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 19, 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 [20]

1.1.2 Transferable deep features for keyword spotting

- Transferable deep features for keyword spotting [22]

1.1.3 Imagenet: A large-scale hierarchical image database

- Imagenet: A large-scale hierarchical image database [8]

1.1.4 Imagenet classification with deep convolutional neural networks

- Imagenet classification with deep convolutional neural networks [18]

1.1.5 Speech Recognition: Keyword Spotting Through Image Recognition.

The authors of the paper "Speech Recognition: Keyword Spotting Through Image Recognition" [14] transformed the KWS task which incoporates audio data into the domain of image classification. They used the Speech Commands Dataset [30] which contains spoken words of the length of one second in order to train and evalutate their model. According to [30], the Speech Commands Dataset V2 [13] 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 [23] 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 [23] (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 [6] and AlexNet [18]. 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 [11].

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

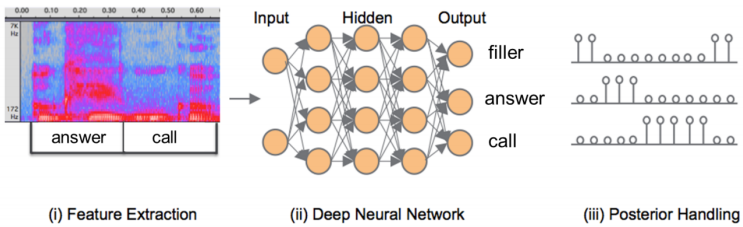


Figure 2: Framework of Deep KWS system, components from left to right: (i) Feature Extraction (ii) Deep Neural Network (iii) Posterior Handling

This framework originally comes from [5]. The only difference is the exchange of the DNN for a CNN.

- Convolutional neural networks for small-footprint keyword spotting [23]

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 [5]

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 [30]

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.

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 [28] shows how approaches learned from ResNet can produce more efficient and accurate models. Raw Waveformbased Audio Classification [20] investigates alternatives to traditional feature extraction for speech and music models. Keyword Spotting through Image Recognition [14] 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

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

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 [12] 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 [23]) 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 [13], 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 [4]
- Honk: A PyTorch reimplementation of convolutional neural networks for keyword spotting [27]
- An experimental analysis of the power consumption of convolutional neural networks for keyword spotting [29]
- Transfer learning for speech recognition on a budget [19]
- Learning and transferring mid-level image representations using convolutional neural networks [21]
- Deep residual learning for small-footprint keyword spotting [28]

2 Method

The method is inspired by [14] 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 [3]

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

Xavier Glorot initialization [10]

Taken from [14]: 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

The TensorFlow Speech Recognition Challenge hosted by Kaggle [1] is using the Speech Commands Data Set v0.01 and contains 64,727 audio files and 31 class labels. The data set is publicly available [12] and is explained in more detail by Warden [30]. The characteristics of the original v0.01 data set are explained as follows by Warden:

> 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.

The original class distribution of 31 words had to be reduced to 12 words to comply with the Kaggle competition guidelines, namely 10 concrete words `yes, no, up, down, left, right, on, off, stop, go` and two placeholder words `unknown, silence`. Words which do not belong to the 10 concrete words are merged into the `unknown` class label, while background noise and simple silence are merged into the `silence` class label. Provided files are further explained on the competition page [1] as follows:

- **train.7z**: Contains a few informational files and a folder of audio files. The audio folder contains subfolders with 1 second clips of voice commands, with the folder name being the label of the audio clip. There are more labels that should be predicted. The labels you will need to predict in Test are `yes, no, up, down, left, right, on, off, stop, go`. Everything else should be considered either `unknown` or `silence`. The folder `_background_noise_` contains longer clips of "silence" that you can break up and use as training input. The files contained in the training audio are not uniquely named across labels, but they are unique if you include the label folder. For example, `00f0204f_nohash_0.wav` is found in 14 folders, but that file is a different speech command in each folder. The files are named so the first element is the subject id of the person who gave the voice command, and the last element indicated repeated commands. Repeated commands are when the subject repeats the same word multiple times. Subject id is not provided for the test data, and you can assume that the majority of commands in the test data were from subjects not seen in train. You can expect some inconsistencies in the properties of the training data (e.g., length of the audio).
- **test.7z**: Contains an audio folder with 150,000+ files in the format `clip_000044442.wav`. The task is to predict the correct label. Not all of the files are evaluated for the leaderboard score.
- **sample_submission.csv**: A sample submission file in the correct format.

After merging words which do not comply with the competition guidelines, the class distribution changed from a balanced distribution to a rather unbalanced distribution towards the "unknown" label as depicted in table 2

Class	Frequency	Percentage
unknown	'41039	0.634032
stop	2380	0.036770
yes	2377	0.036723
up	2375	0.036693
no	2375	0.036693
go	2372	0.036646
right	2367	0.036569
on	2367	0.036569
down	2359	0.036445
off	2357	0.036414
left	2353	0.036353
silence	6	0.000093

Table 2: Descending class distribution of merged data set

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. A Voice-activity detection (VAD) system has not been used based on the simple nature of the one-second audio clips.

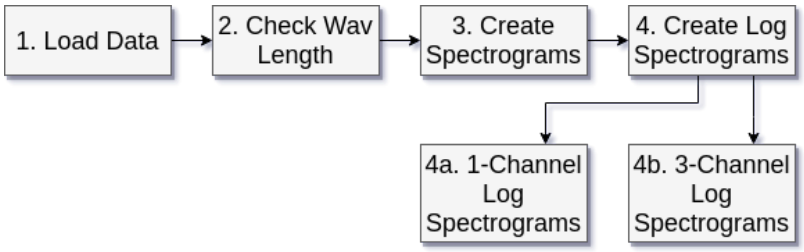


Figure 3: Preprocessing

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 3.

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 16000 KHz. A reduction to a sampling rate of 8000 KHz would also be possible to reduce the used memory space for the final training set.

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 3: 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.

```
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 [14] also suggested that using log spectrograms is more beneficial than using simple spectrograms. Spectrograms were reshaped into `129 x 124` dimensions which differs

from the log spectrograms but does not influence the experiment in any way because solely log spectrograms were used for training purposes.

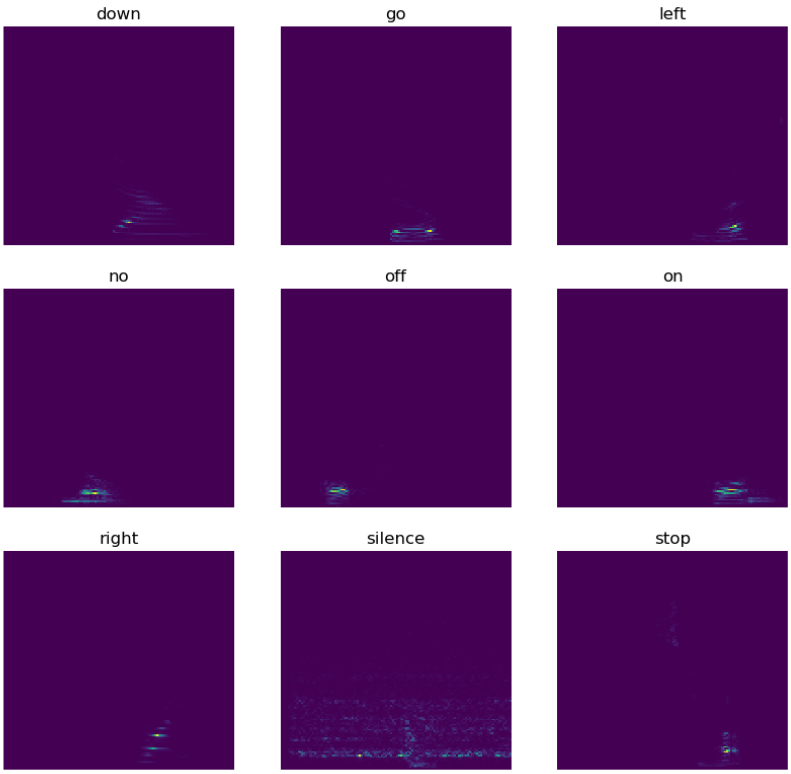


Figure 4: Spectrogram samples of nine different classes

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. Log spectrograms were reshaped into `99 x 161` dimensions.

```
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

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 the `matplotlib` package 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.

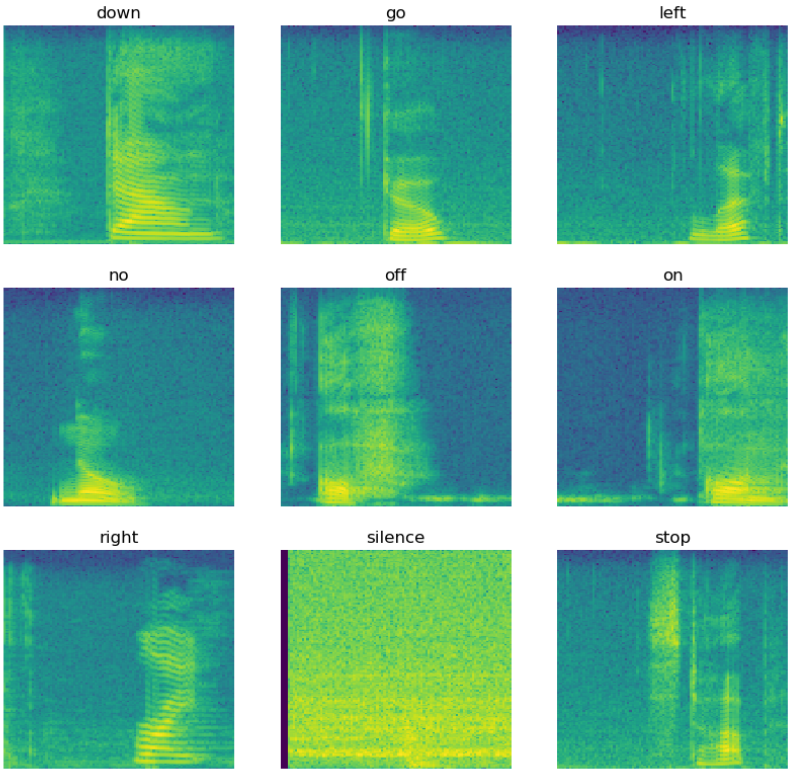
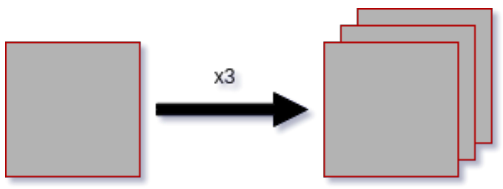


Figure 5: Log spectrogram samples of nine different classes

(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

The choice of models can be divided into the following groups: **Baseline** and **CNN models with pre-trained ImageNet weights**. Table 4 shows an overview of the evaluated models in this project with the complexity of each model based on its amount of trainable parameters.
MNIST and the Lightweight CNN model are used as a baseline because they showed good performance in previous work when applied to spectrograms. The other models, namely VGG16, Inception V3 and ResNet were used because they each won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in their respective years and differ in their architecture. The baseline models were initialized with Xavier initialization and their performances were not evaluated with ImageNet weight initialization.

Model	**Total params**	**Trainable params**	**Non-trainable params**
Lightweight 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 4: Model complexity ordered by amount of parameters

Each model has been trained over 10 epochs with either Xavier Glorot initialization while the CNN models have also been using pre-trained ImageNet weights while all layers remained trainable which is against traditional transfer learning techniques where a certain amount of layers are not trainable any more (freezing layers). After 10 epochs the final accuracy has been evaluated in order to see if pre-trained ImageNet weights would give a boost to the convergence rate. Another evaluation has been made after the first epoch with regards to accuracy to see if pre-trained ImageNet weights would give a higher start accuracy based on the already learned image features from another domain than the provided spectrograms, namely general images in the ImageNet dataset.

All models used Rectified Linear Unit as depicted in figure 7 as their activation function due to its success in the domain of image classification [7]. However, Softmax is applied as the final activation function in all output layers in order get proper probabiliy scores.

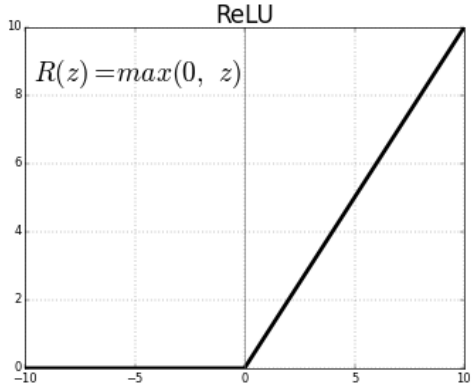
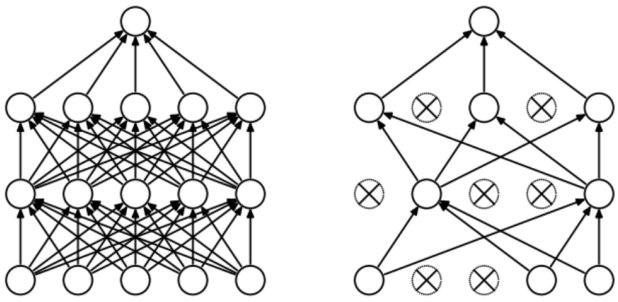


Figure 7: Rectified Linear Unit as activation function

Throughout all used architectures Dropout [25] with a value of 0.2 has been used as a regularization technique as depicted in figure 9 and as proposed in [7].



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

Figure 8: Dropout as generalization technique

Batch normalization as depicted in figure 9 and proposed in [16] has been applied to all architectures due to the memory space consumption of the training set and the need to use a batch size of 32. Batch normalization showed good results in boosting the overall training process of feedforward neural networks with fewer training steps, acts as a additional regularization technique and reduces internal covariate shift by normalizing each batch respectively.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 9: Batch normalization to speed up training and reducing internal covariate shift

The following sections include descriptions of the architecture and special characteristics of each model.

3.3.1 Baseline models

MNIST

The chosen architecture of the so called "MNIST" model is burrowed from the TensorFlow repository [2] and showed an accuracy of 98% on the task for grayscale digit recognition [9]. In order to guarantee consistent comparison conditions, Batch normalization layers have been added. The MNIST architecture therefore looks as follows:

1. 2D Convolutional layer with 32 features, kernel size of 5×5 and ReLU activation function.
2. MaxPooling layer with a pool size of 2×2
3. Batch normalization layer with $momentum = 0.99, \epsilon = 0.001$
4. 2D Convolutional layer with 64 features, kernel size of 3×3 and ReLU activation function.
5. MaxPooling layer with a pool size of 2×2
6. Dropout layer with $p = 0.2$
7. Batch normalization layer with $momentum = 0.99, \epsilon = 0.001$
8. Dense layer with 1024 neurons and ReLU activation
9. Dropout layer with $p = 0.2$
10. Batch normalization layer with $momentum = 0.99, \epsilon = 0.001$
11. Dense layer with 12 neurons (number of classes) and Softmax

Lightweight CNN

The "Lightweight CNN" has been burrowed from the Kaggle competition in order to have another baseline modes which is known to perform on a solid basis with spectrograms [1]. The main architecture looks as follows:

1. Batch normalization layer with $momentum = 0.99, \epsilon = 0.001$
2. 2x 2D Convolutional layer with 8 features, kernel size of 2×2 and ReLU activation function.
3. MaxPooling layer with a pool size of 2×2
4. Dropout layer with $p = 0.2$
5. 2x 2D Convolutional layer with 16 features, kernel size of 3×3 and ReLU activation function.
6. MaxPooling layer with a pool size of 2×2
7. Dropout layer with $p = 0.2$ 2D Convolutional layer with 32 features, kernel size of 3×3 and ReLU activation function.
8. MaxPooling layer with a pool size of 2×2
9. Dropout layer with $p = 0.2$
10. Batch normalization layer with $momentum = 0.99, \epsilon = 0.001$
11. Dense layer with 128 neurons and ReLU activation function
12. Batch normalization layer with $momentum = 0.99, \epsilon = 0.001$
13. Dense layer with 128 neurons and ReLU activation function
14. Dense layer with 12 neurons (number of classes) and Softmax

[1] `https://www.kaggle.com/alphasis/light-weight-cnn-lb-0-74`

3.3.2 CNN models

The top layers of each CNN model have been removed in order to take other input tensors than the usual ones found in the ImageNet database which are `224x224` pixels to fit the log spectrogram dimensions. The here called "standard top layer configuration" has been stacked at the top of every CNN model in order to have a consistent configuration and to work with log spectrograms. The standard top layer configuration looks as follows:

1. Dropout layer with $p = 0.2$
2. Batch normalization layer with $momentum = 0.99, \epsilon = 0.001$
3. Dense layer with 1024 neurons and ReLU activation function
4. Batch normalization layer with $momentum = 0.99, \epsilon = 0.001$
5. Dense layer with 1024 neurons and ReLU activation
6. Dense layer with 12 neurons (number of classes) and Softmax activation function

VGG16

The VGG architecture has been proposed in 2014 and incorporates increasing depth using an architecture with very small (3×3) convolution filters and a depth of 16 to 19 weight layers [24]. The architecture [2] depicted in figure 10 also won ImageNet Challenge in 2014.

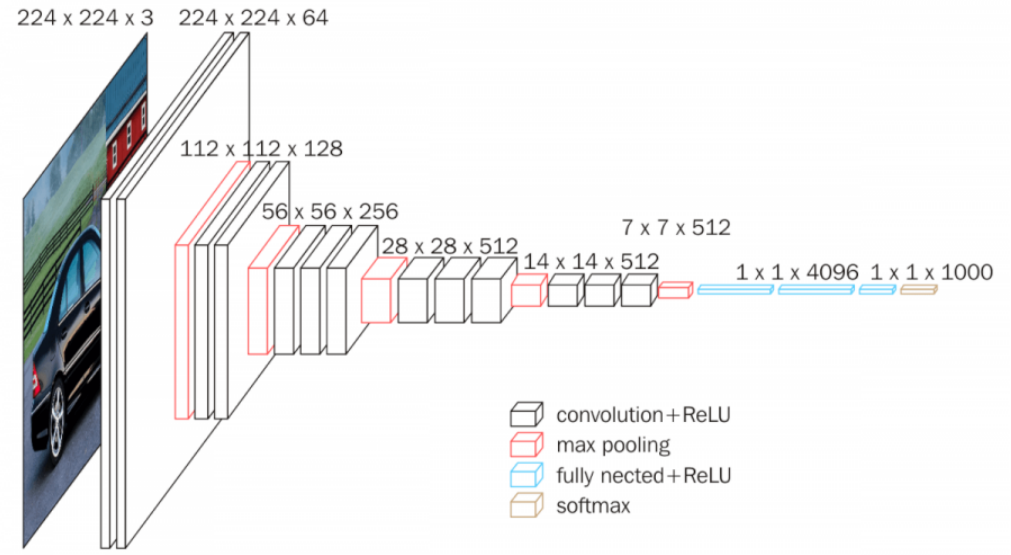


Figure 10: VGG16 architecture

Because VGG16 is such a commonly used CNN model in the domain of image classification and used in prior work with keyword spotting, it is also used in this project as the default CNN. Furthermore, Keras also kindly provides the Python code and the pre-trained ImageNet weights for this model.

[2] `https://neurohive.io/en/popular-networks/vgg16/`

Inception V3

The Inception V3 architecture is the winner of the ImageNet Challenge of 2015 and outperformed the VGG architecture [26]. As depicted in figure 11, this architecture is 42 layers deep while the computation cost is only about 2.5 higher than that of GoogLeNet, and much more efficient than that of VGG [3]. This efficiency is accomplished by using an efficient grid size reduction compared to a too greedy max pooling operation in the convolutional layer like VGG16.

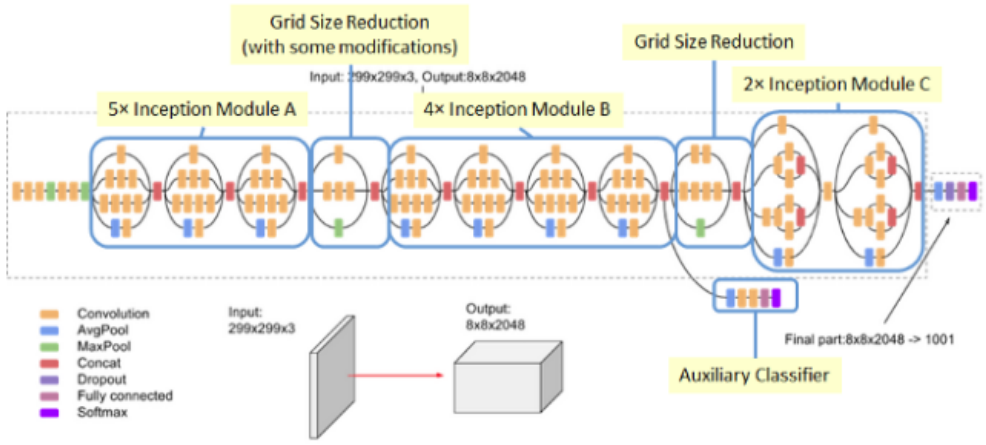


Figure 11: Inception-v3 Architecture (Batch Norm and ReLU are used after Conv)

In order to draw good conlusions with regards to the benefits of the ImageNet initialization while being model architecture agnostic at the same time, Inception V3 differs from the VGG16 model sufficiently to use it as the second CNN model. Furthermore, Keras also kindly provides the Python code and the pre-trained ImageNet weights for this model.

[3] https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification\-in-ilsvrc-2015-17915421f77c

ResNet50

Deep residual networks outperformed the VGG architecture in the task of image classification by being eight times deeper than VGG nets but still having lower complexity [15]. So called "Skip connections" which are the building blocks of residual learning can be used to skip the training of a few layers as depicted in figure 12.

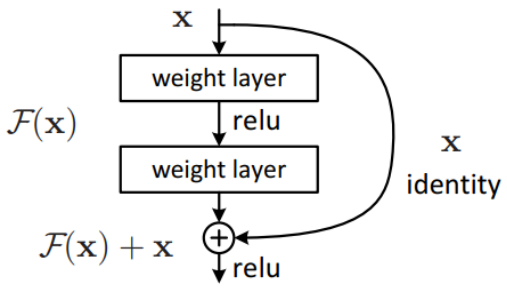


Figure 12: Residual learning: a building block.

Deep residual learning has also been applied to the domain of small-footprint keyword spotting and outperformed classical CNN architectures [28] which makes it a good fit for this project. Keras standard "ResNet50" model has been used with the standard top layer configuration.

4 Experiments

The main factors in this experiment are the accuracy and loss metrics for both the train and validation set after (i) the first epoch and (ii) after the last epoch which is set to 10 for all models.

i The inspection of the first epoch should show if there are higher accuracy/lower loss values at the beginning of the training process depending which initialization strategy was chosen

ii The inspection of the last epoch should show if there are higher accuracy/lower loss values at the end of the training process and if the convergence rate is higher towards a maximum depending which initialization strategy was chosen

The first part of the experiment focuses on the baseline models, namely MNIST and the lightweight CNN. These models are not initialized with pre-trained ImageNet weights but solely with Xavier Glorot initialization. The baseline models serve as a starting point with regards to accuracy and loss values.

The second part of the experiment focuses on the CNN models for which pre-trained ImageNet weights exist. The models are evaluated with regards to point (i) and (ii) with Xavier Glorot Initialization and with pre-trained ImageNet weights.

The results of the first part are used for evaluationg the overall performance of the CNN models while the second part serves to answer the main research question if pre-trained ImageNet weights are beneficial when it comes to the task of speech recognition on spectrograms. All other parameters which are descibed in section 3.3 remain the same.

5 Analysis and Results

The following sections contain the results of the previously described experimental setup.

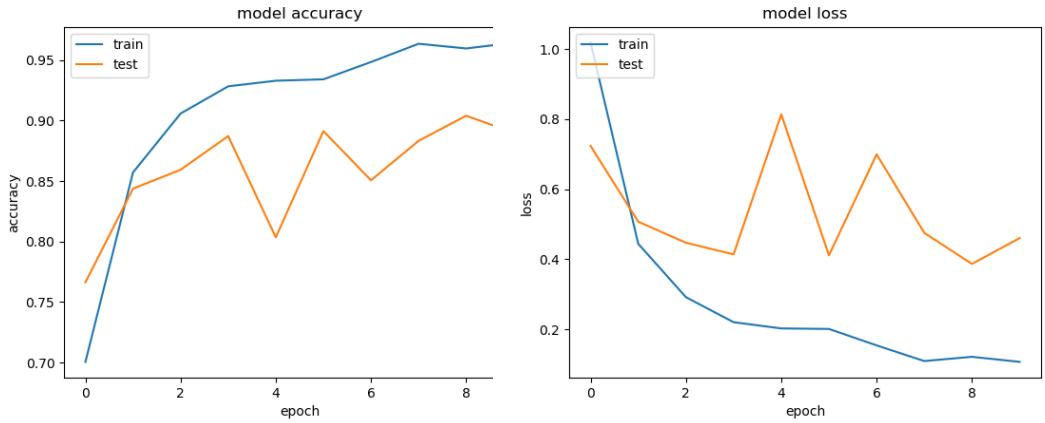
5.1 Baseline

According to table 5, the initial values of the MNIST and lightweight CNN model do not differ significantly while the lightweight CNN performs slighty better on both the training and validation set. The training accuracy is $\approx 70\%$ for both models at the beginning and should serve as the baseline accuracy value for later comparison.

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

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

During the training process a clear convergence towards a maximum is evident in both the MNIST and lightweight CNN model as depicted in figure 13 and 14 respectively.



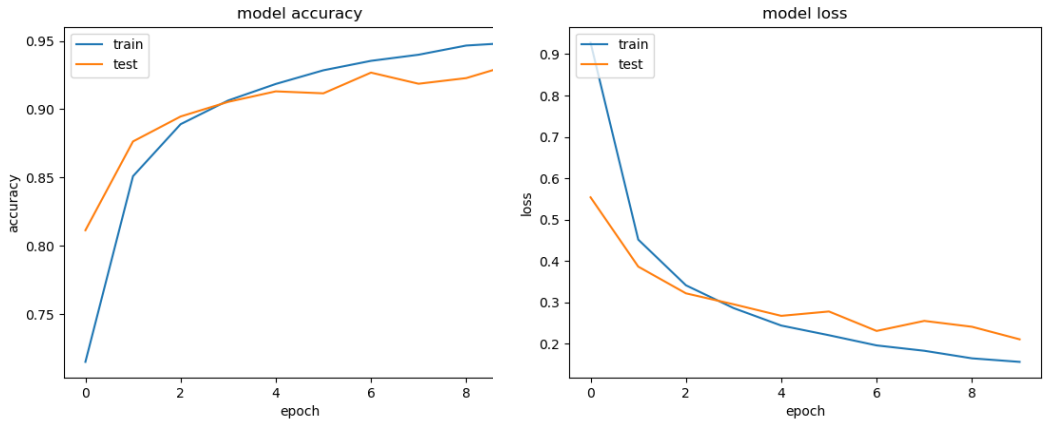
(a) Accuracy (b) Loss

Figure 13: Accuracy and loss after 10 epochs for the MNIST 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 6: Final baseline results with Xavier Glorot initialization

The final training accuracies of both models also do not differ significantly according to table 6 and are $\approx 95\%$.



(a) Accuracy (b) Loss

Figure 14: Accuracy and loss after 10 epochs for the Lightweight CNN model with Xavier Glorot initialization

5.2 CNNs

The following subsections contain the results of the three different CNN models, namely VGG16, Inception V3 and ResNet50 with the Xavier Glorot initialization and the pre-trained ImageNet weights initialization.

5.2.1 Xavier initialization

All three CNN models lay in the same range of initial accuracy values which is between 65% and 70% but still is below the initial values of the baseline models. Resnet performed the best in the initial run from all three models with about 4 % higher accuracy than VGG16 and Inception V3.

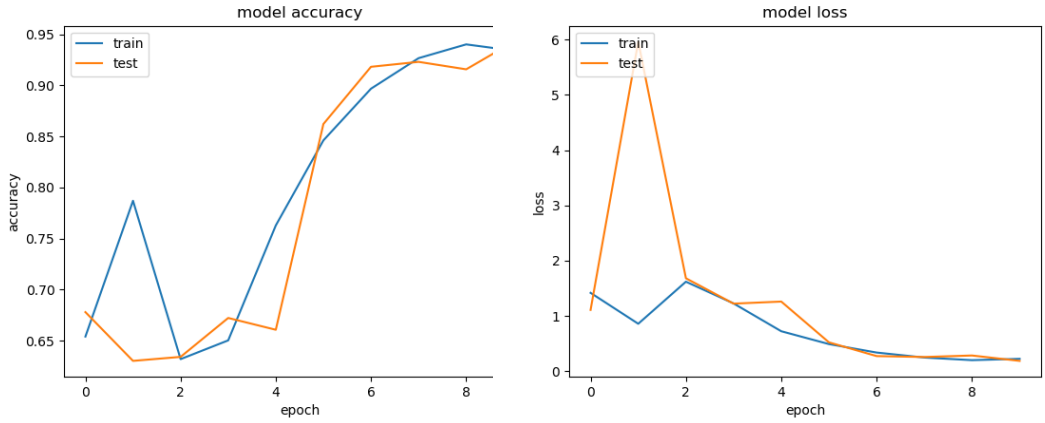
Model	**Train Acc**	**Train Loss**	**Val Acc**	**Val Loss**
Inception V3	0.6539	1.4176	0.6778	1.1090
VGG16	0.6519	1.3202	0.6906	1.3379
ResNet50	0.6916	1.2478	0.6351	5.7467

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

When it comes to the final accuracy values as depicted in table 8, only Inception V3 performed worse than the baseline models while ResNet performed equally well as MNIST . VGG16 reached the highest training accuracy with about 97%. So, VGG16 started with the lowest initial training accuracy but achieved the highest training accuracy at the end of 10 epochs from the three CNN models.
According to figure 15, Inception V3 started with some heavy fluctuation in the beginning until the third epoch and then converged towards a maximum with regards to accuracy.

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

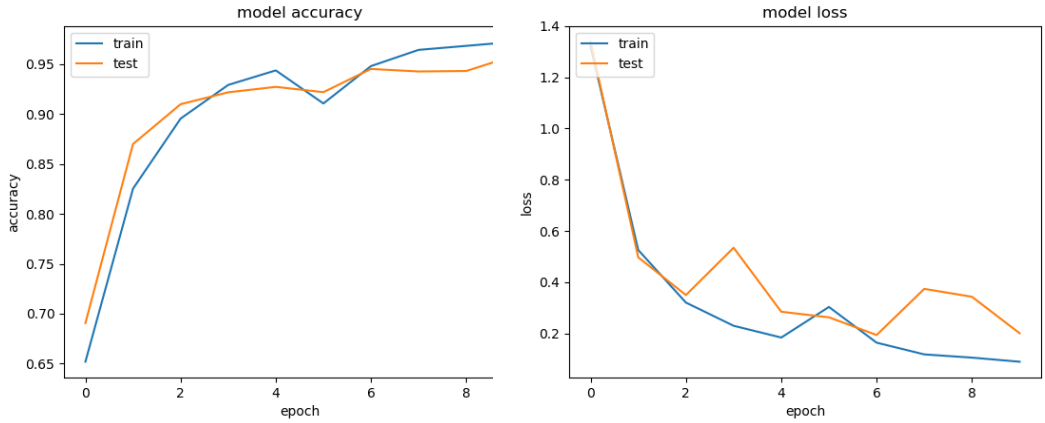
Table 8: Final CNN results with with Xavier Glorot initialization



(a) Accuracy (b) Loss

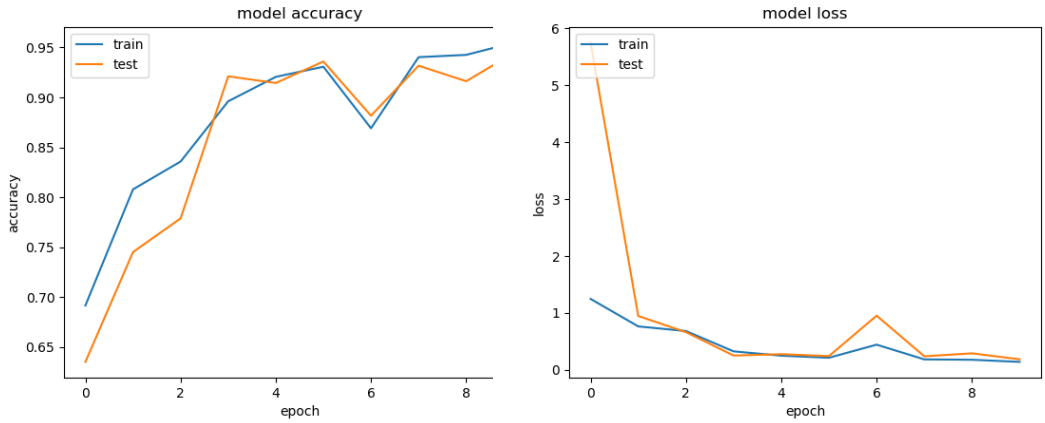
Figure 15: Accuracy and loss after 10 epochs for the Inception V3 model with Xavier Glorot initialization

VGG16 and ResNet converged more smoothly in contrast to Inception V3 when it comes to accuracy values and did not encounter fluctuation as Inception V3 as depicted in figure 16 and 17 respectively.



(a) Accuracy (b) Loss

Figure 16: Accuracy and loss after 10 epochs for the VGG16 model with Xavier Glorot initialization



(a) Accuracy (b) Loss

Figure 17: Accuracy and loss after 10 epochs for the ResNet50 model with Xavier Glorot initialization

5.2.2 Imagenet weight initialization

The initial values of all three CNN models initialized with the pre-trained ImageNet weights lay around 62% and 63% depicted in table 9. These values are worse than the baseline model values shown in table 5 and the CNN models which are initialized with Xavier Glorot initialization shown in table 7.

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

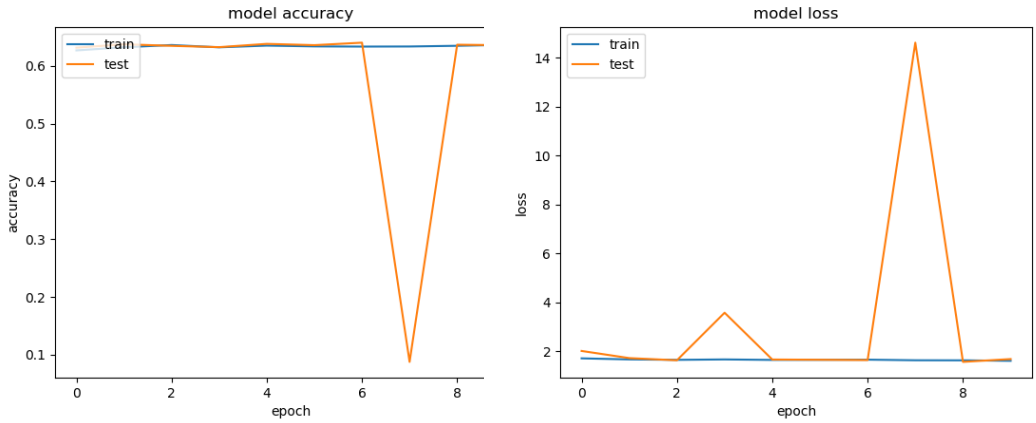
Table 9: CNN results after one epoch with Imagenet initialization

The final accuracy values as shown in table 10 show that VGG16 performed the best again while having a slightly higher accuracy value than the Xavier Glorot initialization strategy after 10 epochs. Inception V3 performed almost 30% worse according to the final accuracy value compared to the Xavier Glorot initialization. Figure 18 shows that there is nearly no training progress involved during the 10 epochs and that the accuracy value is constant at around 63% which makes it the worst performing model initialized with pre-trained ImageNet weights.

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

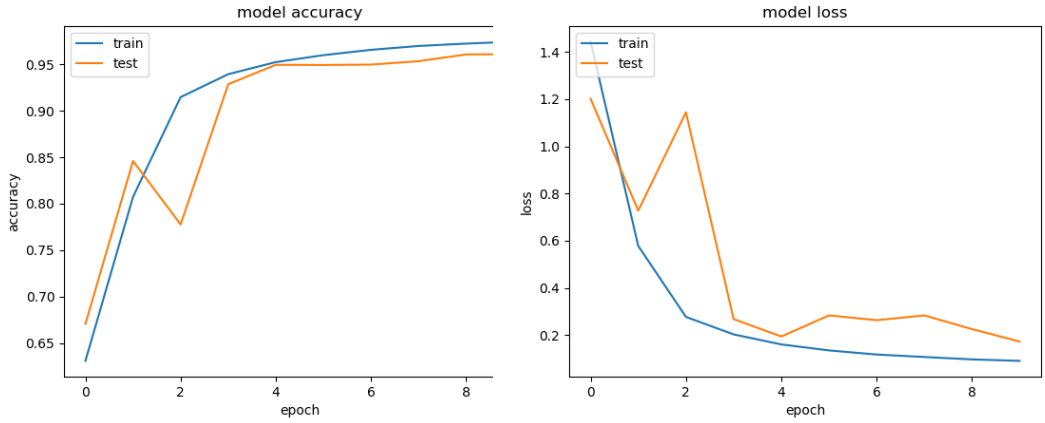
Table 10: Final CNN results with Imagenet initialization

ResNet scored 4% lower in terms of accuracy after 10 epochs when initialized with pre-trained ImageNet weights, but both VGG16 and ResNet showed a convergence towards a maximum in terms of accuracy depicted in figure 19 and 20 respectively in contrast to Inception V3.



(a) Accuracy (b) Loss

Figure 18: Accuracy and loss after 10 epochs for the Inception V3 model with Imagenet initialization



(a) Accuracy (b) Loss

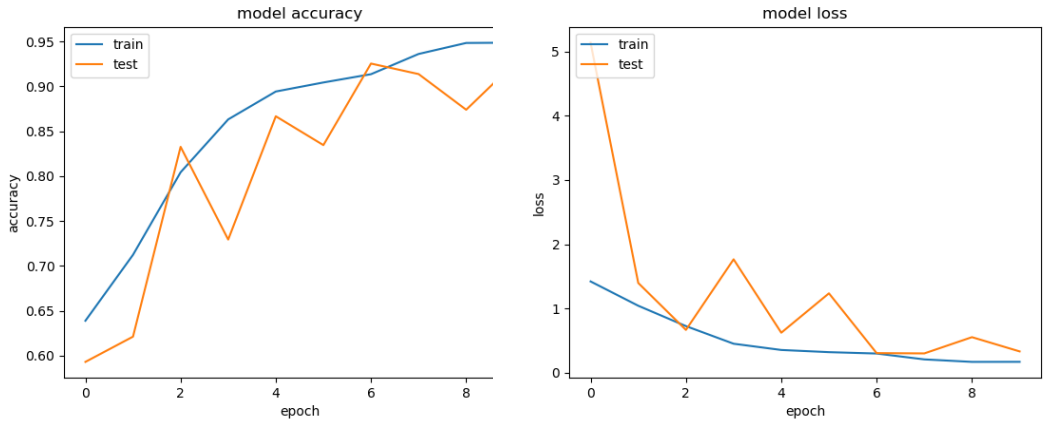
Figure 19: Accuracy and loss after 10 epochs for the VGG16 model with Imagenet initialization

The experiments show that there is not initial boost in training accuracy when the CNN models are initialized with pre-trained ImageNet weights. This fact is more obviously shown in table 11 with the initial accuracy values after one epoch for both initialization strategies.

Model	**Xavier init acc**	**ImageNet init acc**
Inception V3	0.6539	0.6268
VGG16	0.6519	0.6307
ResNet50	0.6916	0.6389

Table 11: Accuracy comparison after one epoch with Xavier and ImageNet initialization

The final accuracy values for both initialization strategies are shown in table 12. Only



(a) Accuracy (b) Loss

Figure 20: Accuracy and loss after 10 epochs for the ResNet50 model with Imagenet initialization

VGG16 achieved a slightly higher accuracy value at the end while using the ImageNet initialization strategy while the other two models performed worse.

Model	**Xavier init acc**	**ImageNet init acc**
Inception V3	0.9338	0.6334
VGG16	0.9723	0.9745
ResNet50	0.9548	0.9134

Table 12: Accuracy comparison after 10 epochs with Xavier and ImageNet initialization

6 Discussion

- Original learned domain of ImageNet inputs differs too much from the target domain of keyword spotting using log spectrograms
- Batch Normalization normalizes input per mini-batch and helps speeding up the training process but also makes the initialization strategy less important [16]
- The different CNN topologies indicate that pre-trained ImageNet weights are not a suitable initialization strategy when it comes to the presented task of keyword spotting
- Adam as optimization strategy may be not the best fit based on it initialization bias correction terms [17]. Stochastic gradient descent may be the better fit. Other optimization strategies are also worth investigating
-

The training accuracy did not start higher as expected but lower when using pre-trained ImageNet weights as the initialization strategy. The final training accuracy only improved slightly for one model, namely VGG16 while the other two models performed worse.
The main problem here is probably the naive approach to make all layers of the CNN models trainable. By making all layers trainable the idea behind transfer learning is almost lost because the trained weights of the other domain are overwritten by the inputs of the new domain. The initialization of weights from another domain, namely ImageNet data than the target domain therefore probably leads to a weight distribution which is not ideal as an initialization approach and shifts too far from the distribution which needs to be learned. The fact that all layers of the pre-trained CNN models have been made trainable leads to a longer and more costly error-correction process inside the networks while trying to correct the already learned basic ImageNet features to fit the new domain of spectrograms. A better approach to test if pre-trained ImageNet weights are beneficial to the task of keyword spotting would probably be to recursively "freeze" the layers of the chosen model which makes them untrainble and check which is the optimal amount of frozen layers which leads to the highest accuracy. The main approach to transfer learning is to use a certain amount of frozen or fixed layers which learned the weight distribution of another domain and then stack another shallow model on top which uses the already learned features while fine-tuning the shallow model on top. The shallow model on top which is called "standard top layer configuration" in this project is explained in section 3.3.2 but is probably superfluous in this setup because all layers of the tested models are trainable and therefore are able to learn the new target domain.

7 Conclusion

The experiments have shown that using pre-trained ImageNet weights as an initialization strategy for CNNs in order to tackle the task of keyword spotting using spectrograms is not beneficial but rather harmful. Unfortunately, the hypothesis that pre-trained ImageNet weights as an initialization strategy for the task of speech recognition would be beneficial could not be proven.

8 Future Work

- Use other image representations like spectrograms based on MFCC

- 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

	experimental	**theoretical**
aspect	**(max. points)**	**(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 21: Weighted grading

- 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

References

[1] Tensorflow speech recognition challenge. `https://www.kaggle.com/c/tensorflow-speech-recognition-challenge`. Accessed: 2019-08-05.

[2] Tensorflow speech recognition challenge. `https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/tutorials/mnist`. Accessed: 2019-08-05.

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

[4] 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.

[5] 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.

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

[7] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8609–8613. IEEE, 2013.

[8] 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.

[9] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[10] 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.

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

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

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

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

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

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

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[25] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[26] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

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

[28] 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.

[29] 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.

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