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

September 15, 2019

Contents

1 Introduction

Virtual voice assistants like Amazon Alexa or Google Echo are becoming a part of our everyday life more and more. In order to communicate with these assistants and giving the same experience to every user, the speech recognition system inside these assistants has to be robust towards different dialects and interfering background noises. The so called "keyword spotting" (KWS) task which is responsible for identifying special keywords embedded in a longer utterance is a special case of speech recognition. Keyword spotting algorithms are not solely used by virtual voice assistants with a special, isolated embodiment like Alexa or Echo, but are also part of mobile phones which are operating on Google's Android or Apple's iOS operating systems. Google offers the possibility to search by voice on Android devices by beginning the utterance with "OK, Google" [24] which is a prime example of keyword spotting. These KWS systems for mobile phones are mostly implemented in such a way that they are constantly listening to their environment and a so called "voice activity detection" (VAD) system with lower power consumption is preceding the actual keyword spotting system in order to spare the battery and only use the KWS system when a human voice signal is actually present. One could think that these KWS systems are operating on transformed audio data like mel frequency cepstral coefficients (MFCC) which is a representation of the short-term power spectrum of a sound but many of them are actually using spectrograms. Spectrograms are physical representations of the spectrum of frequencies of a (audio) signal as it varies through time as shown in figure 1 taken from [16]. Spectrograms offer the unique possibility to approach a KWS task by utilizing well performing methods out of the image classification domain.
Given the fast progress in the field of image classification and the spectrogram representation, it is a tempting idea to transform the domain of audio data into the domain of image data to utilize all available image classification approaches. One prominent type of network in the domain of image classification are convolutional neural networks (CNNs) which produced state-of-the-art results in the past ImageNet competitions [11] [20]. ImageNet is a large database designed for the use in visual object recognition software research and contains more than 14 milltion images. All images have been manually annotated with corresponding labels.
The vast majority of different CNN architectures in combination with the ImageNet dataset resulted in the idea that mid-level image representations could be transferred from the domain of ImageNet data to another, similar domain. This idea is known as transfer learning and has been successfully used in the past between different domains which incorporate image data [21]. To the best of my knowledge, not much research has been done on transfer learning where pre-trained ImageNet weights have been used on spectrograms for keyword spotting tasks.

(a) (b)

Figure 1: A comparison of the spectrogram (a) and oscillogram (b) for the same audio recording of a person saying the word "bed".

1.1 Previous work

The task of keyword spotting (KWS) has been approached differently in the past. A common technique is a Keyword/Filler Hidden Markov Model (HMM) which was later superseded by Deep Neural Networks (DNNs) [8]. The deep KWS system proposed in [8] consists of three components as shown in figure 2, namely

i A feature extraction module

ii A deep neural network

iii A posterior handling module

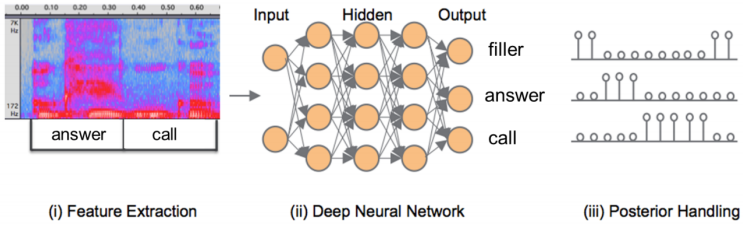


Figure 2: Framework of Deep KWS system, components from left to right: (i) Feature Extraction (ii) Deep Neural Network (iii) Posterior Handling

The feature extraction module produces 40-dimensional log-filterbank energies which are computed every 10 ms over a window of 25 ms. The deep neural network is a standard feed-forward fully connected neural network with k hidden layers and n hidden nodes per layer, each computing a non-linear function of the weighted sum of the output of the previous layer. The last layer uses a Softmax activation function which outputs an estimate of the posterior of each output label while hidden layers are using the rectified linear unit (ReLU) activation function. The training is done using cross-entropy as loss function and asynchronous stochastic gradient descent as the optimization strategy. The posterior handling method is applying posterior smoothing and a confidence score which serves as a final decision method. The results show that this framework outperforms the HMM system on both clean and noisy data. The authors state that transfer learning is used in order to initialize the hidden layers of the network and that all layers are updated in training. This results in more robust feature representations and avoids bad local optima. Unforunately, the authors do not state which dataset is used for transfer

learning.

Later work picks up the three module framework shown in figure 2 and replaces the DNN with a CNN [23]. The CNN architecture is evaluated by limiting the amount of multiplications and parameters in order to fit it to the constraints of small-footprint keyword spotting tasks. The authors state that CNNs perform better with regards to the keyword spotting task because they capture translational invariance due to different speaking styles with less parameters than DNNs. The CNN outperforms the DNN and offers a 27-44% relative improvement in false reject rates. The proposed pooling in time strategy also produces a 41% relative improvement over a DNN architecture for clean and noisy data. The feature extraction step remains the same as in [8]. Other architectures which are inspired by CNNs like Convolutional Recurrent Neural Networks (CRNNs) also showed promising results in the area of keyword spotting [7].

The success of CNNs in the domain of speech recognition on log-mel filterbank features inspired others to transform audio data to image representations where CNN architectures are known to perform well. So the KWS task which incoporates audio data was transformed to the domain of image classification [16]. The authors use the Speech Commands Dataset [29] which contains spoken words of the length of one second in order to train and evalutate their model. According to [29], the Speech Commands Dataset V1 [15] comprises one-second audio clips which were sampled at 16kHz. A vector representation of these one-second audio clips would therefore be of the form $\mathbb{R}^{16000}$. The authors evaluated three different models, namely a low latency CNN, a CNN inspired by TensorFlow's MNIST tutorial and a custom adversarially trained CNN which is inspired by MCDNN [9] and AlexNet [20]. The adversarial approach is used because Dropout was counter-productive [14]. The authors use grayscale spectrograms of size $28x28$ as model inputs. Based on the fact that the authors use stochastic gradient descent at the optimization strategy, the parameter initialization is important and Xavier initialization is used [13]. The authors show that their regularization method of "Virtual Adversarial Training" achieved a maximum of 92% validation accuracy and that it is possible to achieve good results by converting an audio recognition problem into the better studied domain of image classification.

Previous work also shows that it is possible to transfer the hidden layer activations for extracting deep features and applying them to the task of key word spotting [22]. This results into more robust feature learning but is only used for domains which are still related to speech. The task of transferring learned features from one related domain to the other is also shown when it comes to visual recognition tasks [21]. The authors show that transferred features lead to significantly improved results for object detection when using the ImageNet data to compute mid-level image representations.

Nevertheless, not much research has been done on transfer learning when it comes to transferring learned mid-level representations in the image domain to the audio domain.

1.2 Research Question

Therefore, this project focuses on the idea if already learned features from the image domain, e.g., ImageNet dataset could be used as an initilization strategy to boost the performance of CNN classifiers which operate on audio data which has been transformed to spectrograms. The problem is narrowed by concentrating on the KWS task and the usage of pre-trained CNN models which have been trained on ImageNet data. The Ten-

sorFlow Speech Recognition Challange [3] is kindly providing the audio dataset, namely the Speech commands dataset [15].

The main research question is therefore whether pre-trained ImageNet weights are boosting the initial accuracy of CNN models by serving as an initialization strategy for the task of speech recognition on log spectrograms in contrast to common initialization strategies like Xavier initialization proposed in [16]. The hypothesis is therefore that the pre-trained ImageNet weights should improve the initial accuracy of the CNN models.

2 Method

The method is inspired by [16] where three different models have been evaluated on their capability to handle audio data transformed to images. All layers of all models are trainable. One of the baseline models is the MNIST model which is also used in the Tensorflow Speech Recognition tutorial [5]. This model together with another one ("Lightweight CNN") taken from the Kaggle forum and explained later on is used as the initial baseline when it comes to accuracy. These two models solely use Xavier initialization as used in [16] and introduced in [13]. This initialization strategy is explained as follows:

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

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

The main part of the project is evaluating the performance of three different CNN architectures, namely VGG16, ResNet50 and Inception V3 with regards to their initial accuracy values when they are initialized with the Xavier strategy and then with pre-trained ImageNet model weights. These models are used because they each have unique topologies and won the ImageNet challange in their respective years. The different topologies should therefore guarantee that an improvement in initial accuracy values is actually due to the initialization strategy and not the topology itself. Log spectrograms are used as the default input as proposed in [16]. The models are trained over a period of 10 epochs and the early stopping criterion is discarded in order to have comparable training results and inspecting convergence rates.

3 Set-up

3.1 Dataset

The TensorFlow Speech Recognition Challenge hosted by Kaggle [3] 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 [15] and is explained in more detail by Warden [29]. 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 [3] 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 certain words, the class distribution changed from a balanced distribution to a rather unbalanced distribution towards the "unknown" label as depicted in table 1

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 1: 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.

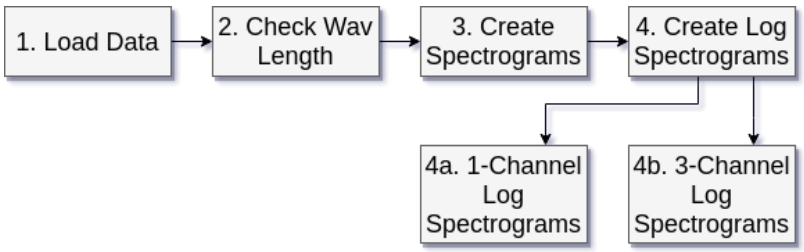


Figure 3: Preprocessing

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 2.

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

The different paths were iterated and the wav files were loaded 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.

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 [16] 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.

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 6 which should be beneficial for the model training. One potential problem however is shown in figure 6 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:
```

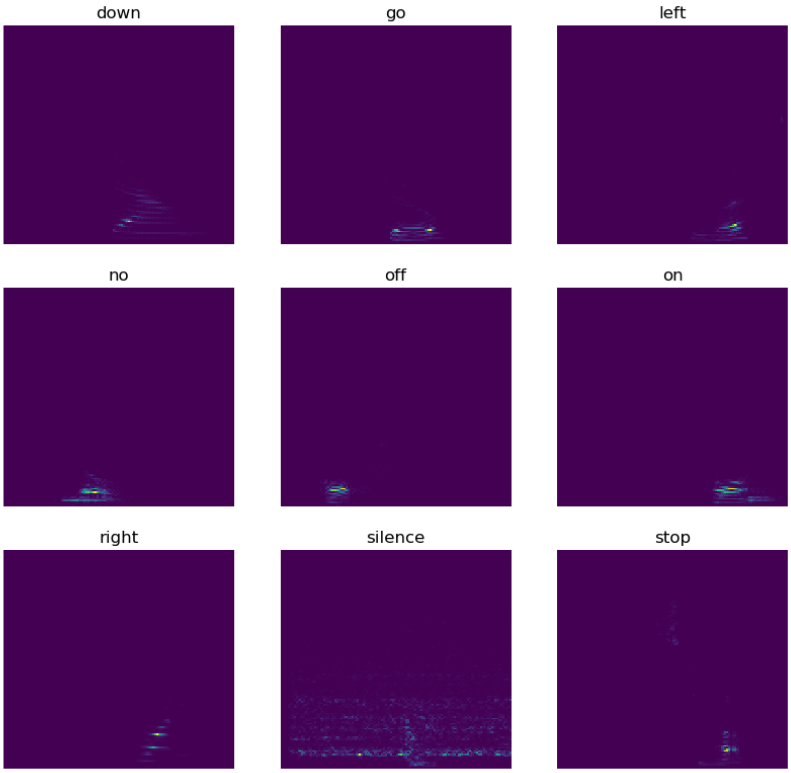


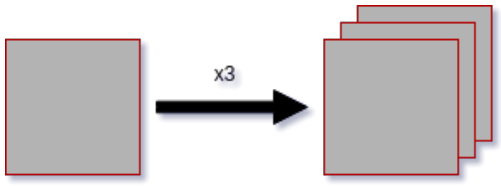
Figure 4: Spectrogram samples of nine different classes

```
        sig = sig[0:sample_rate]

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

    # 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

(a) 1-channel log spectrogram (b) 3-channel RGB image



(c) Duplication of 1-channel log spectrogram

Figure 5: Conversion between 1-channel and 3-channel log spectrograms

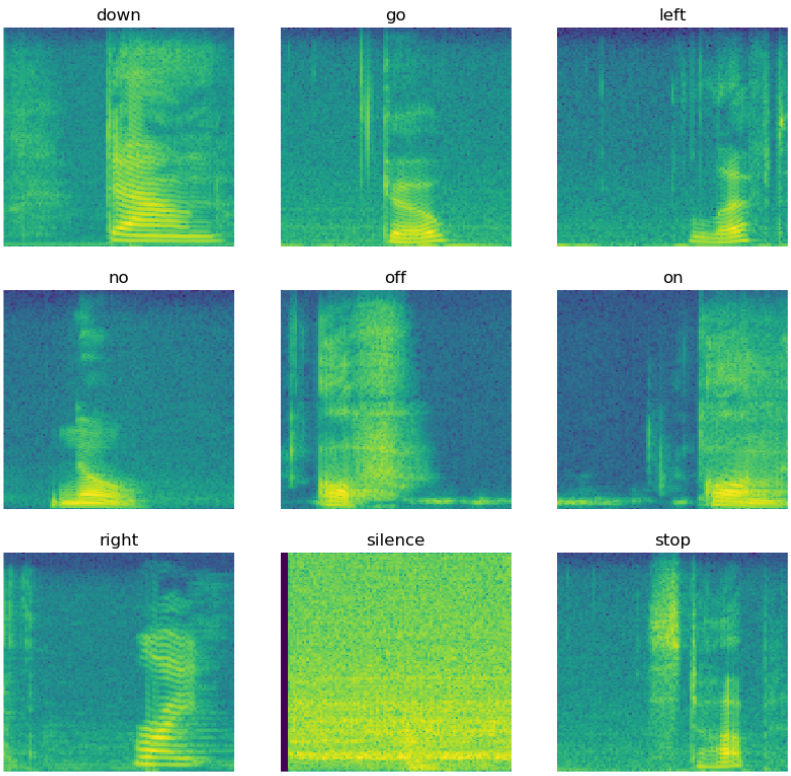


Figure 6: Log spectrogram samples of nine different classes

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 6 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 5.

3.3 Parameters

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 [10]. However, Softmax is applied as the final activation function in all output layers in order get proper probabiliy scores.

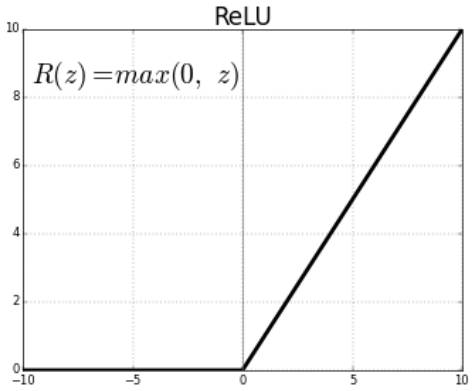


Figure 7: Rectified Linear Unit as activation function

Throughout all architectures Dropout has been used [26] with a value of 0.2 as a regularization technique as depicted in figure 9 and as proposed in [10].

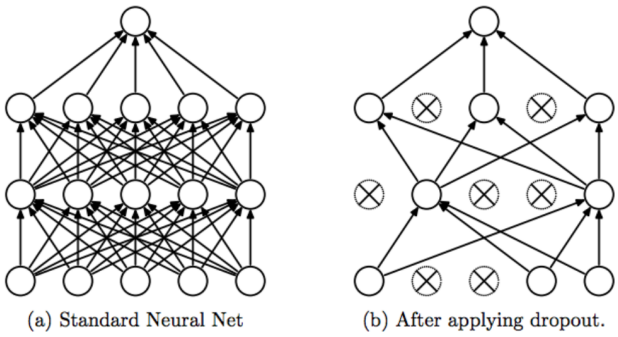


Figure 8: Dropout as generalization technique

Batch normalization as depicted in figure 9 and proposed in [18] 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.

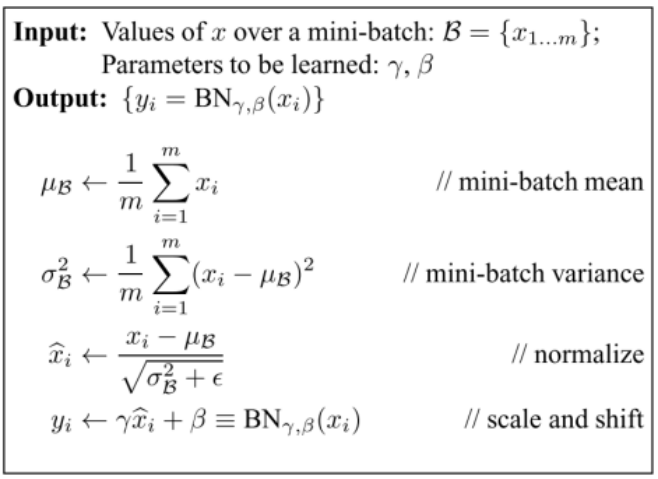


Figure 9: Batch normalization to speed up training, reducing internal covariate shift and as general regularization technique

Adam [19] has been used as the main optimization technique for all models with a learning rate of 0.001 while the training set has been split up into a 80% training and 20% validation set. An overview of the parameters is shown in table 3

Parameter	**Value**
Dropout	0.2
Optimization	Adam
Loss	Categorical Crossentropy
Training/validation ratio	80:20
Epochs	10
Batch size	32
Activation	ReLU

Table 3: Overall parameters

3.4 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 ResNet50 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.

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

3.4.1 Baseline models

MNIST

The chosen architecture of the so called "MNIST" model is burrowed from the TensorFlow repository [4] and showed an accuracy of 98% on the task for grayscale digit recognition [12]. 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 model which is known to perform well 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`

3.4.2 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 on 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

CNN models which originally do not use batch normalization therefore get this feature on top in order to make the models more comparable to each other.

VGG16

The VGG architecture has been proposed in 2014 and incorporates increasing depth using an architecture with very small (3×3) convolutional filters and a depth of 16 to 19 weight layers [25]. The main VGG architecture [2] is depicted in figure 10 and also won the ImageNet Challenge in 2014.

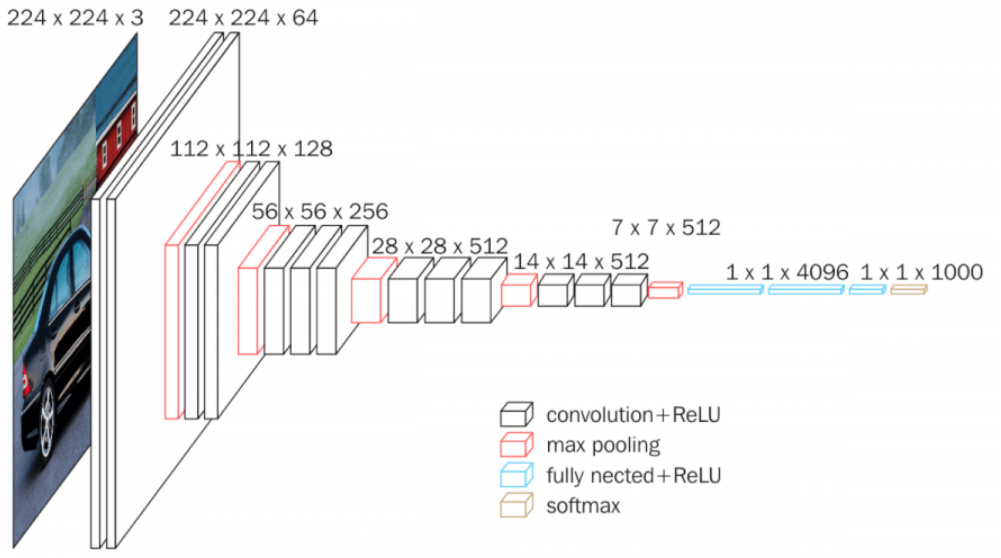


Figure 10: VGG16 architecture

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

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

pre-trained ImageNet weights for this model [6].

Inception V3

The Inception V3 architecture is the winner of the ImageNet Challenge of 2015 and outperformed the VGG architecture [27]. Its main architecture is depicted in figure 11. The 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 greedy max pooling operation in the convolutional layers like VGG16.

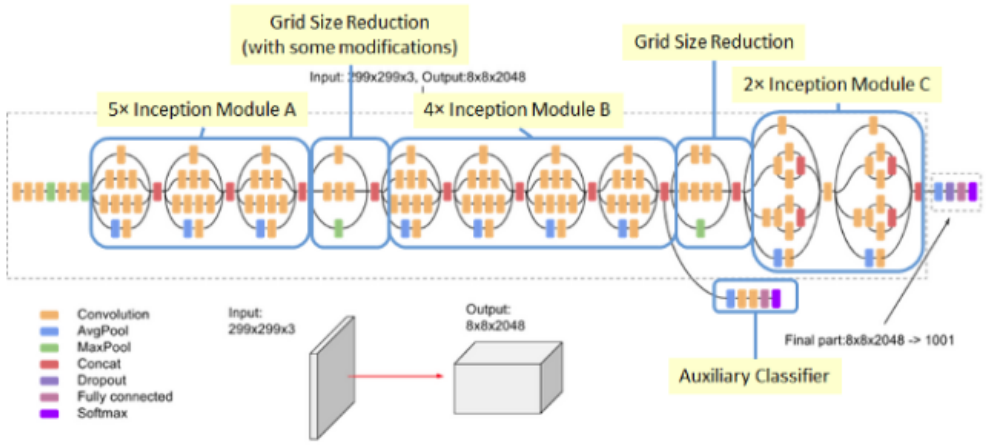


Figure 11: Inception-v3 Architecture (Batch Norm and ReLU are used after Conv)

In order to draw good conlusions with regards to the benefits of the ImageNet initialization strategy while being 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 [1].

[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 [17]. 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.

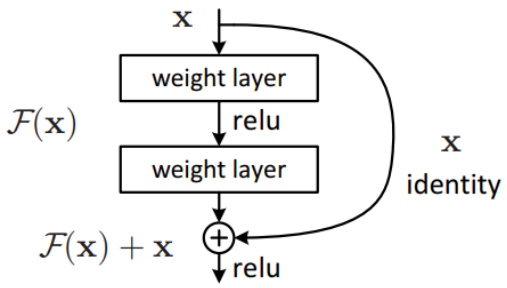


Figure 12: Residual learning: a building block.

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 [2] 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 training 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 on 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 on 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 answers 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 3.4 remain the same.

5 Analysis and Results

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

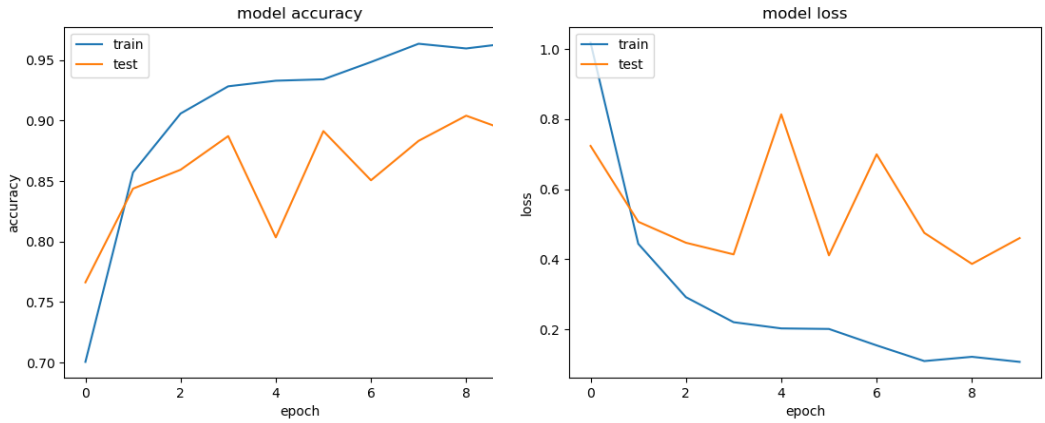
5.1 Baseline

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.



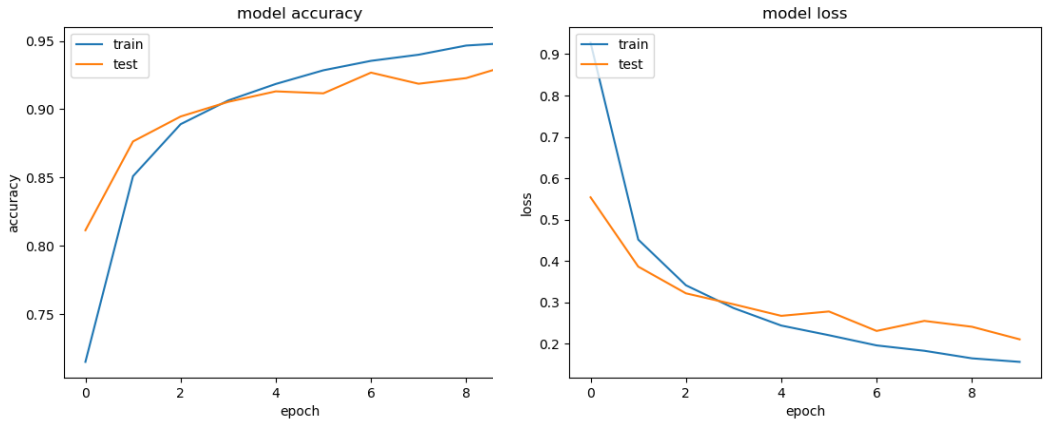
(a) Accuracy (b) Loss

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
Lightweight 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 is still below the initial values of the baseline models. Resnet performed the best in the initial run of all three models with about a 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

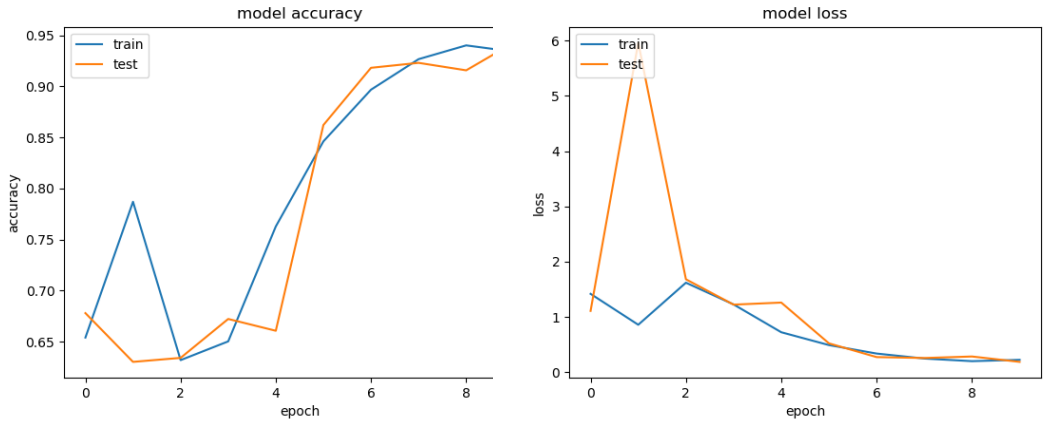
Table 7: CNN results after one epoch with Xavier Glorot initialization

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.

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.

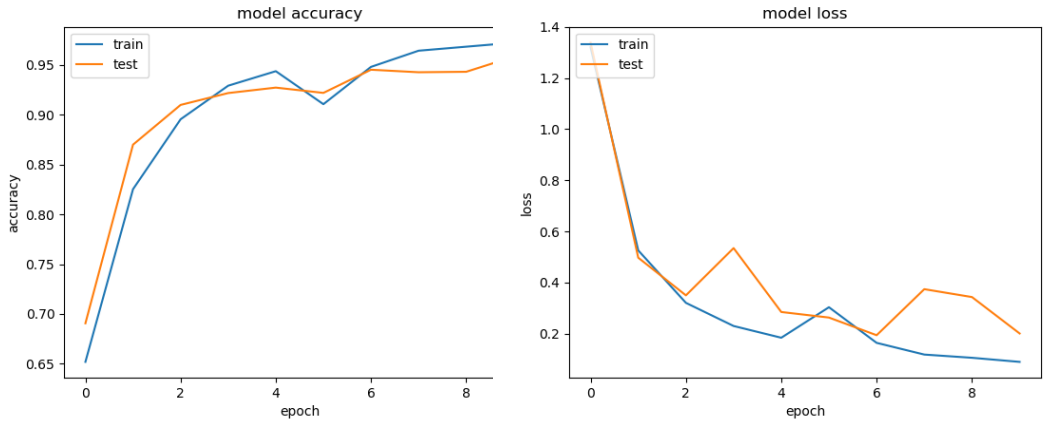
Model	Train Acc	Train Loss	Val Acc	Val Loss	Time (sec)
Inception V3	0.9338	0.2238	0.9427	0.1868	2332.23
VGG16	0.9723	0.0900	0.9583	0.2011	2530.47
ResNet50	0.9548	0.1421	0.9445	0.1873	3807.93

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

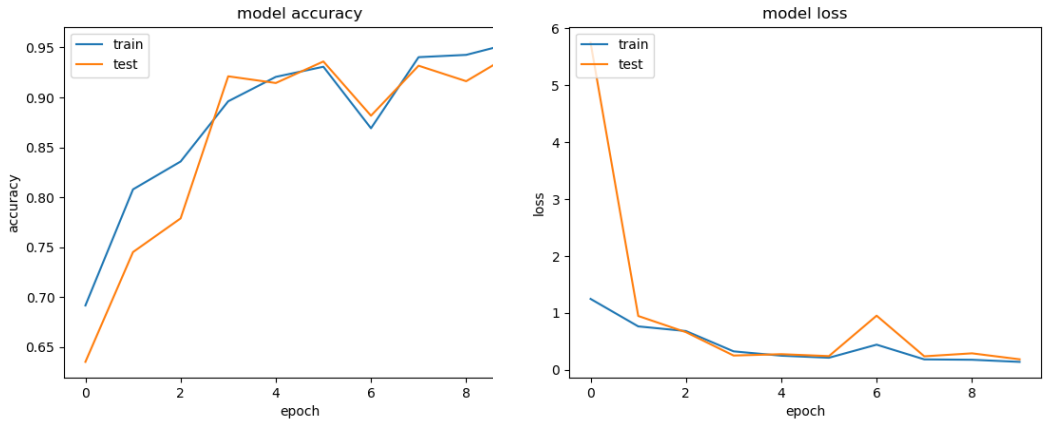


(a) Accuracy (b) Loss

Figure 16: Accuracy and loss after 10 epochs for the VGG16 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



(a) Accuracy (b) Loss

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

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

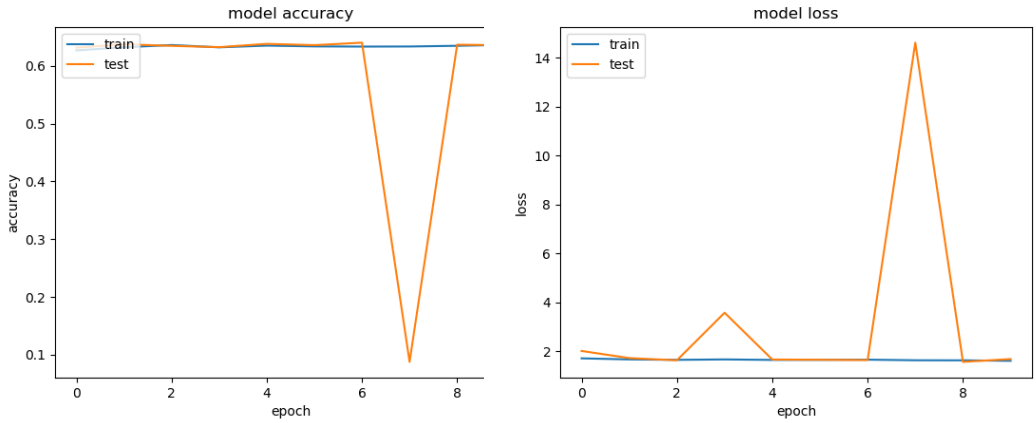
The final accuracy values as shown in table 10 show that VGG16 performed the best again while having a slightly higher accuracy value than the Xavier Glorot initialization strategy after 10 epochs. Inception V3 performed almost 30% worse according to the final accuracy value compared to the Xavier Glorot initialization. It is not clear what the cause of that is but future work should probably incorporate a k-fold cross-validation setup to validate if this performance is just an exception or constant across multiple model runs. 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

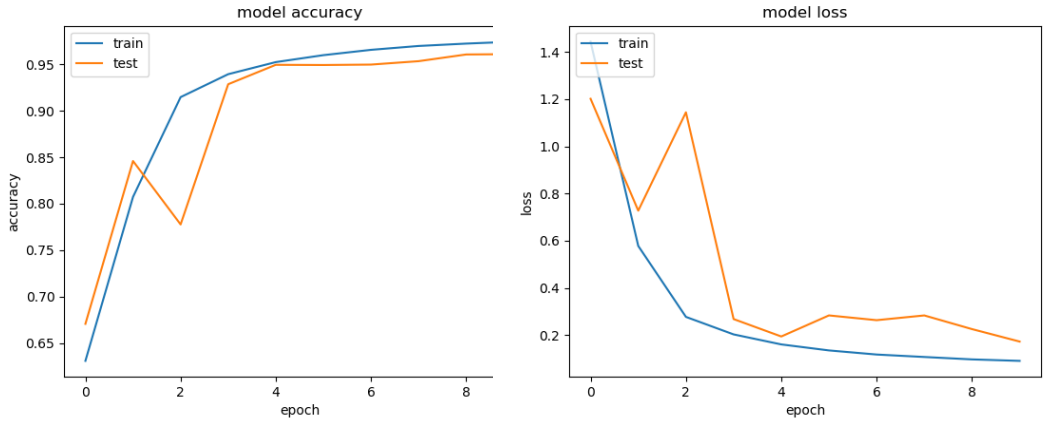
ResNet scored 4% lower in terms of accuracy after 10 epochs when initialized with pre-trained ImageNet weights, but both VGG16 and ResNet showed a convergence towards a maximum in terms of accuracy depicted in figure 19 and 20 respectively in contrast to Inception V3.

The experiments show that there is no initial boost in training accuracy when the CNN models are initialized with pre-trained ImageNet weights. This fact is more ob-



(a) Accuracy (b) Loss

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



(a) Accuracy (b) Loss

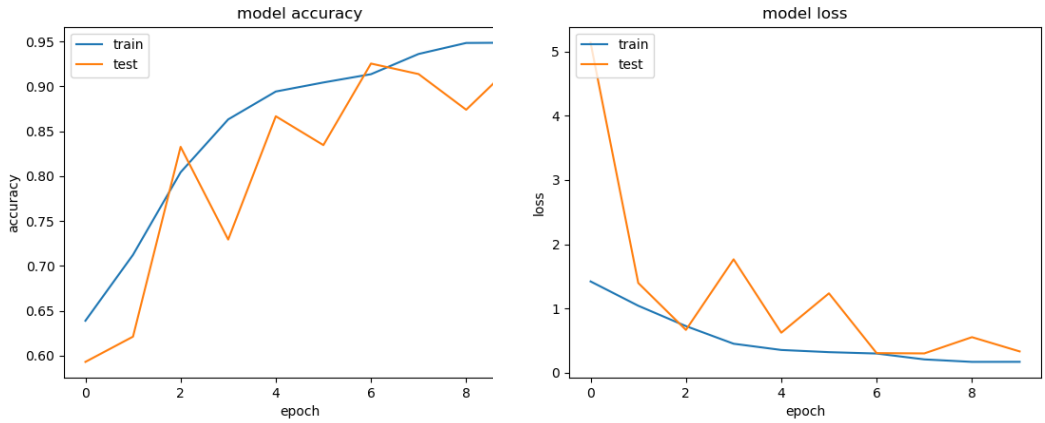
Figure 19: Accuracy and loss after 10 epochs for the VGG16 model with Imagenet initialization

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



(a) Accuracy (b) Loss

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

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

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 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. This is probably the reason why the ImageNet initialization strategy leads to lower starting accuracies. Given that the ImageNet dataset is including images based on 3-channels, namely RGB and the produced spectrograms are only based on grayscale and therefore 1-channel, the pre-trained weights could have been of more use if the original ImageNet dataset would have been converted to grayscale before transferring them to fit the target domain. 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 this top model. The shallow model on top which is called "standard top layer configuration" in this project is explained in section 3.4.2 but is probably superfluous in this setup because all layers of the tested models are trainable and are therefore able to learn the new target domain based on their complexity. Another explanation could be that the setup is right but the originally learned domain of ImageNet inputs differs too much from the target domain of keyword spotting using log spectrograms.
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 because they all experienced the same effect of lower starting accuracies but on the other hand all layers of all CNN models have been made trainable which might negates the statement that pre-trained ImageNet weights are not suitable for this case.
Batch Normalization normalizes input per mini-batch and helps speeding up the training process but also makes the initialization strategy less important [18] which is a crucial point which has not been considered carefully enough before setting up the experiment. During the experiment a certain progress in convergence rate towards the accuracy maximum has been shown by using batch normalization but it might have been a better option to drop the batch normalization layers in favor of having clearer results about the initialization strategies and only use dropout as a regularization technique.
Adam as an optimization strategy may not be the best fit based on its initialization bias correction terms [19]. Stochastic gradient descent may be the better fit because it has also been used in previos works in similar domains [16] while other optimization strategies might also be worth investigating as long as they do not correct the initialization bias.

7 Conclusion

The experiments have shown that the usage of 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.

On the other hand, there is still room for improvement in terms of setup variables to disprove this statement by adjusting the transfer learning strategy by freezing different amounts of layers, using a different optimization strategy or using another generalization technique than batch normalization which is not influencing the importance of the initialization strategy as discussed in section 6.

8 Future Work

Future work could concentrate on the effect of batch normalization with regards to transfer learning and in how far it influences the importance of the right initialization strategy.
Another point which is worth investigating is the right amount of fixed or frozen layers while applying transfer learning between domains which do not directly relate to each other like speech recognition and image classification as presented in this project. The right amount of fixed layers and how far fine-tuning has to be applied in order to make use of already learned features from another domain seems to be of great importance.

The idea of recursive feature elimination as used in text analysis to determine the most contributing features could also be applied to the right amount of fixed or frozen layers. Additional preprocessing and feature engineering steps also seem interesting to investigate when transfer learning is applied, e.g., the correlation between accuracy and the dimensions of spectrograms while using already learned features from another domain. The detection of outliers and removing bad audio samples could also be beneficial in order to boost the overall accuracy.
Using spectrograms based on other features like mel frequency cepstral coefficients (MFCC) and see if transfer learning is beneficial could also be an interesting area to investigate in the future.

References

[1] Inception v3 implementation. `https://github.com/keras-team/keras-applications/blob/master/keras_applications/inception_v3.py`. Accessed: 2019-08-05.

[2] Resnet 50 implementation. `https://github.com/keras-team/keras-applications/blob/master/keras_applications/resnet50.py`. Accessed: 2019-08-05.

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

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

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

[6] Vgg16 implementation. `https://github.com/keras-team/keras-applications/blob/master/keras_applications/vgg16.py`. Accessed: 2019-08-05.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

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

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

[17] 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.

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

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

[20] 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.

[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. In *Interspeech*, 2015.

[24] Johan Schalkwyk, Doug Beeferman, Françoise Beaufays, Bill Byrne, Ciprian Chelba, Mike Cohen, Maryam Kamvar, and Brian Strope. "your word is my command": Google search by voice: A case study. In *Advances in speech recognition*, pages 61–90. Springer, 2010.

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

[26] 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.

[27] 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.

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