Track D: Audio

Audio classification on google speech commands dataset using CNN, LSTM and GRU models

ERLEND TREGDE, SANDER WESSTØL, LINOR UJKANI

SUPERVISOR
Sander Riisøen Jyhne

University of Agder, 2025
Faculty of Engineering and Science
Department of Engineering and Sciences

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Ja

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Acknowledgements

We would like to thank Sander Jyhne for valuable guidance and support throughout this project.

Further thanks go to Håvard Karlsen Solheim for providing audio clips used in the inference experiments.

Abstract

The project investigates deep learning models for classifying short spoken commands from the Google Speech Commands dataset. The work follows the SEMMA method, where the data are sampled and explored, and the audio is preprocessed into spectrograms and MFCC features before modelling. The models are then evaluated in the final stage, Assess. The project sets out to answer the following research question:

For the Google Speech Commands dataset, which deep learning architecture achieves the best classification performance, and how do key hyperparameters influence performance?

Several CNN, LSTM, and GRU architectures are implemented, starting with simple baselines and then increasing depth and complexity. The best result is achieved by a custom CNN with five convolutional layers and one fully connected layer, achieving a test accuracy of 95.80%. In contrast, the best LSTM and GRU models achieve accuracies of 94–95%. These results show that careful preprocessing and relatively simple architectures can give high performance when classifying audio clips on the Google Speech Command dataset.

Contents

Acknowledgements **ii**

Abstract **iii**

List of Figures **vi**

List of Tables **vii**

1 Introduction **1**
1.1 Motivation, Research Question and Objective 1
1.2 Use of AI tools 1

2 Background **2**
2.1 SEMMA 2
2.2 Preprocessing 2
2.3 Spectrograms 2
2.4 MFCC 2
2.5 Waveforms 3
2.6 CNN 3
2.7 RNNs 4
2.8 LSTM 5
2.9 GRU 6

3 Method **7**
3.1 SEMMA applied 7
3.1.1 Sample 7
3.1.2 Explore 8
3.1.3 Modify 8
3.1.4 Model 8
3.1.4.1 Deep learning 8
3.1.4.2 Baseline 8
3.1.5 Assess 8
3.1.5.1 Inference 9

4 Results **10**
4.1 Dataset 10
4.2 Sample 10
4.3 Explore 11
4.4 Modify 13
4.5 Model 14
4.5.1 CNN 14
4.5.1.1 Existing CNN architectures 16
4.5.1.2 Final takes 17

4.5.2 LSTM . . . 17
4.5.2.1 Existing LSTM architectures . . . 19
4.5.2.2 Bidirectional and Deep Bidirectional LSTM: . . . 19
4.5.2.3 ConvLSTM: . . . 19
4.5.2.4 LSTM with Attention: . . . 19
4.5.2.5 Peephole LSTM: . . . 20
4.5.2.6 Final takes . . . 21
4.5.3 GRU . . . 21
4.5.3.1 Existing GRU architectures . . . 23
4.5.3.2 Bidirectional GRU . . . 23
4.5.3.3 Deep Bidirectional GRU . . . 23
4.5.3.4 GRU with Attention . . . 23
4.5.3.5 Seq2Seq GRU . . . 23
4.5.3.6 Light GRU . . . 23
4.5.3.7 Final takes . . . 24
4.6 Assess . . . 25
4.6.1 Inference . . . 26

5 Discussions **27**
5.1 Sample . . . 27
5.2 Explore . . . 27
5.3 Modify . . . 27
5.4 Models . . . 28
5.4.1 CNN . . . 28
5.4.2 LSTM . . . 28
5.4.3 GRU . . . 28
5.5 Assess . . . 29
5.5.1 Inference . . . 29

6 Conclusions **30**

Bibliography **31**

List of Figures

2.1 Example of a very minimal neural network, taken from [28] 3
2.2 Recurrent Neural Network architecture from [9] 4
2.3 Figure of lstm arhitecture from [16] 5
2.4 Figure from GeeksForGeeks [8] 6

3.1 SEMMA figure made in Canva 7

4.1 Plot of the different frequencies observed in the dataset. 12
4.2 Plot of the different amplitudes observed in the dataset. 12
4.3 MFCC and Mel spectrogram of house 13
4.4 Model performance on the test dataset, when changing the number of layers. Bottom blue is the baseline. 15
4.5 Accuracy for test dataset for the architecture comparisons 17
4.6 Accuracy for test dataset for the architecture comparisons 19
4.7 Accuracy for test dataset for the existing LSTM architecture comparisons . . 20
4.8 Graphed results of the stacked GRU results 22
4.9 Existing GRU architectures compared to baseline 24
4.10 Confusion matrix of best model 25

List of Tables

4.1 Classification results of multiple attempts: c = convolutional, f = linear . . . 15
4.2 Classification results comparing various CNN architectures 16
4.3 Classification results for different layer and fully connected configurations . . 18
4.4 Classification results comparing various LSTM-based architectures 20
4.5 Classification results using stacked GRU networks with varying GRU (l) and fully connected (f) layers. 22
4.6 Classification results comparing various GRU-based architectures 24
4.7 Classification results comparing various LSTM-based architectures 26

Chapter 1

Introduction

In this project, the goal is to build deep learning models that can classify short spoken-word commands from the Google Speech Commands dataset using the method SEMMA. From a deep learning perspective, this task is interesting because raw audio must be transformed into a representation that neural networks can learn from, such as spectrograms or MFCCs. Small design changes, including the number of layers, architectural designs, loss functions, and related techniques, can significantly affect performance. The goal is not only to train the model, but also to explore how architecture choices, preprocessing, and training strategies influence classification accuracy and generalization.

1.1 Motivation, Research Question and Objective

The motivation for this project is to get experience with deep neural networks for audio by implementing and training at least one deep model (CNN, LSTM/GRU) and systematically experimenting with architectural variations, preprocessing, and training strategies using the method SEMMA.

Building on this motivation, the objective of this assignment is to classify audio clips using different models, such as CNNs, LSTMs, and GRUs, and observe how modifications to the model's layers and attributes affect classification accuracy, resulting in the following research question:

- **For the Google Speech Commands dataset, which deep learning architecture achieves the best classification performance, and how do key hyperparameters influence performance?**

1.2 Use of AI tools

For this project, several AI tools have been utilized when working on the tasks. ChatGPT 5.1 and Grammarly have been used to fix grammatical errors. ChatGPT 5.1 has not been used to write paragraphs for the task. However, it has been used as a tool to get suggestions on report structure and generate ideas. For the coding and implementation part of the project, GitHub Copilot and Claude Sonnet 4.5 have been used to debug the code.

Chapter 2

Background

2.1 SEMMA

SEMMA (Sample, Explore, Modify, Model, Assess) is a process model developed by SAS for structuring data mining projects. It is organised as a five-stage cycle. In the Sample stage, a subset of the original dataset is selected that is large enough to capture the important information, but still small enough to be handled efficiently. In the Explore stage, this sample is analysed to identify unexpected patterns, trends, and anomalies that can generate insight and ideas. The Modify stage focuses on preparing the data by creating, selecting, and transforming variables to support an effective model selection process. In the Model stage, different modelling techniques are applied, typically by letting the software automatically search for combinations of variables that can reliably predict the desired outcome. Finally, in the Assess stage, the resulting models are evaluated with respect to their usefulness, reliability, and overall performance in the data mining task. [1]

2.2 Preprocessing

Preprocessing is the act of manipulating the raw data into another format that the model can encode and read in contrary to its original form. This can be preprocessed, such as conversion to MFCC or conversion to spectrograms.

2.3 Spectrograms

A spectrogram is a visual representation of sound changes over time, using an axis system. This includes having the sound frequencies on the vertical axis, the time on the horizontal axis, and the amplitude of the sound demonstrated by the color intensity in the given coordinate. Spectrograms are generated by using a transformation called short-time fourier transform, which divides the sound into many small overlapping windows, and calculated the frequency spectrum for each window. Often use cases for spectrograms are for analysing audio and musical notes, revealing patterns that arent originally noticeable in the raw waveform. A version of a spectrogram is a mel spectrogram, which applies a mel-scale transformation that adjusts the original frequency spacing to match human hearing. This allows for image classification models like CNN to classify audio. [23, 27]

2.4 MFCC

Mel Frequency Cepstral Coefficient (or MFCC) is a way to represent the power spectrum of sound. MFCC is a mathematical representation of the sound produced by humans when they speak. The MFCC breaks down human speech into simpler parts, which helps capture characteristics in speech patterns, making it easier to distinguish between different speech patterns. Similar to a mel spectrogram, the MFCC uses a mel-scale to approximate how the human ear processes audio, which is very sensitive to changes in the lower frequencies. [18, 4]

2.5 Waveforms

Generally a waveform is the shape of the input graphed in a function of time. A waveform in audio is just a visual representation of the raw data in a function over time. This would therefore be the amplitude of the audio in the vertical direction and the time of the audio in the horizontal direction [25, 26].

2.6 CNN

Convolutional Neural Network (Called CNNs) are a type of deep learning architectures that is specifically designed for processing images, or other grid-like structured data. This model architecture does mainly do three different things before classification, which again can be divided into a "feature extraction" part and a "classification" part. A simple CNN architecture model can be seen in figure 2.1. [28]

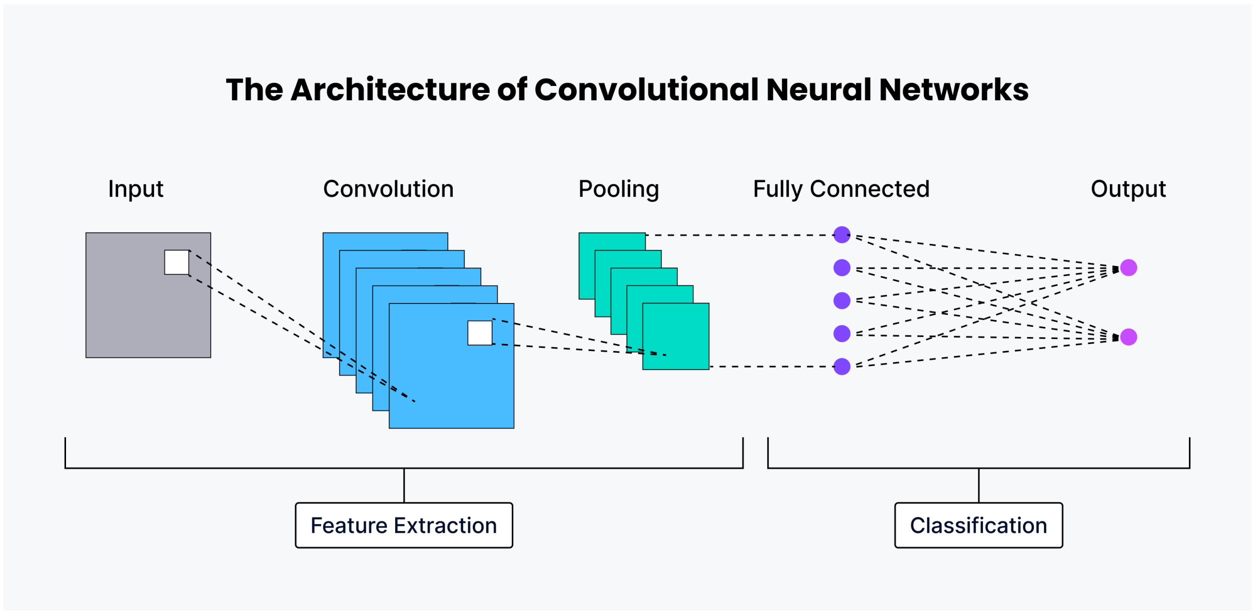


Figure 2.1: Example of a very minimal neural network, taken from [28]

The model looks at a local field of the image using convolutional layers, extracting key features. It shares its parameters (kernel) across the entire image. Eventually, it uses down-sampling of the image with pooling operations. This allows the convolutional layers to detect smaller local patterns, such as textures or edges, across the whole image. Thereafter, pooling is often followed to reduce the dimensions of the image while keeping important information, as this decreases both computational time and complexity. Finally the output from these convolutional feature extractions is flattened and passed to fully connected layer(s) for a final classification. [28]

2.7 RNNs

Recurrent Neural Networks (Called RNNs) are a type of deep learning architectures that is specifically designed to process sequential data. It does this by maintaining an internal memory (called hidden state) of the previous inputs which again lets the RNN use feedback loops with information from previous timesteps to impact current computations. This allows RNNs to capture "dependencies" across sequences. The architecture of the RNN can be seen in figure 2.2

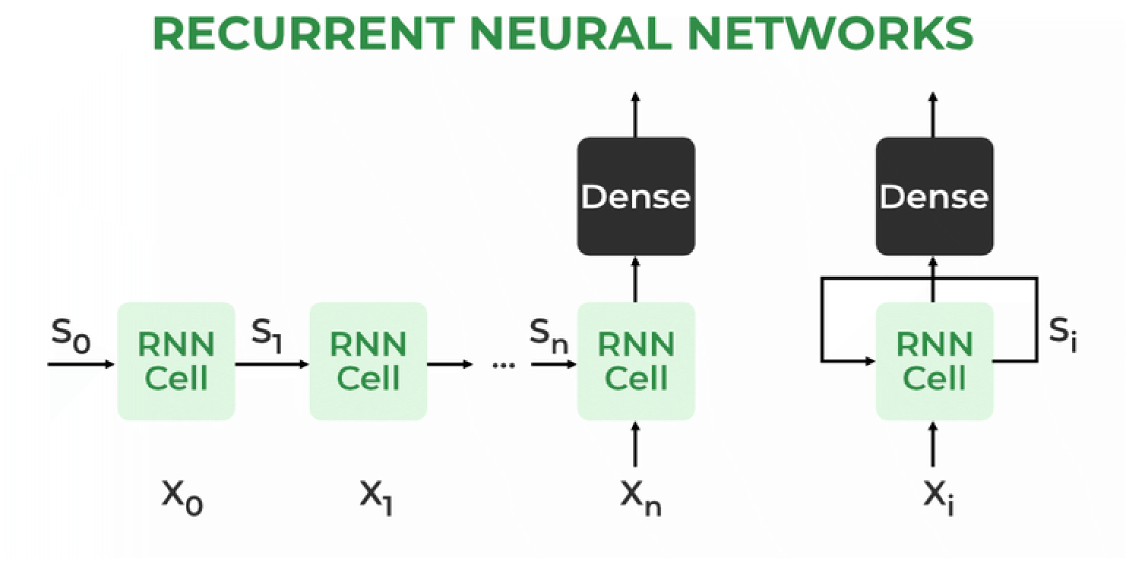


Figure 2.2: Recurrent Neural Network architecture from [9]

At each time step in the RNN, the hidden state is updated using the current inputs and the previous hidden state. The way it calculates the new hidden state is by using the following relation:

$$h_t = \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t)$$

Where W_{hh} is the recurrent weight matrix, and W_{xh} is the input weight matrix. These weights are updated using backpropagation through time (or BPTT), which works like normal backpropagation but with time steps to compute gradients. Models of this kind tend to struggle with vanishing or exploding gradients. However, more advanced variants such as the LSTM and GRU provide mechanisms to address this challenge. [21, 9]

2.8 LSTM

Long Short-Term Memory (LSTM) networks are a subtype of Recurrent Neural Networks that are used to recognize patterns in data sequences, such as sensor data, stock prices, or natural language [16]. They extend standard RNNs by adding a long-term memory called the cell state in addition to the hidden state, which serves as short-term memory. This architecture allows LSTMs to retain selected information over many time steps and handle longer dependencies than regular RNNs [16]. An overview of the LSTM architecture is shown in Figure 2.3.

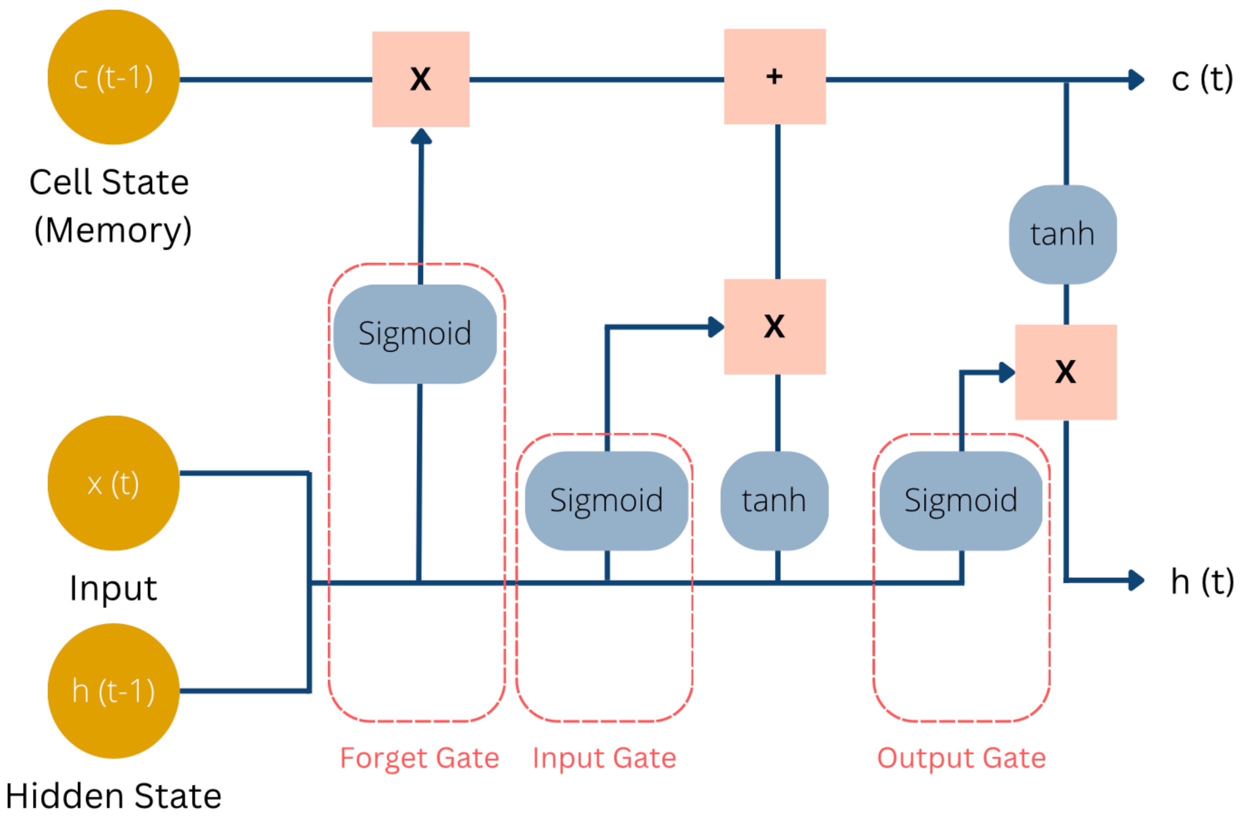


Figure 2.3: Figure of lstm arhitecture from [16]

In each computational step, the current input $x(t)$, the previous cell state $c(t-1)$, and the previous hidden state $h(t-1)$ are processed by three gates: the forget gate, the input gate, and the output gate [16]. In the forget gate, a sigmoid function decides which parts of the previous information should be kept (values close to 1) or discarded (values close to 0), and the result is multiplied by the old cell state so that unneeded information is removed. In the input gate, the model determines the value of the current input and adds the relevant parts to the cell state, thereby forming the new long-term memory $c(t)$. In the output gate, another sigmoid function together with a tanh-activated version of the cell state determines the new hidden state $h(t)$, which is both the short-term memory and the output of the LSTM at time t [16]. By selectively forgetting, updating, and exposing information in this way, LSTMs can keep relevant context over long sequences and have therefore been used in applications such as keyboard completion, voice assistants, game-playing agents, and machine translation [16].

2.9 GRU

Gated Recurrent Unit is a type of RNN that uses two distinct mechanisms, a reset gate and an update gate, to filter information in an iteration, selectively [6]. GRU was introduced as a simpler alternative to LSTM while reducing the architectural complexity and retaining a gating mechanism [6]. But unlike LSTM which uses a hidden state and a separate cell state, GRU combines both into a single hidden state and reduces the amount of parameters therefore making this model more computationally efficient.

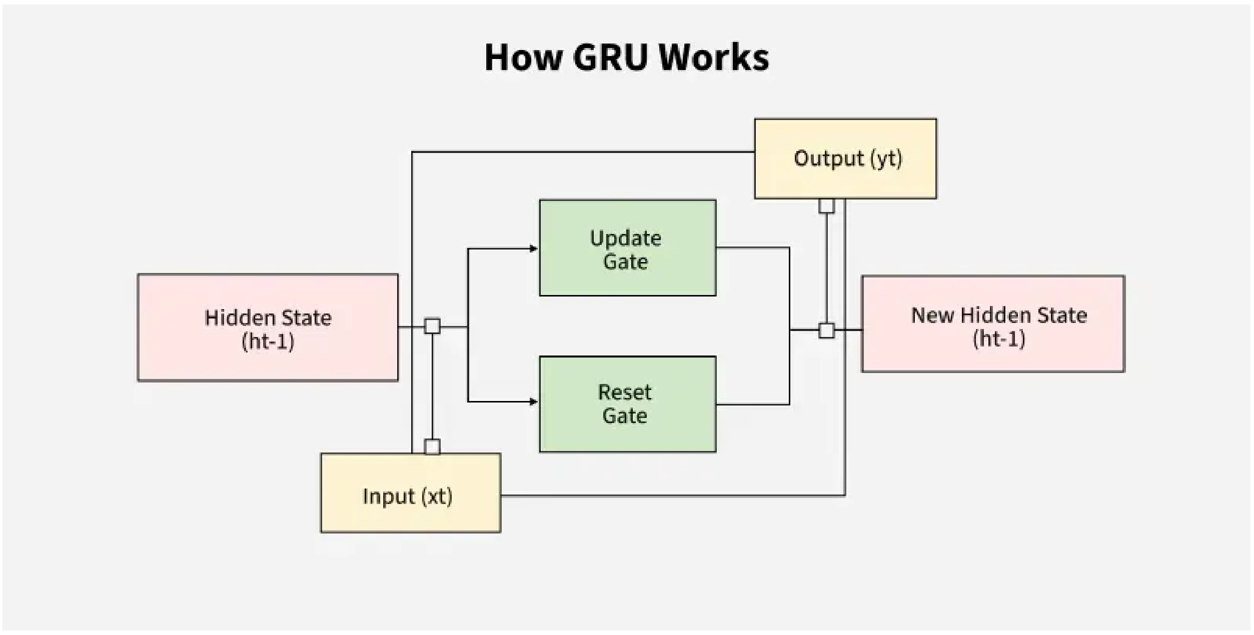


Figure 2.4: Figure from GeeksForGeeks [8]

The GRU cell consists of two gates the update gate and the reset gate, this regulates how previous information and new inputs are combined at each iteration. the update gate z_t determines what will be carried forward from the previous hidden state s_{t-1}, together these gates determine the new hidden state which serves as the short term memory or the output of gru at the time t [6]. The hidden states are then computed as:

$$s_t = z_t \odot s_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

by blending old and new information this way, GRU retains relevant context over long sequences while remaining computationally efficient. This makes the training faster and useful in tasks such as natural language processing, time series prediction, speech recognition and where efficiency and performance are to be balanced [6].

Chapter 3

Method

3.1 SEMMA applied

In this project, SEMMA is used as a lightweight process model to structure the work on the Google Speech Commands classification task. Since the assignment focuses primarily on model development, experimentation, and evaluation rather than deployment or business analysis, SEMMA is adapted to fit these requirements. In particular, the Modify → Model → Assess steps are performed iteratively to allow multiple experiments with different preprocessing strategies, architectures, and hyperparameters. Figure 3.1 shows the SEMMA workflow as applied to the project.

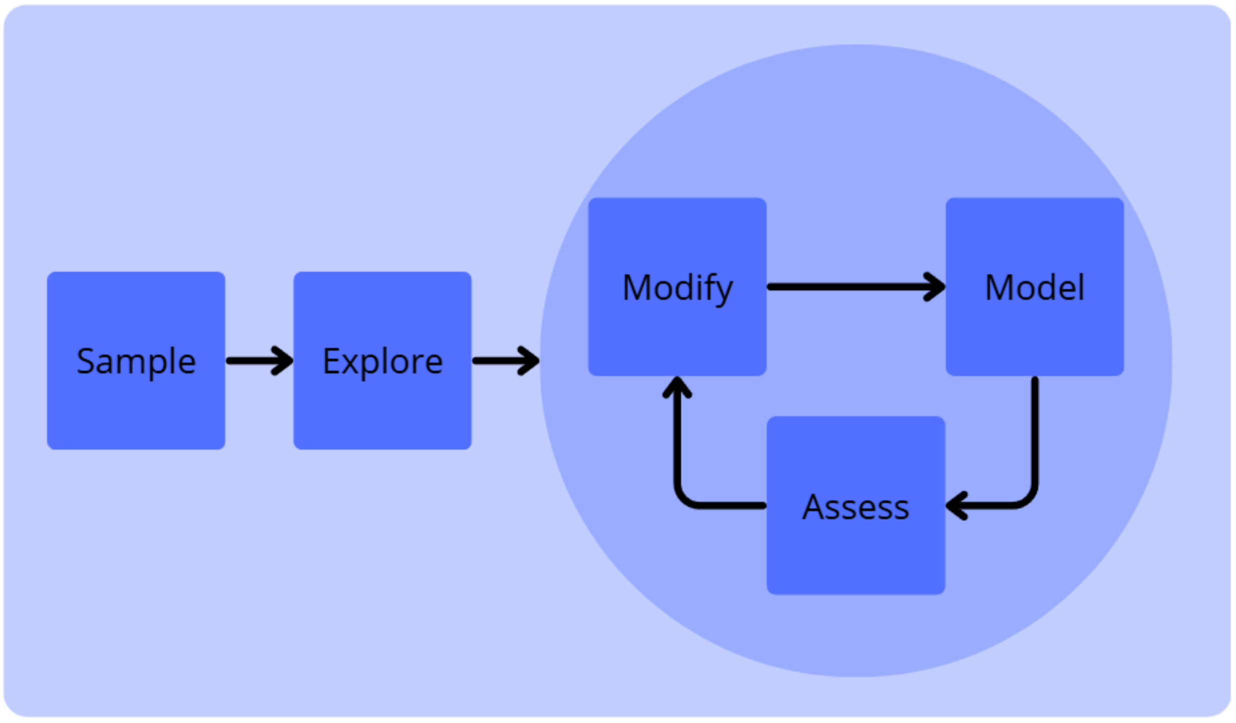


Figure 3.1: SEMMA figure made in Canva

3.1.1 Sample

The sample step will be the step where the data is downloaded. The data will be downloaded from kagglehub, and then the audio files are structured into their predefined folder hierarchy, while the directory paths are registered for subsequent preprocessing steps.

3.1.2 Explore

In the explore step, an exploratory data analysis is done on the sampled data. This includes inspecting the class distribution, verifying the duration and sampling rate of the audio clips, and visualising example waveforms and spectrograms. The goal is to detect potential issues such as class imbalance or corrupted samples, and to gain intuition about the characteristics of the dataset and get a better understanding of the dataset.

3.1.3 Modify

In the modify step, all preprocessing operations will be applied to the raw audio. This includes feature extraction like spectrograms or MFCCs and normalisation of the dataset. Different preprocessing pipelines can be defined here to be compared later in the modelling phase.

3.1.4 Model

In the model stage, diffrent architectures are implemented and trained in the preprocessed data. This includes CNNs and recurrent models such as LSTMs and GRU models. Experiments on the key hyperparameters. This includes the number of layers, learning rate, hidden layers, etc., and will be conducted to find the best architecture for the task.

3.1.4.1 Deep learning

All models in this project will be implemented using PyTorch and trained under consistent conditions to ensure fair comparison. The Adam optimizer will be used with an initial learning rate of 0.001, as it adapts learning rates for each parameter and generally performs well across different architectures. Models will be trained for 50 epochs initially, though this may be adjusted if overfitting is observed during training.

The training approach will follow an iterative methodology: starting with minimal baseline architectures for each model type (CNN, LSTM, GRU), then systematically adding complexity through additional layers and architectural modifications. All models will be trained on the same preprocessed data splits (spectrograms or MFCC), depending on the model, to ensure fair comparison. Each modification and its impact on performance will be documented in the Results section, allowing for analysis of how different architectural choices and hyperparameters influence classification performance.

3.1.4.2 Baseline

During the model training and testing, a simple baseline will be used as a starting point to observe how different changes to the model affect its performance outcome. Such a baseline will be created for each model based on the simplest structure of that model. By using this baseline, it will be easier and clearer to see how the different modifications to the model change the model behaviour.

3.1.5 Assess

Finally, in Assess, we compare these models using metrics such as metrics (accuracy/F1), confusion matrices, and training behavior. The results from this phase are used to analyse the strengths and limitations of each model and to guide further iterations through the Modify and Model steps.

3.1.5.1 Inference

In the inference, the model that achieves the best overall accuracy in the Assess phase will be tested on audio clips recorded with our own voices. These recordings will be preprocessed in exactly the same way as the Google Speech Commands dataset, as in the Modify stage. This makes it possible to see how the final model performs on real-world examples that were not part of the original dataset

Chapter 4

Results

4.1 Dataset

The Google Speech Commands dataset is an open audio collection designed for limited-vocabulary speech recognition and keyword spotting tasks. It consists of over 100,000 one-second WAV clips at 16 kHz, each containing a single spoken English word from a small vocabulary. It contains recordings of digits “zero” to “nine,” commands like “yes,” “no,” “up,” “down,” “on,” “off,” “stop,” “go,” and several other words. Additionally, some separate background noise are recorded. The data is gathered from thousands of different speakers using phone and laptop microphones in realistic environments, and is intended to help researchers train and fairly compare small, on-device models that detect when specific trigger words are spoken [24].

4.2 Sample

The dataset used in this task was the “google speech commands” dataset which was downloaded kagglehub as explained in the method. This was done by using the kagglehub python library. Which looks like this:

```
import kagglehub

def download_dataset():
    path = kagglehub.dataset_download("neehakurelli/google-speech-commands")
    return path
```

Listing 4.1: Kagglehub download dataset

The downloaded data had a total size of 2.06 GB, consisting of a total of 64,727 audio files. This is split into 51,782 files for training and 12,945 files for training and validation. The structure of the downloaded data looks like this:

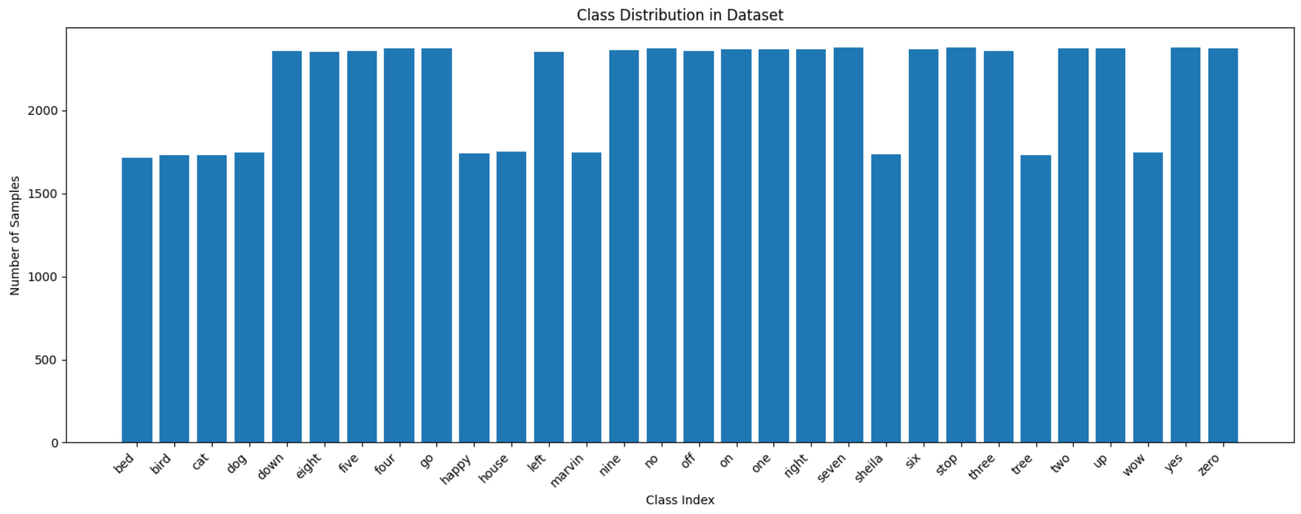
```
data:
  raw:
    _background_noise_
    bed
    bird
    cat
    dog
    down
    eight
    five
    ...
```

Listing 4.2: Data structure

All audio files are of type “wav”, with a duration of exactly 1 second and a sample rate of 16kHz. There are a total of 30 classes, or 31 including background noises. All the classes are the following: [Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop", "Go", "Zero", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Bed", "Bird", "Cat", "Dog", "Happy", "House", "Marvin", "Sheila", "Tree", "Wow", "_background_noise"].

4.3 Explore

To get a better understanding of the dataset we are working with, a short exploratory data analysis of the dataset is provided here. First of all the distribution of the dataset seems to be pretty balanced, meaning no class imbalance that will impact the performance of the model. The distribution is shown in figure 4.3.



```
Class Balance Analysis:
  Most common: 'stop' with 2380 samples
  Least common: 'bed' with 1713 samples
  Imbalance ratio: 1.39:1
  Mean samples per class: 2157.4
  Std deviation: 297.2
```

Listing 4.3: Google speech command class distribution and values

This is all to show how balanced the dataset it, which seems to be fairly balanced as explained above. To get a better overview of how the audio in the dataset varies, and behaves, an additional graph of how the data ranges in frequency is shown bellow in figure 4.1.

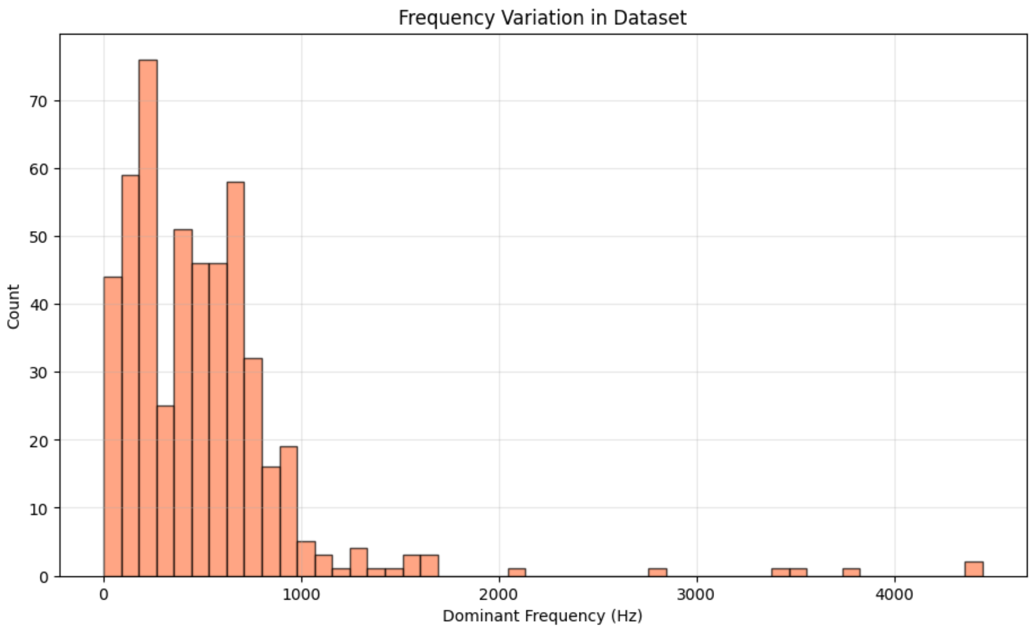


Figure 4.1: Plot of the different frequencies observed in the dataset.

Additionally another plot of the audios amplitude is shown bellow, based on the same reasoning as previous plot. This plot is shown in figure 4.2.

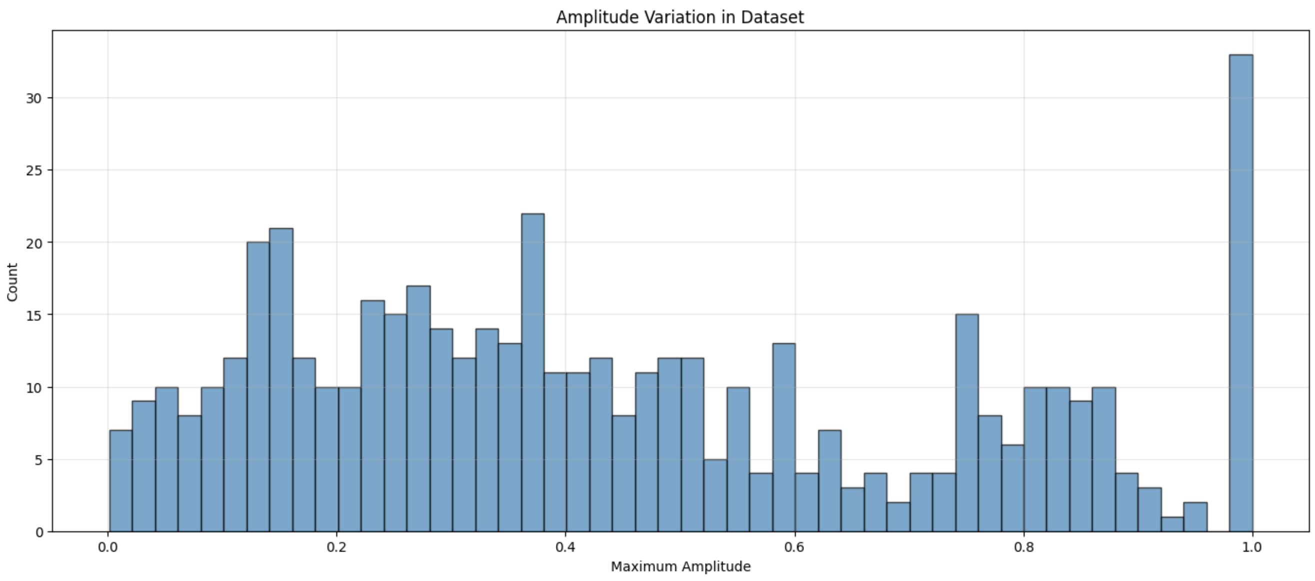


Figure 4.2: Plot of the different amplitudes observed in the dataset.

Looking at the data and the different plots generated from the pure raw data, the dataset seems to be delivering very clean and even data. This is a good thing when training models, as the models performance is heavily based on the quality of the data used to train it. Which in this case seems to be of high quality.

4.4 Modify

From hereonout the data used to train the models, will be preprocessed to two different types: “MFCC”, and “Spectrograms”. This will be the actual data the models receives, and traines on. This will also be used when testing the results afterwards, as the training data and test data must have the same structure. A sample data preprocessed using these different techniques can be seen in figure 4.3.

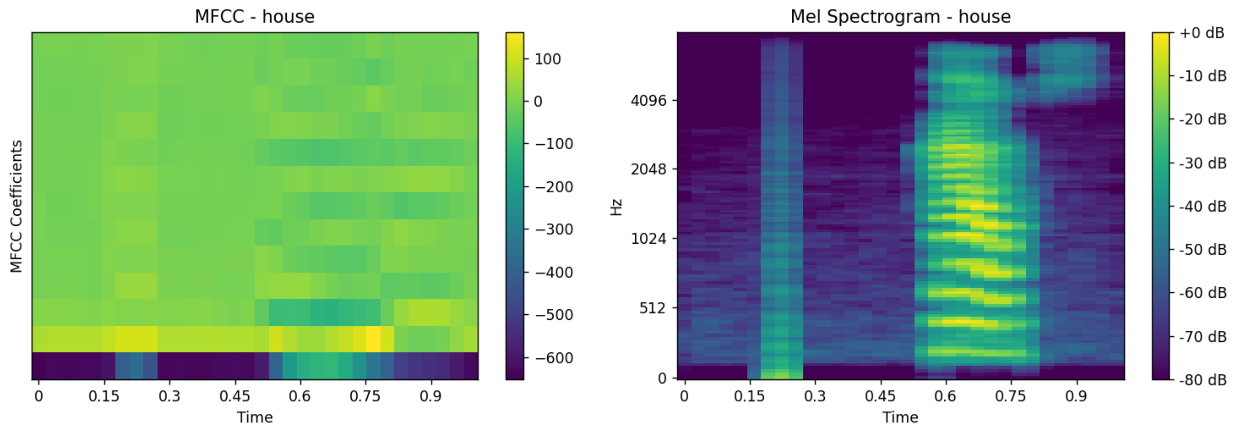


Figure 4.3: MFCC and Mel spectrogram of house

4.5 Model

What kind og models did we train and how The models we trained were a CNN, an LSTM, and a GRU model. These were all made very simple at first and then modified by adding multiple layers and checking different model architectures. For now, all models will be trained for 50 epochs, which may change depending on whether overfitting is observed or not during the training, along with the Adam optimizer. Also learning rate is set to 0.001 for all models. Any change applied will be documented below:

4.5.1 CNN

For the CNN, the first model architecture attempted was a straightforward, minimal approach. This consisted only of a single convolutional layer, along with a single pooling layer, before flattening into an output layer. This was done to create a similar model to the one demonstrated in image 2.1 (background), which will be used to compare how the model performs when we apply changes to it. Basically, this provided a baseline for the rest of the tests on the model.

```
=================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
=================================================================================

CNN                                      [1, 30]                   --
├─Conv2d: 1-1                            [1, 128, 128, 32]         1,280
├─MaxPool2d: 1-2                         [1, 128, 64, 16]          --
├─Flatten: 1-3                           [1, 131072]               --
├─Linear: 1-4                            [1, 30]                   3,932,190
=================================================================================

Accuracy:   69.96%
F1 Score:   0.6999
Precision: 0.7109
Recall:     0.6996

Best test loss: 4.9099
=================================================================================
```

Listing 4.4: CNN minimal

An additional attempt was made by removing the pooling, which halves the image, and observing the behaviour of the model. By doing so, the model provided an accuracy of 62.19% at best. Additionally, the total number of parameters increased dramatically, given it's still just a single hidden layer (Conv2d) and then an output layer (Linear). By applying this pooling, the number of parameters went down from 15,728,670 to 3,932,190. While the accuracy went up from 65.76% to 69.96%. Thus, additional pooling was added for each additional layer later on.

The next thing observed during the training is that the accuracy struggles to converge and continues to fluctuate. An attempt to counter this was made by adding batch normalization to the model after the convolutional layer, before the pooling. This would hopefully make it converge better and faster. Applying these changes lead to the model converging faster and reaching an accuracy of 70.32%, which is already better than the baseline.

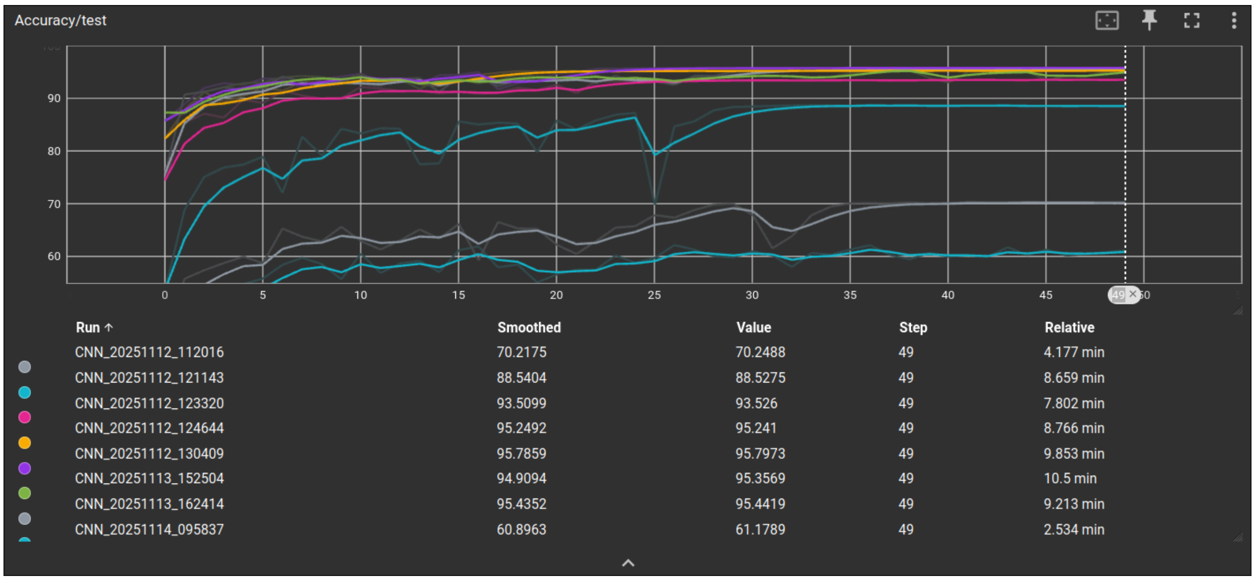
Another observation was made on the activation function used in the model. The originally used activation function was RelU. Swapping this out for another activation function like Sigmoid in the same models generally resulted in around a 2% decrease in accuracy, and an increase in its loss by about 0.05. During the training stage, the model seemed less "stable", and struggled to converge, compared to ReLU. This suggested that the ReLU activation was better suited for this particular case. Swapping the Sigmoid activation for a Tanh activation resulted in similar results to the Sigmoid, just with a more stable learning curve. However, none reached the same performance as RelU.

Then for the next set of experiments, a single new layer was added to the model to observe how it behaved. This was either a convolutional layer or a linear layer. By just adding a single layer, the model accuracy increased by 18.43%. By seeing this increase additional layers was added, and provided these results:

Layers	Accuracy (%)	Precision	Recall	F1-Score	Loss
1c1f	70.32	0.7039	0.7064	0.7032	1.2857
2c1f	88.74	0.8877	0.8886	0.8874	0.6244
3c1f	93.59	0.9361	0.9366	0.9359	0.3125
4c1f	95.30	0.9530	0.9533	0.9530	0.2324
5c1f	**95.80**	**0.9580**	**0.9583**	**0.9580**	0.2210
1c2f	70.14	0.7028	0.7091	0.7014	1.0566
2c2f	88.59	0.8862	0.8871	0.8859	0.4953
3c2f	93.85	0.9359	0.9363	0.9358	0.3043
4c2f	94.92	0.9493	0.9499	0.9492	0.2360
5c2f	95.45	0.9545	0.9547	0.9545	**0.2174**
1c3f	69.87	0.6985	0.7010	0.6987	1.3102
2c3f	88.12	0.8809	0.8818	0.8812	0.6501
3c3f	92.95	0.9298	0.9303	0.9295	0.3421
4c3f	94.67	0.9468	0.9472	0.9467	0.2556
5c3f	95.25	0.9526	0.9529	0.9525	0.2345

Table 4.1: Classification results of multiple attempts: c = convolutional, f = linear

The increase in model performance is very noticeable when adding more layers to the baseline. Especially in the first couple of additional layers, but this stops increasing very fast.



	Run ↑	Smoothed	Value	Step	Relative
●	CNN_20251112_112016	70.2175	70.2488	49	4.177 min
●	CNN_20251112_121143	88.5404	88.5275	49	8.659 min
●	CNN_20251112_123320	93.5099	93.526	49	7.802 min
●	CNN_20251112_124644	95.2492	95.241	49	8.766 min
●	CNN_20251112_130409	95.7859	95.7973	49	9.853 min
●	CNN_20251113_152504	94.9094	95.3569	49	10.5 min
●	CNN_20251113_162414	95.4352	95.4419	49	9.213 min
●	CNN_20251114_095837	60.8963	61.1789	49	2.534 min

Figure 4.4: Model performance on the test dataset, when changing the number of layers. Bottom blue is the baseline.

4.5.1.1 Existing CNN architectures

After attempting multiple iterations of adding different layers to the model, the best performance achieved was 95.80%. This was the model with 5 convolutional layers, along with pooling for each one, and one linear layer for the output. This is the new baseline, which will be used when comparing against the following known CNN architectures:

LeNet5:, Which is a very simple and old model using the CNN arhitecture. This model has been the groundwork for many other CNN architectures such VGG and ResNet. The LeNet-5 model is very simple consisting of 2 convolutional layers, 2 down-sampling layers and 3 fully connected layers. Testing this model, will also show how the newer model architectures compare to the fundamental one. [10, 17]

VGG-16:, Since this model is a versatile model using a CNN architecture. It's a simple model consisting of 13 convolutional layers and 3 fully connected layers, making 16 in total. It has proven to be a strong choice within image classification, and thus fits well to classify the spectrograms generated from the audio samples. However, this model might be too big for the Google Speech Commands dataset, despite its simplicity. [13].

ResNet-18:, Which is a residual network model using a CNN architecture. It consists of 18 layers both convolutional layers and residual blocks. Where the residual blocks allows the input to bypass one or more of the layers, by connecting outputs from layers to input on other layers with the same depth, helping to prevent the vanishing gradient problem. This makes the ResNet-18 operate with a good balance between its depth and the accuracy, and might work well with the audio classification problem [11].

MobileNetV2:, Which is a CNN architecture model designed for high speed while keeping the accuracy high. Generally optimized for mobiles low computational devices. This uses inverted residual blocks which, unlike normal resudal blocks, makes connections to layers of different depths, helping with a more efficient information flow. This model consists of 53 layers, 1 initial convolutional layer, 1+16 bottleneck layers (depthwise + projection for the first one, expansion + depthwise + projection for the rest), final convolutional layer, and then a fully connected layer. [14]

EfficientNet-B0:, Which is a CNN architecture model that uses a compound coefficient to scale its netwrok depth. This achieves a high performance while requireing fewer computational resources. The EfficientNet-B0 consists of 50 layers in total. This is built up of blocks called MBConv1, consisting of 2 convolutional layers, and MBConv6 constisting of 3 convolutional layers each. In total the model has this structure: conv $\rightarrow$ MBConv1 $\rightarrow$ 2·MBConv6 $\rightarrow$ 2·MBConv6 $\rightarrow$ 3·MBConv6 $\rightarrow$ 3·MBConv6 $\rightarrow$ 4·MBConv6 $\rightarrow$ MBConv6 $\rightarrow$ Conv $\rightarrow$ Fully connected. [7]

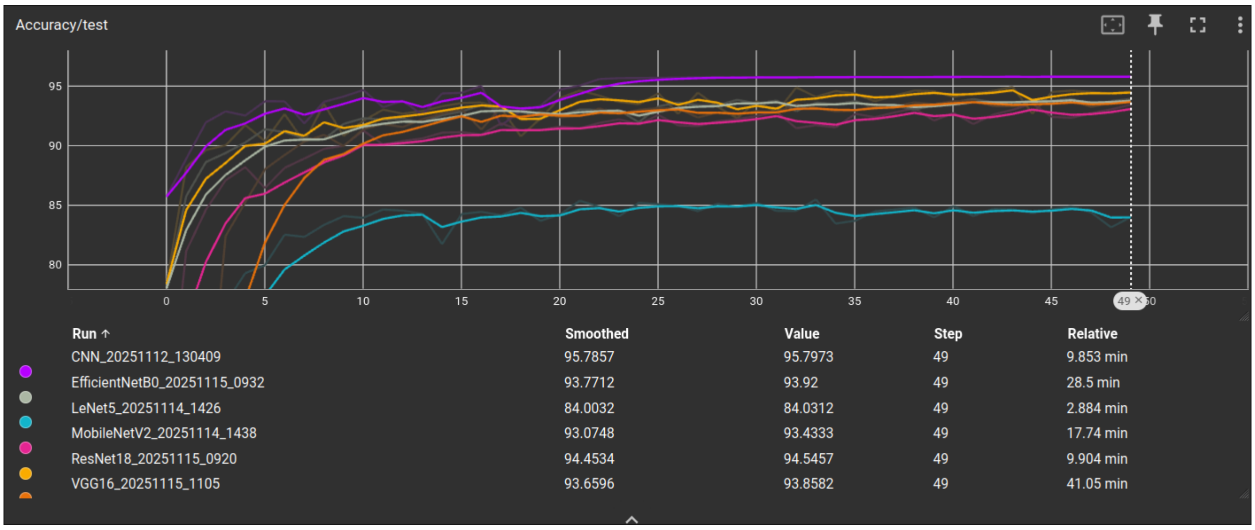
From the comparison, we get these results:

Model	Accuracy (%)	Precision	Recall	F1-Score	Loss
LeNet5	85.50	0.8551	0.8565	0.8550	0.5369
MobileNetV2	93.53	0.9355	0.9362	0.9353	0.3222
ResNet-18	94.92	0.9494	0.9499	0.9492	0.2314
EfficientNet-B0	93.93	0.9393	0.9397	0.9393	0.2872
VGG16	93.86	0.9388	0.9395	0.9386	0.2581
5c1f	**95.80**	**0.9580**	**0.9583**	**0.9580**	**0.2210**

Table 4.2: Classification results comparing various CNN architectures

4.5.1.2 Final takes

Out of all the tested modifications and pre-existing architectures, there were plenty of good candidates. The ResNet-18 performed best out of the existing architectures, but for this particular example, the model that performed the absolute best was the custom architecture created from testing with different numbers of layers in the start of the section. The models' performance throughout the training can be seen in figure 4.4.



	Run ↑	Smoothed	Value	Step	Relative
●	CNN_20251112_130409	95.7857	95.7973	49	9.853 min
●	EfficientNetB0_20251115_0932	93.7712	93.92	49	28.5 min
●	LeNet5_20251114_1426	84.0032	84.0312	49	2.884 min
●	MobileNetV2_20251114_1438	93.0748	93.4333	49	17.74 min
●	ResNet18_20251115_0920	94.4534	94.5457	49	9.904 min
●	VGG16_20251115_1105	93.6596	93.8582	49	41.05 min

Figure 4.5: Accuracy for test dataset for the architecture comparisons

4.5.2 LSTM

For the LSTM, the first model architecture attempted consisted of a single LSTM layer with a hidden size of 128, followed by a single fully connected output layer. This was done to create a baseline model that processes the sequential MFCC features, which will be used to compare how the model performs when we apply changes to it. This model will work as a baseline for the rest of the expiriments.

```
==================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==================================================================================
LSTM                                     [1, 30]                   --
├─LSTM: 1-1                              [1, 32, 128]              73,216
├─Linear: 1-2                            [1, 30]                   3,870
==================================================================================

Accuracy:  89.18%
F1 Score:  0.8918
Precision: 0.8931
Recall:    0.8918

Best test loss: 0.3761
==================================================================================
```

Listing 4.5: LSTM minimal

The baseline LSTM model achieved an accuracy of 89.18%, which is notably higher than the minimal CNN baseline (69.96%). This suggests that the sequential nature of MFCC features works well for LSTM. The model has 77,086 total parameters, significantly fewer than the baseline for CNN at 3,933,470 parameters, making it more efficient while achieving better performance.

After this, an attempt to add dropout was made to prevent overfitting. The first experiment applied dropout only on the output layer with a single LSTM layer, while the second experiment used two LSTM layers with dropout applied between the LSTM layers and on the output. For dropout with 1 LSTM layer, the model achieved the best accuracy of 88.95%, while dropout with 2 LSTM layers got 92.74%. By adding a second LSTM layer with dropout, the accuracy increased from 88.95% to 92.74%. However, the total number of parameters increased from 77,087 to 209,182. Despite the increase in parameters, the additional layer with dropout helped the model achieve higher accuracy.

The next experiment tested different activation functions in the classifier network. Unlike CNNs where activation functions are applied after convolutional layers, LSTMs have built-in activation functions (sigmoid and tanh) [16]. Therefore, the activation function experiments were applied only to the additional classifier layer between the LSTM output and the final prediction layer. Three activation functions were compared: ReLU, Sigmoid, and Tanh.

LSTM_ReLU 92.94%, LSTM_Sigmoid: 92.82%, LSTM_Tanh: 92.90%

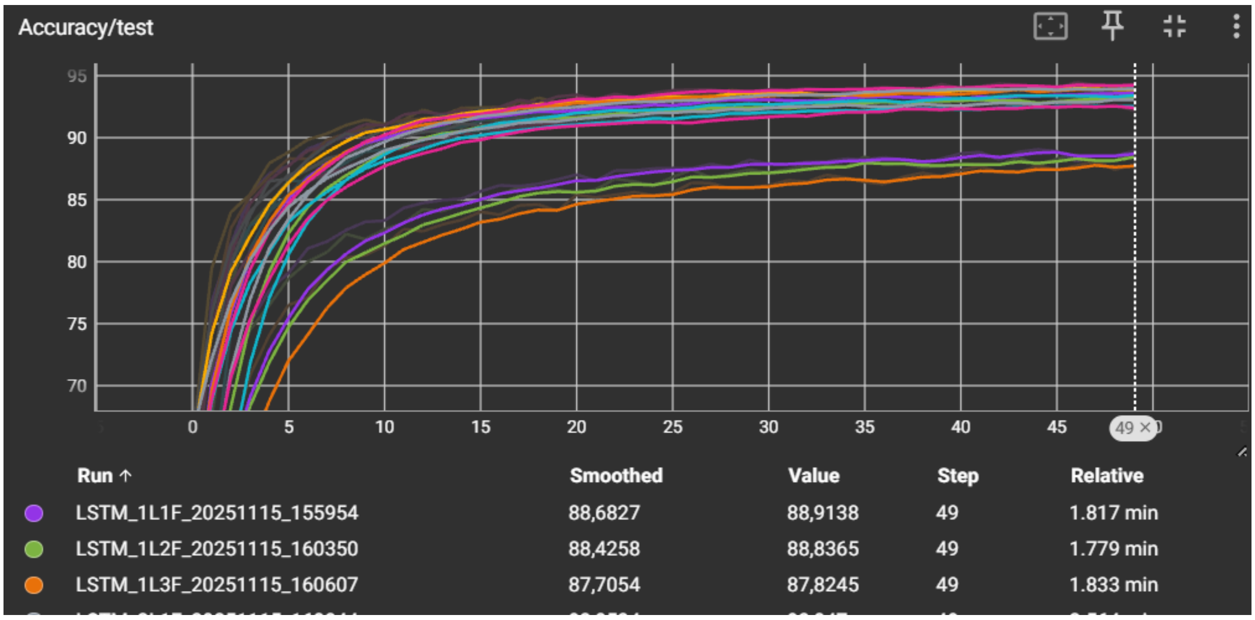
All three activation functions performed similarly, with accuracies ranging from 92.82% to 92.94%. Unlike the CNN experiments, where ReLU significantly outperformed Sigmoid, the LSTM showed minimal sensitivity to the choice of activation function in the classifier. ReLU was selected as the new baseline as it achieved the highest accuracy.

The next experiment tested the number of LSTM layers (L) and fully connected layers (F) in the classifier network. Starting from 1 LSTM layer, additional layers were progressively added, with each configuration tested against 1, 2, and 3 FC layers.

Table 4.3: Classification results for different layer and fully connected configurations

Layers	**Accuracy (%)**	**Precision**	**Recall**	**F1-Score**	**Loss**
1L1F	89.05	0.8924	0.8905	0.8909	0.3739
2L1F	93.34	0.9342	0.9334	0.9336	0.2563
3L1F	94.07	0.9419	0.9407	0.9410	0.2257
4L1F	94.16	0.9423	0.9416	0.9418	0.2318
5L1F	94.47	0.9460	0.9447	0.9450	0.2215
1L2F	88.84	0.8897	0.8884	0.8886	0.3977
2L2F	92.58	0.9274	0.9258	0.9262	0.2956
3L2F	93.66	0.9378	0.9366	0.9369	0.2582
4L2F	94.05	0.9418	0.9405	0.9408	0.2377
5L2F	94.14	0.9430	0.9414	0.9418	0.2378
1L3F	88.10	0.8827	0.8810	0.8812	0.4217
2L3F	92.68	0.9279	0.9268	0.9270	0.3030
3L3F	93.32	0.9340	0.9332	0.9334	0.2850
4L3F	93.60	0.9371	0.9360	0.9362	0.2777
5L3F	93.55	0.9371	0.9355	0.9359	0.2812

The results suggest that increasing LSTM layers improved accuracy, where the jump from 1 to 2 layers got a 4.29% increase (89.05% to 93.34%) in accuracy. However, adding more FC layers consistently decreased performance, suggesting that adding additional FC layers adds unnecessary complexity. The best model was 5L1F with 94.47% accuracy, which becomes the new baseline for the architecture comparisons. In Figure 4.6, the models comparisons through the training can be seen.



	Run ↑	Smoothed	Value	Step	Relative
●	LSTM_1L1F_20251115_155954	88,6827	88,9138	49	1.817 min
●	LSTM_1L2F_20251115_160350	88,4258	88,8365	49	1.779 min
●	LSTM_1L3F_20251115_160607	87,7054	87,8245	49	1.833 min

Figure 4.6: Accuracy for test dataset for the architecture comparisons

4.5.2.1 Existing LSTM architectures

After attempting multiple iterations of adding different layers to the model, the best performance achieved was 94.47% with the 5L1F configuration. This model, with five stacked LSTM layers and a single fully connected output layer, is used as the new baseline when comparing against the following known LSTM-based architectures.

4.5.2.2 Bidirectional and Deep Bidirectional LSTM:

Bidirectional Long Short-Term Memory (BiLSTM) extends the standard LSTM by processing the input sequence in both forward and backward directions and connecting the two outputs at each time step. This bidirectional processing allows the model to capture contextual information from both past and future inputs, which is especially useful in tasks where the full sequence context matters, such as natural language processing and speech recognition. A deeper variant with multiple stacked bidirectional layers (Deep BiLSTM), which increases the model capacity while retaining the same bidirectional processing principle, will also be implemented. [5].

4.5.2.3 ConvLSTM:

Convolutional Long Short-Term Memory (ConvLSTM) combines convolutional neural networks with LSTMs to handle spatiotemporal sequences, such as video frames or other structured time series. Instead of fully connected operations inside the LSTM cell, ConvLSTM uses convolutions to learn spatial hierarchies while still modelling temporal dependencies over time. This makes ConvLSTM particularly suitable for tasks like video prediction, action recognition, and other applications where both spatial structure and temporal evolution are important [5].

4.5.2.4 LSTM with Attention:

LSTMs with attention mechanisms augment the recurrent architecture with an additional module that learns attention weights over the hidden states of the sequence. These weights dynamically highlight which time steps are most relevant for a given prediction, allowing the model to focus on informative parts of long sequences instead of compressing all information into a single hidden state. This improves both interpretability and performance in sequence tasks such as machine translation and sentiment analysis, where contextual information is important [5].

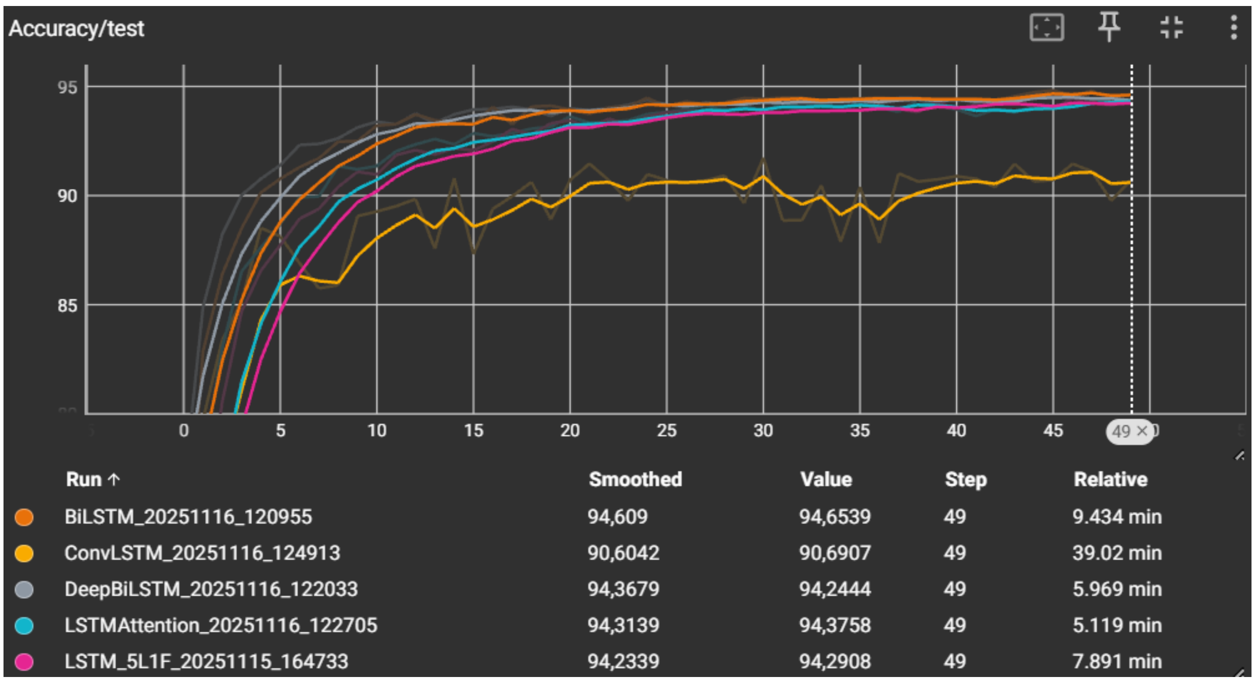
4.5.2.5 Peephole LSTM:

Peephole LSTM extends the traditional LSTM by adding weighted connections from the internal cell state (the Constant Error Carousel) directly to the input, forget, and output gates. In standard LSTM, the gates only see the external inputs and cell outputs, which can limit their ability to decide when to store, forget, or read information if the output gate is closed. By allowing the gates to inspect the current cell state even when the output gate is closed, peephole connections provide additional timing information and have been shown to improve performance on tasks that require learning precise temporal intervals and delays compared to traditional LSTM [15].

From the comparison, we get these results:

Model	Accuracy (%)	Precision	Recall	F1-Score	Loss
Bidirectional LSTM	**94.82**	**0.9490**	**0.9482**	**0.9484**	**0.2220**
Deep Bidirectional LSTM	94.68	0.9475	0.9468	0.9469	0.2140
LSTM with Attention	94.43	0.9460	0.9443	0.9447	0.2088
Peephole LSTM	94.14	0.9426	0.9414	0.9417	0.2355
ConvLSTM	91.73	0.9186	0.9173	0.9176	0.3655
5L1F (Baseline)	94.47	0.9460	0.9447	0.9450	0.2215

Table 4.4: Classification results comparing various LSTM-based architectures



	Run ↑	Smoothed	Value	Step	Relative
●	BiLSTM_20251116_120955	94,609	94,6539	49	9.434 min
●	ConvLSTM_20251116_124913	90,6042	90,6907	49	39.02 min
●	DeepBiLSTM_20251116_122033	94,3679	94,2444	49	5.969 min
●	LSTMAttention_20251116_122705	94,3139	94,3758	49	5.119 min
●	LSTM_5L1F_20251115_164733	94,2339	94,2908	49	7.891 min

Figure 4.7: Accuracy for test dataset for the existing LSTM architecture comparisons

4.5.2.6 Final takes

Since the PyTorch library [19] provides an LSTM layer that already captures most of the structure, changing the classifier part of the network (such as adding more fully connected layers) did not noticeably improve performance, whereas adding more LSTM layers did. The best-performing model was the bidirectional LSTM, which consists of 10 LSTM layers in total (5 forward and 5 backward). In the next section, experiments are conducted on the GRU model, which is a related gated recurrent architecture to the LSTM model.

4.5.3 GRU

The first GRU model used in the experiments included one GRU layer with a hidden size of 128 and a single fully connected output layer. This simple design provides a baseline for measuring how architectural changes affect performance on sequential MFCC data.

```
Layer (type:depth-idx)                    Output Shape             Param #
==================================================================================
GRU_Minimal                               [1, 30]                  --
├─GRU: 1-1                                [1, 32, 128]             54,912
├─Linear: 1-2                             [1, 30]                  3,870
==================================================================================

Epoch: 41
Accuracy:  90.00%
F1 Score:  0.9000
Precision: 0.9013
Recall:    0.9000

Best test loss: 0.3541 (epoch 37)
==================================================================================
```

Listing 4.6: GRU minimal

Following the baseline evaluation, dropout regularization was introduced to address overfitting. The first configuration applied dropout only on the output layer of a single GRU layer model. A second configuration employed two stacked GRU layers, with dropout inserted between the layers as well as on the output. The single layer GRU with dropout reached an accuracy of 90.60%, while the two-layer GRU variant achieved 92.75%. This demonstrates that introducing an additional GRU layer, combined with dropout, led to a notable improvement in performance.

The next experiment investigated the impact of different activation functions in the classifier network. Unlike CNNs, where activation functions follow convolutional layers, GRUs already incorporate sigmoid and tanh within their internal gating mechanisms, meaning additional activation functions can only be applied in the classifier layers between the GRU output and the final prediction layer. Three activation functions were evaluated: ReLU, which helps reduce vanishing gradients; Sigmoid, which compresses values into the range $0, 1$, and Tanh, which outputs values between $1, 1$. The goal of these experiments was to determine which activation function best complements the GRU architecture for audio classification[12].

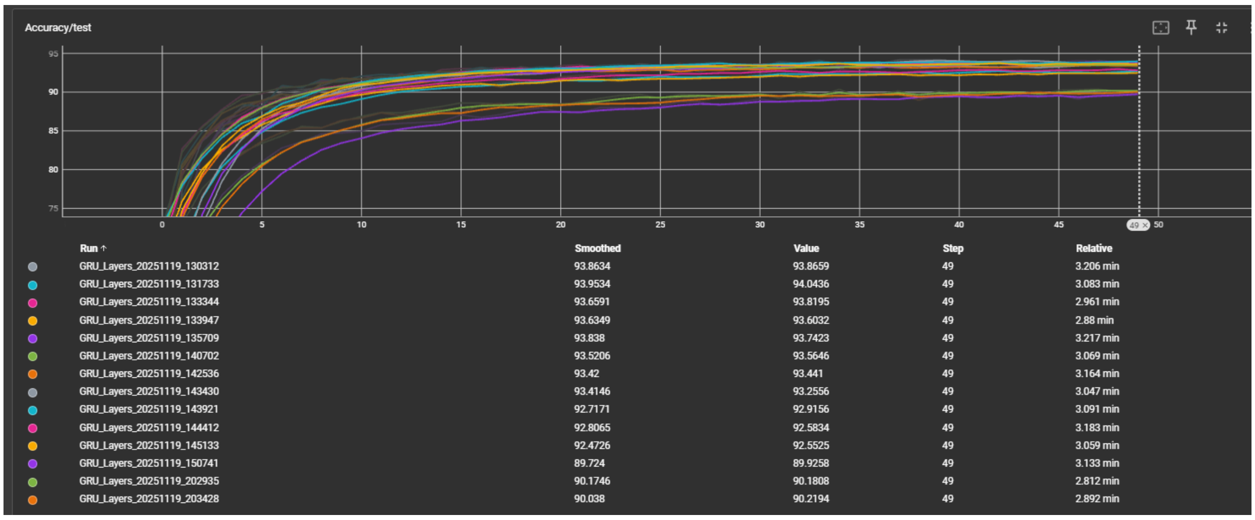
GRU with ReLU: 93.02%, GRU with Sigmoid: 92.92%, GRU with Tanh: 92.89%
All three activation functions produced comparable results, with accuracies ranging from 92.89% to 93.02%. In contrast to the CNN experiments where ReLU clearly outperformed Sigmoid the GRU model showed little sensitivity to the activation function used in the classifier. ReLU was chosen as the new baseline, as it achieved the highest accuracy among the tested options.

The next experiment examined the impact of varying the number of GRU layers (L) and fully connected (FC) layers (F) in the classifier network. Beginning with a single GRU layer and later up to 5, and each configuration was evaluated using 1, 2, and 3 FC layers.

A systematic grid search was conducted to determine the optimal number of GRU layers and fully connected (FC) layers in the classifier network. The experiments tested 1 to 5 stacked GRU layers combined with 1 to 3 FC layers, resulting in 15 different configurations. Stacking multiple GRU layers allows the model to learn increasingly abstract temporal representations at each level, while the FC layers transform these representations into class predictions. This exhaustive search identifies the architecture that best balances model capacity with training efficiency.

Layers	**Accuracy (%)**	**Precision**	**Recall**	**F1-Score**	**Loss**
1l1f	90.22	0.9041	0.9022	0.9026	0.3348
1l2f	90.39	0.9052	0.9039	0.9042	0.3265
1l3f	89.93	0.9004	0.8993	0.8995	0.3356
2l1f	92.61	0.9269	0.9261	0.9262	0.2874
2l2f	93.15	0.9324	0.9315	0.9317	0.2622
2l3f	92.92	0.9318	0.9292	0.9298	0.2654
3l1f	93.75	0.9390	0.9375	0.9378	0.2408
3l2f	93.60	0.9369	0.9360	0.9362	0.2496
3l3f	93.63	0.9383	0.9363	0.9368	0.2523
4l1f	93.91	0.9403	0.9391	0.9394	0.2355
4l2f	93.92	0.9398	0.9392	0.9393	0.2378
4l3f	94.05	0.9420	0.9405	0.9408	0.2341
5l1f	94.09	0.9418	0.9409	0.9411	0.2213
5l2f	94.18	0.9438	0.9418	0.9424	0.2311
5l3f	93.95	0.9410	0.9395	0.9399	0.2428

Table 4.5: Classification results using stacked GRU networks with varying GRU (l) and fully connected (f) layers.



	Run ↑	Smoothed	Value	Step	Relative
●	GRU_Layers_20251119_130312	93.8634	93.8659	49	3.206 min
●	GRU_Layers_20251119_131733	93.9534	94.0436	49	3.083 min
●	GRU_Layers_20251119_133344	93.6591	93.8195	49	2.961 min
●	GRU_Layers_20251119_133947	93.6349	93.6032	49	2.88 min
●	GRU_Layers_20251119_135709	93.838	93.7423	49	3.217 min
●	GRU_Layers_20251119_140702	93.5206	93.5646	49	3.069 min
●	GRU_Layers_20251119_142536	93.42	93.441	49	3.164 min
●	GRU_Layers_20251119_143430	93.4146	93.2556	49	3.047 min
●	GRU_Layers_20251119_143921	92.7171	92.9156	49	3.091 min
●	GRU_Layers_20251119_144412	92.8065	92.5834	49	3.183 min
●	GRU_Layers_20251119_145133	92.4726	92.5525	49	3.059 min
●	GRU_Layers_20251119_150741	89.724	89.9258	49	3.133 min
●	GRU_Layers_20251119_202935	90.1746	90.1808	49	2.812 min
●	GRU_Layers_20251119_203428	90.038	90.2194	49	2.892 min

Figure 4.8: Graphed results of the stacked GRU results

4.5.3.1 Existing GRU architectures

4.5.3.2 Bidirectional GRU

Bidirectional Gated Recurrent Units (BiGRU) extend the standard GRU by processing the input sequence in both forward and backward directions simultaneously. At each time step, the model reads the outputs from both directions, allowing it to capture contextual information from both past and future frames. This bidirectional processing is particularly beneficial for audio classification where understanding the full temporal context improves recognition accuracy, as later sounds may provide context for interpreting earlier ones [2].

4.5.3.3 Deep Bidirectional GRU

Deep Bidirectional GRU builds upon the bidirectional architecture by stacking multiple bidirectional GRU layers. This deeper variant increases the model's capacity to learn hierarchical temporal representations while maintaining the bidirectional processing principle at each layer. The combination of depth and bidirectionality enables the model to capture both temporal patterns in lower layers and high-level abstractions in upper layers, processing information from both temporal directions throughout the entire network hierarchy [2].

4.5.3.4 GRU with Attention

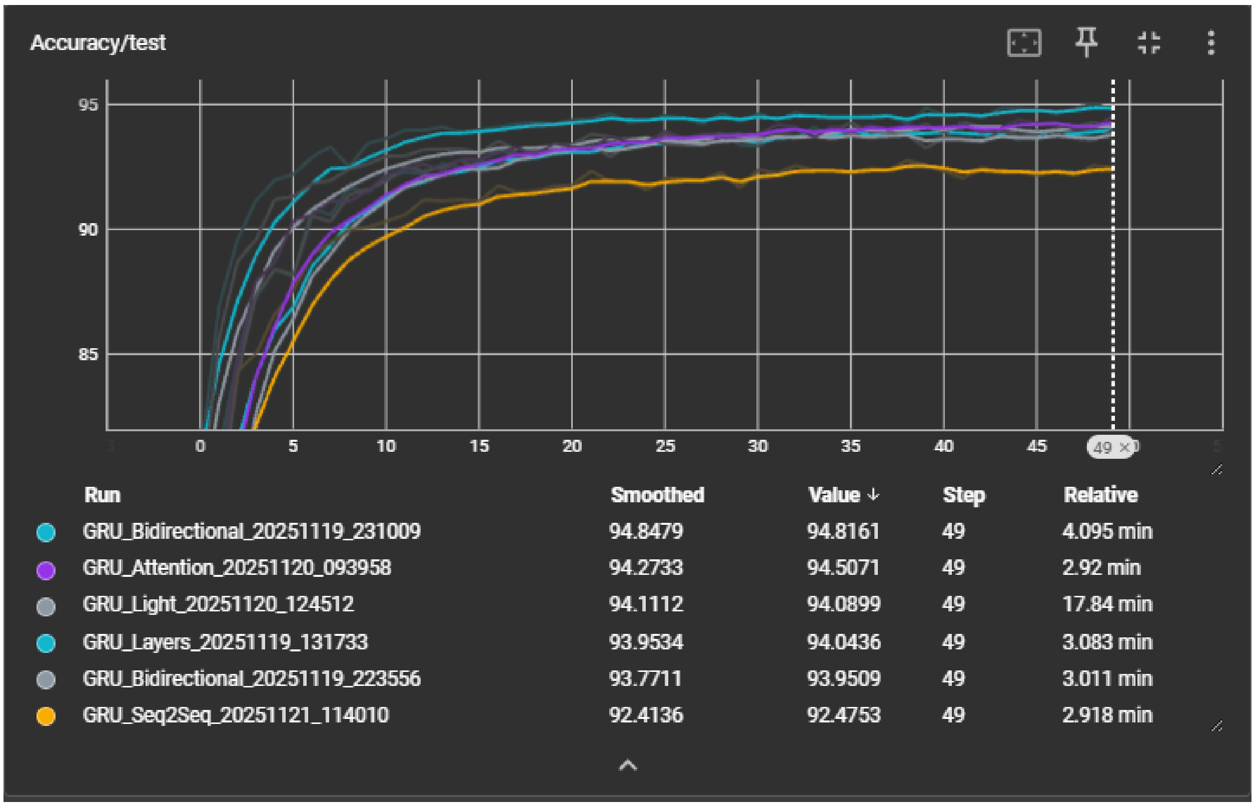
The attention mechanism augments the GRU architecture by learning to dynamically weight the importance of different time steps in the sequence. Instead of relying solely on the final hidden state, attention computes weighted scores across all hidden states, allowing the model to focus on the most relevant parts of the audio sequence for classification. This mechanism is particularly useful for longer sequences where important discriminative features may occur at various temporal positions, as it enables the model to selectively attend to informative frames while down weighting less relevant ones. [22]

4.5.3.5 Seq2Seq GRU

Sequence-to-Sequence Gated Recurrent Units (Seq2Seq GRU) employ an encoder and decoder architecture where two separate GRU networks work in tandem to process sequential data. The encoder GRU processes the input sequence and compresses it into a fixed size context vector that captures the essential information from the entire input. The decoder GRU then takes this context vector and generates an output sequence, which in classification tasks is used to produce the final prediction[12]. This architecture was originally designed for tasks like machine translation where input and output sequences have different lengths, but has been adapted for classification by using the decoder's ability to learn rich representations from the encoded context. The separation of encoding and decoding phases allows the model to learn hierarchical representations and can be particularly effective when combined with attention mechanisms[12].

4.5.3.6 Light GRU

Light Gated Recurrent Unit (Light GRU) is a computationally efficient variant of the standard GRU that reduces model complexity while maintaining competitive performance[20]. Unlike standard GRU which employs separate update and reset gates to control information flow, Light GRU simplifies the gating mechanism by using a single gate to regulate



	Run	Smoothed	Value ↓	Step	Relative
●	GRU_Bidirectional_20251119_231009	94.8479	94.8161	49	4.095 min
●	GRU_Attention_20251120_093958	94.2733	94.5071	49	2.92 min
●	GRU_Light_20251120_124512	94.1112	94.0899	49	17.84 min
●	GRU_Layers_20251119_131733	93.9534	94.0436	49	3.083 min
●	GRU_Bidirectional_20251119_223556	93.7711	93.9509	49	3.011 min
●	GRU_Seq2Seq_20251121_114010	92.4136	92.4753	49	2.918 min

Figure 4.9: Existing GRU architectures compared to baseline

the balance between the previous hidden state and the candidate activation. This architectural simplification significantly reduces the number of parameters and computational operations, making Light GRU particularly suitable for resource constrained environments or real time applications. The simplified gating mechanism maintains the core recurrent structure's ability to capture temporal dependencies while offering faster training and inference times, though potentially at the cost of some representational capacity compared to the full GRU architecture [20].

From the comparison, we get these results:

Model	Accuracy (%)	Precision	Recall	F1-Score	Loss
Bidirectional GRU	93.95	0.9406	0.9395	0.9397	0.2458
Deep Bidirectional GRU	**94.81**	**0.9490**	**0.9481**	**0.9484**	**0.2212**
GRU with Attention	94.51	0.9427	0.9451	0.9455	0.2179
seqs2eq GRU	92.41	0.9258	0.9241	0.9249	0.3203
Li GRU (Light GRU)	94.09	0.9425	0.9411	0.9412	0.228
5l2f (baseline)	94.18	0.9438	0.9418	0.9424	0.2311

Table 4.6: Classification results comparing various GRU-based architectures

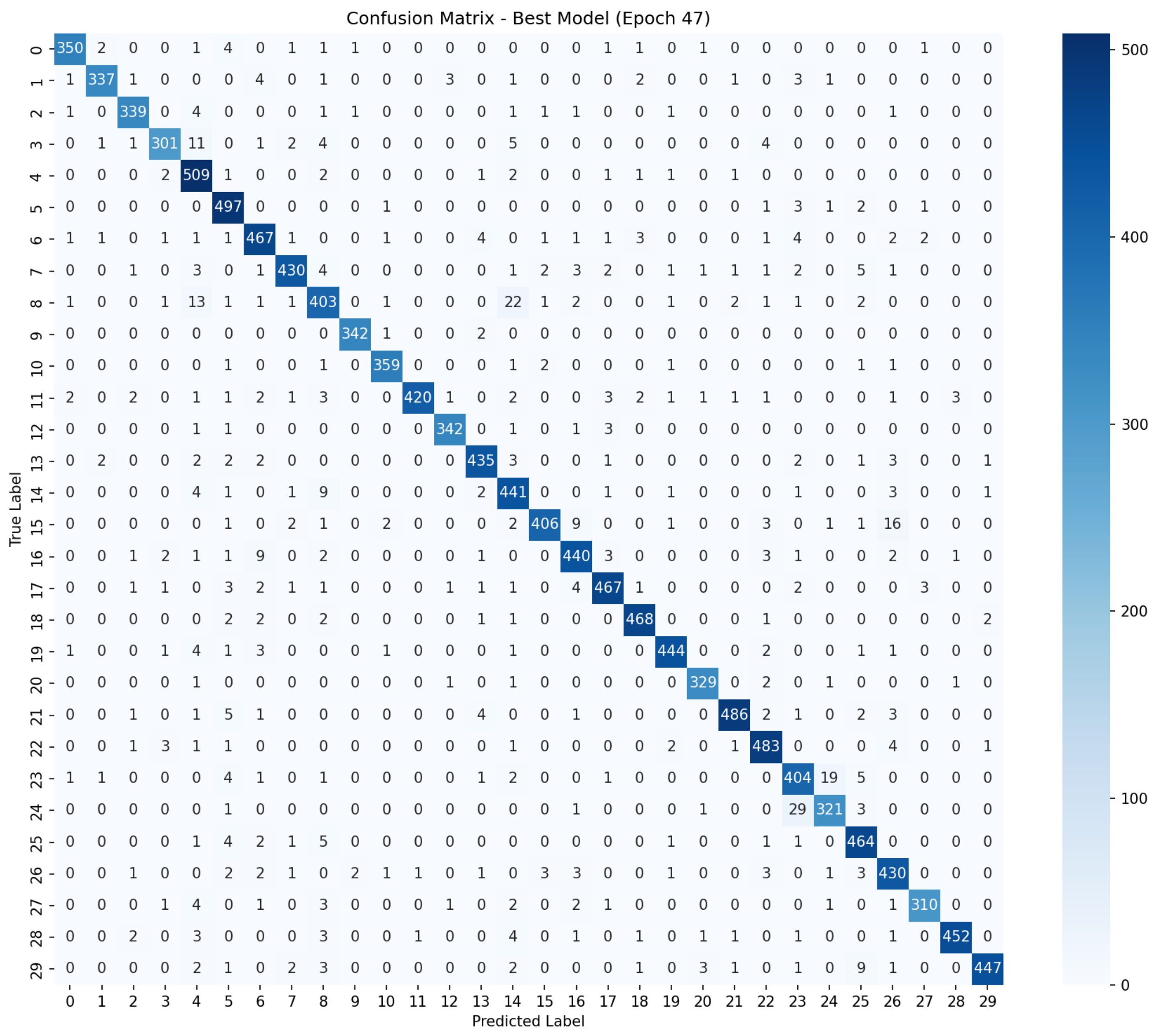
4.5.3.7 Final takes

Since the PyTorch library [3] also provides a GRU layer that already captures most of the network structure, modifying the classifier section of the model, for example by adding more fully connected layers, did not lead to any significant improvement in performance. In contrast, increasing the number of GRU layers did produce better results. The model with the highest performance was the deep-bidirectional GRU, which contains a total of ten GRU layers, five processing the input in the forward direction and five processing it in the backward direction.

4.6 Assess

From all the previous experiments and iterations, here are the best performing model from each model type compared against one another in the same table. This shows how far off of one another each of the models were, showing that all three architectures reach a similar performance, the custom CNN model still achieved a whole percentage more than the two other models. This suggests that for this particular example the feature extraction from spectrograms and convolutional layers provided the best performance. However the existing architectures provided strong competitors, and are most likely the better choice for overall everyday use

As the CNN custom model performed the best out of the various models attempted in this assignment, its confusion matrix showing predictions vs actual labels, is also included and shown in fig 4.10.



True \ Predicted	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	350	2	0	0	1	4	0	1	1	1	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	1	0	0
1	1	337	1	0	0	0	4	0	1	0	0	0	3	0	1	0	0	0	2	0	0	1	0	3	1	0	0	0	0	0
2	1	0	339	0	4	0	0	0	1	1	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	1	0	0	0
3	0	1	1	301	11	0	1	2	4	0	0	0	0	0	5	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0
4	0	0	0	2	509	1	0	0	2	0	0	0	0	1	2	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0
5	0	0	0	0	0	497	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	3	1	2	0	1	0	0
6	1	1	0	1	1	1	467	1	0	0	1	0	0	4	0	1	1	1	3	0	0	0	1	4	0	0	2	2	0	0
7	0	0	1	0	3	0	1	430	4	0	0	0	0	0	1	2	3	2	0	1	1	1	1	2	0	5	1	0	0	0
8	1	0	0	1	13	1	1	1	403	0	1	0	0	0	22	1	2	0	0	1	0	2	1	1	0	2	0	0	0	0
9	0	0	0	0	0	0	0	0	0	342	1	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	1	0	0	1	0	359	0	0	0	1	2	0	0	0	1	0	0	0	0	0	1	1	0	0	0
11	2	0	2	0	1	1	2	1	3	0	0	420	1	0	2	0	0	3	2	1	1	1	1	0	0	0	1	0	3	0
12	0	0	0	0	1	1	0	0	0	0	0	0	342	0	1	0	1	3	0	0	0	0	0	0	0	0	0	0	0	0
13	0	2	0	0	2	2	2	0	0	0	0	0	0	435	3	0	0	1	0	0	0	0	0	2	0	1	3	0	0	1
14	0	0	0	0	4	1	0	1	9	0	0	0	0	2	441	0	0	1	0	1	0	0	0	1	0	0	3	0	0	1
15	0	0	0	0	0	1	0	2	1	0	2	0	0	0	2	406	9	0	0	1	0	0	3	0	1	1	16	0	0	0
16	0	0	1	2	1	1	9	0	2	0	0	0	0	1	0	0	440	3	0	0	0	0	3	1	0	0	2	0	1	0
17	0	0	1	1	0	3	2	1	1	0	0	0	1	1	1	0	4	467	1	0	0	0	0	2	0	0	0	3	0	0
18	0	0	0	0	0	2	2	0	2	0	0	0	0	1	1	0	0	0	468	0	0	0	1	0	0	0	0	0	0	2
19	1	0	0	1	4	1	3	0	0	0	1	0	0	0	1	0	0	0	0	444	0	0	2	0	0	1	1	0	0	0
20	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	329	0	2	0	1	0	0	0	1	0
21	0	0	1	0	1	5	1	0	0	0	0	0	0	4	0	0	1	0	0	0	0	486	2	1	0	2	3	0	0	0
22	0	0	1	3	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	2	0	1	483	0	0	0	4	0	0	1
23	1	1	0	0	0	4	1	0	1	0	0	0	0	1	2	0	0	1	0	0	0	0	0	404	19	5	0	0	0	0
24	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	29	321	3	0	0	0	0
25	0	0	0	0	1	4	2	1	5	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	464	0	0	0	0
26	0	0	1	0	0	2	2	1	0	2	1	1	0	1	0	3	3	0	0	1	0	0	3	0	1	3	430	0	0	0
27	0	0	0	1	4	0	1	0	3	0	0	0	1	0	2	0	2	1	0	0	0	0	0	0	1	0	1	310	0	0
28	0	0	2	0	3	0	0	0	3	0	0	1	0	0	4	0	1	0	1	0	1	1	0	1	0	0	1	0	452	0
29	0	0	0	0	2	1	0	2	3	0	0	0	0	0	2	0	0	0	1	0	3	1	0	1	0	9	1	0	0	447

Figure 4.10: Confusion matrix of best model

Model	**Accuracy (%)**	**Precision**	**Recall**	**F1-Score**	**Loss**
Bidirectional LSTM	94.82	0.9490	0.9482	0.9484	0.2220
5c1f (CNN)	**95.80**	**0.9580**	**0.9583**	**0.9580**	**0.2210**
Deep Bidirectional GRU	94.81	0.9490	0.9481	0.9484	0.2212

Table 4.7: Classification results comparing various LSTM-based architectures

4.6.1 Inference

The custom CNN model achieved the best accuracy among the different models and existing architectures. Therefore, to see how this model performers outside the dataset and training set, the model was tested using real-world audio recordings from other people to see how it performs. To do this, however, there are a couple of things that must be done prior to the prediction. As all the audio files are exactly one second each, the new incoming audio files are therefore clipped to be exactly the same length, and then they are preprocessed the same way the dataset was during the training. This audio clipping was done automatically by clipping the edges around the highest amplitude in the audio that is above a set threshold. Doing this inference seemed to give very accurate results:

```
Example automatic audio clipping:
Peak at 0.38s (too early, adjusting window)
Clip from 0.00s to 1.00s (duration: 1.00s)

Actual class: bed
Predicted class: bed, with confidence 97.79%

Actual class: bird
Predicted class: bird, with confidence 100.00%

Actual class: stop
Predicted class: stop, with confidence 98.89%

Actual class: seven
Predicted class: seven, with confidence 100.00%
```

Listing 4.7: Inference example

Chapter 5

Discussions

5.1 Sample

As observed in the downloaded dataset, the google speech commands were providing a reliable set of files to the models, as this dataset was both small to download and containing plenty of well structured data. The downloaded data appeared to be evenly spread, which meant there was very little risk of bias towards any class being overrepresented. Such impalances are very common in unorganized data, and could have been a potential weakness in this task aswell. However, this was obviously not the case as the data was structured nicely and evenly, allowing consistent feature extraction from all the samples. This made it easier to do the task, and it is much faster than generating the data by hand.

5.2 Explore

During the exploratory analysis of the data, it was made clear that most of the features for the audio was even across all the samples. This was shown as all samples were of one second long, and most audio files had dominant frequencies below 1000Hz, maintaining within the human speaking range. The amplitudes observed are also an indicator of the consistent quality of the recorded samples. This again is another indicator of the even and balanced factors of this dataset. However, training the model on only perfect data, would make it hard for the model to recognize spoken words that arent perfect. This was a concern when observing the data. Nonetheless, luckily, the dataset included some outliers of imperfect spoken words, and additionally it included a background noise class, which meant the model was also be able to distinguish between background noise and actually speaking.

5.3 Modify

As not all models would be able to classify raw waveform data, the data was preprocessed and modified into mel spectrograms and MFCC. These modifications did provide solid results when passed to the models. However, additional modifications could have been added to observe how different modifications would impact the performance. Out of the two different modifications, the choice of preprocessing of the data influenced the model performance heavily. Where the CNNs had benefits from the high spatial information from spectrograms, and the other models performed more efficiently with the more compact MFCC version. This shows how important using the correct preprocessing of the data is for the model to perform its best.

5.4 Models

5.4.1 CNN

When training the different models, the CNN was the first model to be modified, and also the model that reached the highest performance overall. When modifying this model, the factor that had the biggest impact and made the biggest improvement in the model performance was the number of convolutional layers added to the model. This could be due to the fact that adding more convolutional layers leads to the model being able to extract more distinct features form the image before giving it a classification. Surprisingly, adding more fully connected layers to this feature extraction didn't improve the model performance at all. This could mean that the convolutional layers were already able to extract the important features from the images, and any additional linear layer would just introduce unnecessary complexity. Even though the layer depth provided the biggest impact, other techniques such as batch normalization and pooling made the training more stable and efficient. Other techniques, such as dropout, didn't have as big an impact on this particular model.

Compared to the existing architectures, the model created by testing and adding single modifications at a time provided the best results. This could be an indicator that the Google speech commands dataset simply requires a simple model for higher accuracy of classification. Meaning most existing architectures would probably be more complex than necessary for this particular dataset.

5.4.2 LSTM

The LSTM models consistently delivered strong results on the MFCC features, and they quickly outperformed the CNN baseline. The single-layer LSTM model already reached an accuracy of 89.18% compared to the CNN baseline at 69.96%. This indicates that the sequential representation of MFCCs is well-suited for classifying audio on the Google Speech Command dataset. Adding more LSTM layers turned out to be the most effective way to improve performance on the LSTM model. The jump from 1 to 2 layers gave a clear boost in accuracy, and the best configuration of layers (5L1F) reached 94.47%. In contrast, stacking additional fully connected layers after the LSTM generally hurts performance, suggesting that most of the learning happens inside the recurrent LSTM layers themselves, and that extra fully connected layers mainly add unnecessary complexity and overfitting risk. Adding dropout to the model helped increase performance, but at the cost of a noticeable increase in parameter count, showing that there is a trade-off between capacity and efficiency.

When comparing the best-performing LSTM model (5L1F) with existing LSTM-based architectures, it turned out that existing architectures just added minimal improvements. Bidirectional LSTM achieved the best overall accuracy at 94.82%. This slightly outperforms the 5L1F LSTM baseline, which suggests that having access to both past and future context provides a small but measurable benefit on this task. Another reason it might have performed better than the 5L1F model is that the bidirectional model consists of 10 LSTM layers, compared to the baseline 5L1F with only 5 LSTM layers. The other existing LSTM architectures (ConvLSTM, LSTM with Attention and Peephole LSTM) did not offer any improvements, indicating that more complex models may not be necessary for the Google Speech Command dataset.

5.4.3 GRU

The GRU models showed good results with increased GRU recurrent layers and not so much improvement with fully connected layers. the single layer baseline GRU model started of with an accuracy of 89.73% which is an improvement considering the other model's baseline. By adding more layers the accuracy surely increased for each increase, but the increase slowed down particularly around the 4 to 5 layer mark 4.8. The connected layers didn't seem to improve the results that much during the iterations of GRU layer 5, so it performed best with 2f rather than 3f, so the best performing configuration here was 5l2f which reached an accuracy of 94.04%. Adding Dropout didn't increase the performance by much but did negatively affect the performance with more parameters so it was therefore not implemented in the other variations.

Comparing the best performing model (5l2f) to the existing GRU architectures. it turns out that existing architectures give some improvements in terms of accuracy like for example GRU with deep bidirectional GRU which uses stacked bidirectional layers to improve the context understanding of the model and therefore gets the best accuracy of 94.81%. it outperforms the best baseline model (5l2f) by a 0.72% increase. Another model that did better than the baseline was the GRU with attention, tries to focus mostly on the most relevant part of the input sequence when making a prediction, this recieved an accuracy score of 94.51% . Though these models perform better, they are also less efficient. While bidirectional is better performing it uses double the amount of layers for since it has both backward layers and forward layers and GRU with attention has utilizes the attention function. The other models existing GRU models like Li gru, seq2seq and standard bidirectional GRU didnt provide any better results, which may mean that there isnt a better performing model that already exists.

5.5 Assess

From the assess the confusion matrix shows that the model for the most part correctly classifies the various command classes, indicated by its diagonal colored cells. However, even though it performed strong most of the time, this also shows that its not able to correctly classify all the different audio files correctly. This suggests that even though the CNN can capture a lot of the features to distinguish between speech patterns, there are still unidentified challenges that the model weren't able to overcome. Additionally, the confusion matrix shows that the model does not show any bias toward a particular command but varies over them all.

Even though all models achieved similar results, the performance jump from the CNN model might be highly impacted by the type of preprocessing used, as the other models used MFCC, while the CNN used the mel spectrogram. Which could again be an indicator that the spectrograms were superior in this particular case.

5.5.1 Inference

During the inference the model seemed to be performing really well with the new unseen data it received. However, as said in the inference chapter this would most likely not be the case if the audio input wasn't cropped to one second prior to given to the model. Moreover, since this model performed so well with the new unseen data, it shows that the model was successful in learning generalizable features from the given training data, and doesn't just memorize the data directly from the dataset. This shows the model is capable in learning multiple features to distinguish between speech patterns.

Chapter 6

Conclusions

This project investigated deep learning architectures for audio classification using the Google Speech Commands dataset. Using the SEMMA methodology to split the project into sample, explore, modify, model and assess. The research question asked which architecture achieves the best classification performance and how key hyperparameters influence results.

The experiments demonstrated that all three model types, CNNs, LSTMs, and GRUs, achieved high accuracies on this task, ranging from 94.47% to 95.80%. The custom CNN architecture with five convolutional layers and one fully connected layer achieved the highest accuracy of 95.80%, slightly better than the recurrent architectures. This suggests that for this particular dataset, spatial feature extraction from mel spectrograms through convolutional operations was more effective than temporal sequence modeling from MFCC features.

The iterations and multiple experiments provided some important factors. First, architectural depth proved to be more important than the width for all three model types, meaning adding more recurrent or convolutional layers consistently improved performance. However, adding extra fully connected layers often resulted in unnecessary complexity, which didn't change the accuracy or decreased it instead. Second, simple architectures seemed to perform better for this dataset, as many complex existing architectures failed to outperform the simple custom models, suggesting that the Google Speech Commands dataset may not require any sophisticated architectures.

Additional inference testing with audio recordings from outside the training data showed that the model successfully managed to generalize its feature extraction to data that's not just in the training data, by correctly classifying the new audio clips. However, this strong performance depended heavily on preprocessing the input audio to match the one-second format of the training data, indicating the importance of consistent data formatting for practical use.

Overall, this work demonstrates that good preprocessing, appropriate feature representation, and relatively simple architectures can achieve high accuracy on audio classification tasks. The SEMMA-based approach provided valuable insights into how different design choices impact model behavior and showed that sometimes simpler, well-tuned models outperform more complex existing architectures for specific datasets such as the Google speech commands dataset.

Bibliography

[1] Ana Azevedo and Manuel Filipe Santos. "KDD, SEMMA and CRISP-DM: A Parallel Overview." In: *Proceedings of the IADIS European Conference on Data Mining.* 2008, pp. 182–185. URL: https://dblp.uni-trier.de/db/conf/iadis/dm2008.html#AzevedoS08.

[2] *Bi-Directional Gated Recurrent Unit - an overview | ScienceDirect Topics.* [Online; accessed 2025-11-20]. URL: https://www.sciencedirect.com/topics/computer-science/bi-directional-gated-recurrent-unit.

[3] PyTorch Contributors. *GRU — PyTorch 2.9 documentation.* [Online; accessed 2025-11-25]. Jan. 2023. URL: https://docs.pytorch.org/docs/stable/generated/torch.nn.GRU.html.

[4] Steven Davis and Paul Mermelstein. "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences." In: *IEEE transactions on acoustics, speech, and signal processing* 28.4 (1980), pp. 357–366.

[5] Exxact Corporation. *5 Types of LSTM Recurrent Neural Networks and What to Do With Them.* https://www.exxactcorp.com/blog/Deep-Learning/5-types-of-lstm-recurrent-neural-networks-and-what-to-do-with-them. Accessed: 2025-11-16.

[6] *Gated Recurrent Unit - an overview | ScienceDirect Topics.* [Online; accessed 10. Nov. 2025]. Nov. 2025. URL: https://www.sciencedirect.com/topics/computer-science/gated-recurrent-unit.

[7] GeeksforGeeks. *Efficientnet Architecture.* https://www.geeksforgeeks.org/computer-vision/efficientnet-architecture/. Accessed: 2025-11-14. 2025.

[8] GeeksforGeeks. *Gated Recurrent Unit Networks - GeeksforGeeks.* [Online; accessed 2025-11-25]. July 2019. URL: https://www.geeksforgeeks.org/machine-learning/gated-recurrent-unit-networks/.

[9] GeeksforGeeks. *Introduction to Recurrent Neural Network.* https://www.geeksforgeeks.org/machine-learning/introduction-to-recurrent-neural-network/. Accessed: 2025-11-14. 2024.

[10] GeeksforGeeks. *LeNet-5 Architecture.* https://www.geeksforgeeks.org/computer-vision/lenet-5-architecture/. Accessed: 2025-11-14. 2024.

[11] GeeksforGeeks. *ResNet18 from Scratch Using PyTorch.* https://www.geeksforgeeks.org/deep-learning/resnet18-from-scratch-using-pytorch/. Accessed: 2025-11-14. 2025.

[12] GeeksforGeeks. *seq2seq Model - GeeksforGeeks.* [Online; accessed 2025-11-25]. Dec. 2018. URL: https://www.geeksforgeeks.org/machine-learning/seq2seq-model-in-machine-learning/.

[13] GeeksforGeeks. *VGG-16 | CNN Model.* https://www.geeksforgeeks.org/computer-vision/vgg-16-cnn-model/. Accessed: 2025-11-14. 2024.

[14] GeeksforGeeks. *What Is Mobilenet V2?* https://www.geeksforgeeks.org/computer-vision/what-is-mobilenet-v2/. Accessed: 2025-11-14. 2025.

[15] Felix A. Gers, Nicol N. Schraudolph, and Jurgen Schmidhuber. "Learning Precise Timing with LSTM Recurrent Networks." In: *Journal of Machine Learning Research* 3 (2002), pp. 115–143.

[16] Niklas Lang. "Understanding LSTM: Long Short-Term Memory Networks for Natural Language Processing." In: *Towards Data Science* (June 2022). URL: https://towardsdatascience.com/an-introduction-to-long-short-term-memory-networks-lstm-27af36dde85d/.

[17] Yann LeCun et al. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[18] *Mel Frequency Cepstral Coefficient - an overview | ScienceDirect Topics*. [Online; accessed 10. Nov. 2025]. Nov. 2025. URL: https://www.sciencedirect.com/topics/computer-science/mel-frequency-cepstral-coefficient.

[19] PyTorch. *torch.nn.LSTM*. https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html. Accessed: 2025-11-16.

[20] Mirco Ravanelli et al. "Light Gated Recurrent Units for Speech Recognition." In: *IEEE Transactions on Emerging Topics in Computing* 2 (Mar. 2018). DOI: 10.1109/TETCI.2017.2762739.

[21] *Recurrent Neural Network - an overview | ScienceDirect Topics*. [Online; accessed 10. Nov. 2025]. Nov. 2025. DOI: 10.1016/B978-0-323-99560-3.00007-7.

[22] Abbas Kazemi Khoshouei Shiva Nosouhian Fereshteh Nosouhian. *A Review of Recurrent Neural Network Architecture for Sequence Learning: Comparison between LSTM and GRU*. [Online; accessed 2025-11-20]. July 2021. URL: https://www.researchgate.net/publication/353214051_A_Review_of_Recurrent_Neural_Network_Architecture_for_Sequence_Learning_Comparison_between_LSTM_and_GRU.

[23] *Spectrogram - an overview | ScienceDirect Topics*. [Online; accessed 10. Nov. 2025]. Nov. 2025. DOI: 10.1533/9781782422716.116.

[24] Pete Warden. "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition." In: *arXiv preprint arXiv:1804.03209* (2018). URL: https://arxiv.org/abs/1804.03209.

[25] *Waveform - an overview | ScienceDirect Topics*. [Online; accessed 10. Nov. 2025]. Nov. 2025. DOI: 10.1016/j.clp.2006.12.002.

[26] Wikipedia contributors. *Waveform*. https://en.wikipedia.org/wiki/Waveform. Accessed: 2025-11-17. 2025.

[27] Yang Yang et al. "Parameterised time-frequency analysis methods and their engineering applications: A review of recent advances." In: *Mechanical Systems and Signal Processing* 119 (2019), pp. 182–221. ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2018.07.039. URL: https://www.sciencedirect.com/science/article/pii/S088832701830445X.

[28] Zilliz. *Convolutional Neural Network*. https://zilliz.com/glossary/convolutional-neural-network. Accessed: 2025-11-14. 2025.