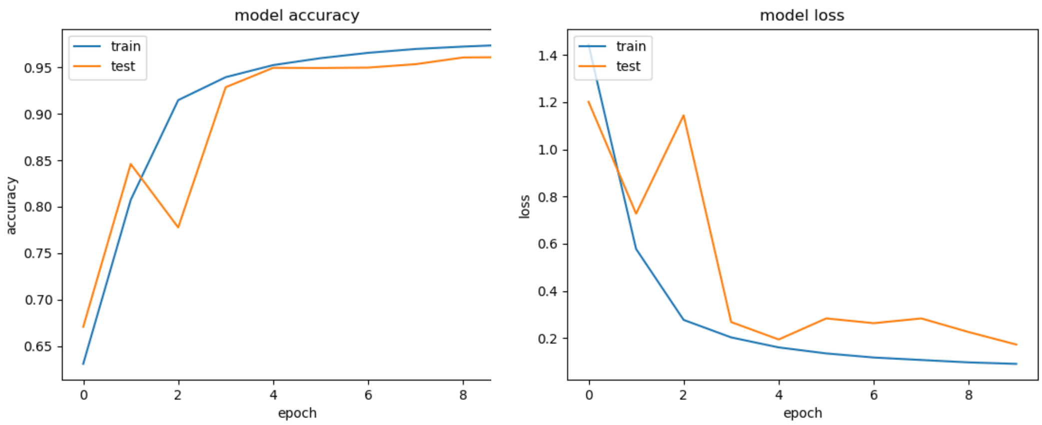
Model	**Train Acc**	**Train Loss**	**Val Acc**	**Val Loss**	**Time (sec)**
Inception V3	0.6334	1.6597	0.6401	1.6433	2184.54
VGG16	0.9745	0.0912	0.9611	0.1730	2541.83
ResNet50	0.9134	0.2991	0.9252	0.3055	3653.67

Table 8: Final CNN results with Imagenet initialization



(a) Accuracy (b) Loss

Figure 12: Accuracy and loss after 10 epochs for the Inception V3 model with Imagenet initialization



(a) Accuracy (b) Loss

Figure 13: Accuracy and loss after 10 epochs for the VGG16 model with Imagenet initialization

Model	**Train Acc**	**Train Loss**	**Val Acc**	**Val Loss**
Inception V3	0.6268	1.7132	0.6315	2.0153
VGG16	0.6307	1.4430	0.6706	1.2011
ResNet50	0.6389	1.4214	0.5931	5.1381

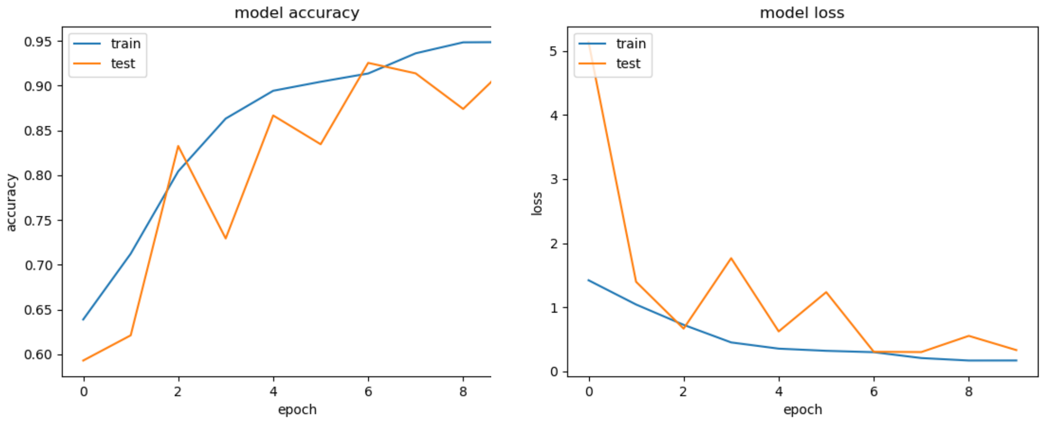
Table 9: CNN results after one epoch with Imagenet initialization

6 Discussion

7 Conclusion

8 Future Work

- Use other image representations like spectrograms based on MFCC



(a) Accuracy

(b) Loss

Figure 14: Accuracy and loss after 10 epochs for the ResNet50 model with Imagenet initialization

- Use recursive layer freezing to investigate which amount of layers would be optimal for transfer learning
- Use additional preprocessing steps like discarding bad audio
- Look further into Batch Normalization
- Look into correlation between accuracy and dimensions of spectrograms

9 References

10 Appendix

- the experiment(s) may be carried out in collaboration with others. In that case: specify in the "author's statement" everybody's contribution
- the thesis itself is written individually and assessed individually
- the ASR performance itself is not relevant for the assessment of the thesis
- the RQ, the literature embedding of the RQ, the description of the method, the justification and set-up of the experiment are relevant for the assessment
- the general university guidelines apply (e.g., with respect to plagiarism)
- there is no minimum number of pages for the thesis

aspect	experimental (max. points)	theoretical (max. points)
Research Question (RQ)	20	20
Literature embedding of the RQ	20	40
Method	20	
Justification experiment(s)	10	
Set-up experiment(s)	30	
Discussion and Conclusion	30	70
Use of figures and tables	10	10
Overall completeness	20	20
Overall clarity, transparency	20	20
Overall coherence (from intro to conclusion)	20	20
Total	*200*	*200*

Figure 15: Weighted grading

References

[1] Tensorflow speech recognition tutorial. `https://www.tensorflow.org/tutorials/sequences/audio_recognition`. Accessed: 2019-08-05.

[2] Sercan O Arik, Markus Kliegl, Rewon Child, Joel Hestness, Andrew Gibiansky, Chris Fougner, Ryan Prenger, and Adam Coates. Convolutional recurrent neural networks for small-footprint keyword spotting. *arXiv preprint arXiv:1703.05390*, 2017.

[3] Guoguo Chen, Carolina Parada, and Georg Heigold. Small-footprint keyword spotting using deep neural networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4087–4091. IEEE, 2014.

[4] Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. *arXiv preprint arXiv:1202.2745*, 2012.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[6] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[8] Speech commands dataset version 1. `http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz`. Accessed: 2019-08-05.

[9] Speech commands dataset version 2. `http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz`. Accessed: 2019-08-05.

[10] Sanjay Krishna Gouda, Salil Kanetkar, David Harrison, and Manfred K Warmuth. Speech recognition: Keyword spotting through image recognition. *arXiv preprint arXiv:1803.03759*, 2018.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[12] Julius Kunze, Louis Kirsch, Ilia Kurenkov, Andreas Krug, Jens Johannsmeier, and Sebastian Stober. Transfer learning for speech recognition on a budget. *arXiv preprint arXiv:1706.00290*, 2017.

[13] Jongpil Lee, Taejun Kim, Jiyoung Park, and Juhan Nam. Raw waveform-based audio classification using sample-level cnn architectures. *arXiv preprint arXiv:1712.00866*, 2017.

[14] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.

[15] George Retsinas, Giorgos Sfikas, and Basilis Gatos. Transferable deep features for keyword spotting. In *Multidisciplinary Digital Publishing Institute Proceedings*, volume 2, page 89, 2018.

[16] Tara Sainath and Carolina Parada. Convolutional neural networks for small-footprint keyword spotting. 2015.

[17] Raphael Tang and Jimmy Lin. Honk: A pytorch reimplementation of convolutional neural networks for keyword spotting. *arXiv preprint arXiv:1710.06554*, 2017.

[18] Raphael Tang and Jimmy Lin. Deep residual learning for small-footprint keyword spotting. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5484–5488. IEEE, 2018.

[19] Raphael Tang, Weijie Wang, Zhucheng Tu, and Jimmy Lin. An experimental analysis of the power consumption of convolutional neural networks for keyword spotting. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5479–5483. IEEE, 2018.

[20] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.