Deep Learning Project - Transformers

Natalia Safiejko, Wojciech Grabias

April 2025

Contents

1 Introduction **4**

1.1 Theoretical Background . . . 4

1.1.1 Core Architecture . . . 4

1.1.2 Key Innovations . . . 4

1.1.3 Advantages . . . 5

1.2 Existing approaches . . . 5

1.2.1 Speech recognition based on Convolutional neural networks and MFCC algorithm . . . 5

1.2.2 Key Word Spotting through Image Recognition . . . 5

1.2.3 Speech Command Recognition in Computationally Constrained Environments with a Quadratic Self-organized Operational Layer . . . 6

2 Dataset Description **6**

2.1 Number of Samples and Classes . . . 6

2.2 Format and Duration . . . 6

2.3 Handling `silence` and `unknown` Classes . . . 6

3 Model Architectures **7**

3.1 Transformer-based Model . . . 7

3.1.1 Wav2Vec 2.0-Based Classification . . . 7

3.1.2 HuBERT-Based Classification . . . 7

3.2 Custom model . . . 7

4 Methodology **8**

4.1 Parameter impact verification . . . 8

4.2 Reproducability . . . 8

5 Hyperparameters **8**

5.1 Training-related . . . 9

5.1.1 Batch size . . . 9

5.1.2 Learning rate . . . 9

5.1.3 Optimizer . . . 10

6 Architecture-specific parameters **10**

6.1 LSTM and RNN . . . 10

6.2 Custom architecture . . . 11

6.2.1 Number of attention heads . . . 11

6.2.2 Hop length . . . 12

7 Class imbalance **13**

7.1 Model evaluation . . . 13

7.2 Loss weights . . . 13

7.3 Undersampling . . . 15

8 Metrics **15**

9 Conclusion and Future Work **16**

1 Introduction

Transformers have revolutionized the field of deep learning, particularly in sequence processing tasks, by leveraging attention mechanisms to overcome the limitations of traditional recurrent and convolutional architectures. This work explores the application of transformer-based models in speech command classification, comparing their performance with existing approaches such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

The study focuses on two prominent transformer architectures—Wav2Vec 2.0 and HuBERT—for feature extraction, combined with downstream classifiers like MLPs, RNNs, and LSTMs. Additionally, a custom transformer model is introduced, which processes audio data through Mel spectrograms and patch-based attention. Key hyperparameters, including batch size, learning rate, and optimizer choice, are analyzed to understand their impact on model performance. The work also addresses challenges such as class imbalance and evaluates metrics like accuracy, precision, and recall.

1.1 Theoretical Background

The Transformer architecture represents a fundamental shift in the design of neural networks for sequence modeling and transduction tasks [6]. By exclusively utilizing attention mechanisms and dispensing with recurrence and convolution, it overcomes the limitations related to sequential computation and gradient propagation inherent to earlier architectures.

1.1.1 Core Architecture

The Transformer adheres to an encoder-decoder paradigm, designed to map an input sequence $(x_1, x_2, \ldots, x_n)$ to an output sequence $(y_1, y_2, \ldots, y_m)$:

- The **encoder** comprises a stack of identical layers, each containing a multi-head self-attention mechanism followed by a position-wise fully connected feed-forward network. It transforms the input sequence into a sequence of continuous representations.

- The **decoder** is similarly structured, with the addition of encoder-decoder attention layers that allow each position in the decoder to attend over all positions in the input sequence.

Formally, given an input matrix $X \in \mathbb{R}^{n \times d_{\text{model}}}$, the encoder computes a sequence of hidden states $H = (h_1, h_2, \ldots, h_n)$ through stacked transformations.

1.1.2 Key Innovations

Three central innovations define the Transformer's capabilities:

- **Attention Mechanism**
  The scaled dot-product attention computes a weighted sum of values V based on the similarity between queries Q and keys K, formulated as:

  $$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V$$

  This mechanism allows the model to directly attend to relevant information at different positions, enabling efficient modeling of global dependencies.

- **Multi-Head Attention**
  Instead of performing a single attention function with queries, keys, and values, multi-head attention projects the inputs into multiple subspaces and performs attention in parallel:

  $$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

  where each head is computed as $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$, capturing diverse linguistic features.

- **Positional Encoding**
  Since the model itself is agnostic to the sequence order, positional encodings are added to the input embeddings. The original work proposes sinusoidal encodings:

  $$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \quad \text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

  Alternatively, positional embeddings can be learned during training.

1.1.3 Advantages

The Transformer architecture yields several notable advantages over previous approaches:

- **Parallelization:** Unlike RNNs, the absence of sequential dependencies enables full parallelization during training, significantly improving computational efficiency.
- **Long-Range Dependencies:** Attention mechanisms can model dependencies between distant elements without degradation.
- **Scalability:** The architecture scales effectively with larger datasets and longer sequences.
- **Performance:** Transformers achieve state-of-the-art results across a range of tasks, from machine translation to text generation, often requiring fewer training iterations.

1.2 Existing approaches

1.2.1 Speech recognition based on Convolutional neural networks and MFCC algorithm

In this study, an automatic speech recognition (ASR) system was developed using a combination of Mel-Frequency Cepstral Coefficients (MFCCs) and Convolutional Neural Networks (CNNs). The MFCC algorithm was employed to extract distinctive features from speech signals, which were then used as input to a CNN classifier. This approach aimed to leverage MFCCs to reduce model complexity while preserving key audio characteristics, and to utilize CNNs for their ability to learn hierarchical representations from time-series data.

Three different CNN architectures were evaluated, varying in convolutional window sizes, number of layers, and dropout rates. The best performing model achieved an accuracy of **88.21%** on the test dataset. [4]

1.2.2 Key Word Spotting through Image Recognition

In this work, the speech recognition problem was addressed by converting audio data into log spectrogram images, thereby allowing the use of well-established convolutional neural network (CNN) architectures from the image domain. Several CNN models were implemented and tested, including a baseline CNN, a low-latency model optimized for memory efficiency, and a deeper CNN trained with Virtual Adversarial Training (VAT) for regularization.

The experiments showed that spectrograms provided more informative input than amplitude-time plots. Among the architectures, the deeper CNN with adversarial training achieved the highest validation accuracy of **92%**. This demonstrates that transforming the problem into the image domain and applying advanced regularization methods like VAT can lead to significant performance improvements in keyword spotting tasks. [2]

1.2.3 Speech Command Recognition in Computationally Constrained Environments with a Quadratic Self-organized Operational Layer

In this study, a novel method was proposed for speech command recognition (SCR) tailored for computationally constrained environments, such as embedded or robotic systems. Instead of compressing deep neural networks, the authors introduced a new network layer combining Self-Organized Operational Neural Networks (SelfONNs) and quadratic-form convolutional kernels.

This hybrid layer enriches the feature representation by integrating a Taylor-series-based expansion with quadratic interactions between input elements, enabling more expressive modeling capabilities. The method processes MFCC features extracted from 1-second audio clips and feeds them into a lightweight LeNet-1-inspired architecture where traditional convolutional layers are replaced with the proposed quadratic SelfONN layers.

Experiments conducted on the Google Speech Commands (GSC) and Synthetic Speech Commands (SSC) datasets showed that the proposed model achieves significant improvements in classification accuracy. Specifically, it reached **89.8%** accuracy on GSC and **97.9%** on SSC, outperforming both conventional CNNs and standard SelfONN layers.[5]

2 Dataset Description

The experiments are conducted on the dataset [7] which consists of over 65,000 labeled audio samples across multiple spoken command categories. The dataset is structured into 1-second mono-channel audio clips, sampled at 16 kHz, each containing a single spoken word.

2.1 Number of Samples and Classes

The core dataset includes 10 target command classes: `yes`, `no`, `up`, `down`, `left`, `right`, `on`, `off`, `stop`, and `go`. In addition, two auxiliary classes are introduced:

- `unknown` – captures all words not belonging to the target classes.
- `silence` – represents background noise or empty audio clips.

The total number of audio samples exceeds 65,000, with a varying number of samples per class. The dataset is split into training, validation, and test subsets according to official partitions provided in `validation_list.txt` and `testing_list.txt`.

2.2 Format and Duration

All audio samples are 16-bit PCM encoded WAV files with a fixed duration of 1 second. This consistency allows straightforward preprocessing and batching during model training.

2.3 Handling `silence` and `unknown` Classes

To improve model robustness, two special classes are handled explicitly:

- `silence` samples are generated from background noise files provided in the `_background_noise_` folder. These are segmented into 1-second clips and labeled accordingly.
- `unknown` samples are selected from words outside the core command set.

3 Model Architectures

3.1 Transformer-based Model

3.1.1 Wav2Vec 2.0-Based Classification

In this approach, we utilize the pre-trained **Wav2Vec 2.0** model to extract meaningful representations from raw audio waveforms. The input audio files are processed using the Wav2Vec 2.0 encoder to obtain high-level feature embeddings of size 768. These embeddings are then used as inputs to various downstream classification models. We evaluate the performance of the following architectures:

- **MLP**: A simple feed-forward network consisting of a linear layer, ReLU activation, dropout, and a final output layer.
- **RNN**: A recurrent neural network with tanh non-linearity and optional bidirectionality, used to capture sequential dependencies in the extracted features.
- **LSTM**: A long short-term memory network designed to capture longer temporal dependencies, optionally bidirectional.

We test both direct classification using the average-pooled embeddings as well as sequence-based classification using the full sequence of embeddings output by Wav2Vec 2.0.

3.1.2 HuBERT-Based Classification

The second approach follows the same pipeline but replaces Wav2Vec 2.0 with the **HuBERT** model for feature extraction. Similar to the Wav2Vec 2.0 setup, the HuBERT encoder outputs either average-pooled embeddings or full sequences of features, which are then used as inputs to the same classification architectures: MLP, RNN, and LSTM. This allows us to compare the effectiveness of Wav2Vec 2.0 and HuBERT representations in downstream speech classification tasks.

In both approaches, the pre-trained feature extractor is kept frozen during training, and only the classification head is trained. The models are trained and evaluated on a dataset of spoken commands, and performance is compared across the different architectural choices.

3.2 Custom model

The model follows a modular pipeline:

1. **Feature Extraction:** Raw audio waveforms are converted into mel-spectrogram representations using a MelSpectrogram transform followed by an amplitude-to-decibel conversion, capturing perceptually meaningful frequency features.
2. **Patch Embedding:** The spectrogram is partitioned into non-overlapping patches of fixed size (p_t, p_f) along the time and frequency dimensions. Each patch, reshaped into a vector, is projected into a high-dimensional embedding space via a linear transformation.

3. **Transformer Encoding:** After appending a learnable classification token and adding positional embeddings, the sequence of patch embeddings is processed by a multi-layer Transformer encoder that models contextual relationships across patches.

4. **Classification Head:** The output corresponding to the classification token is extracted and passed through a lightweight multilayer perceptron (MLP) head to predict class logits.

The architecture introduces several crucial adaptations:

- **Patch-Based Attention**
  Instead of attending over individual time steps or frequency bins, the model processes patches, significantly reducing sequence length and computational burden compared to treating each spectrogram pixel individually.
- **Learnable CLS Token and Positional Embeddings**
  A special learnable token is prepended to the patch sequence to aggregate global information necessary for classification. Fixed-size positional embeddings are added to maintain the order and structure of the patches.
- **Efficient Transformer Encoding**
  The model employs Transformer encoder layers with multi-head self-attention, allowing for parallel and efficient computation while capturing complex temporal-frequency interactions.

4 Methodology

4.1 Parameter impact verification

For each hyperparameter mentioned in section 5, the MLP classifier was the head of the HuBERT and Wav2Vec transformers architectures. In this case, an average pooling of the features across the time dimension was implemented in order to flatten the output of the pretrained models. RNN and LSTM have their architecrue-specific parameters verified in section 6 architecture comparison are introduced in Custom architecture mentioned throughout the entirety of the report is trained from scratch.
The primary metric of models' comparison is accuracy. Despite evident class imbalance, since all models are evaluated under identical conditions, accuracy differences still reflect relative model performance on the task at hand

4.2 Reproducability

Structured methodology had to be implemented in order to ensure reliable and reproducible results. Each model was trained three times for every verified hyperparameter value and the results were then aggregated using the mean and standard deviation of the final performance metrics. Every passage of training process was set by setting random seed of every aspect where randomness was involved, this includes the order of images passed in the training process, results of operations involving GPU acceleration and weight initialization of the networks themselves.

5 Hyperparameters

Hyperparameters play a critical role in determining the performance of a machine learning model. These external configurations, which are not learned during training, significantly influence the learning process. In this section, the hyperparameters used in the project are discussed.

5.1 Training-related

Training-related hyperparameters are crucial for controlling the learning process and ensuring that the model converges to an optimal solution. These include the learning rate, batch size, and the choice of optimizer. Each of these parameters affects the speed, stability, and final performance of the model.

5.1.1 Batch size

Table 1: Different batch sizes and the mean accuracy for each architecture

Architecture	**batch size=8**	**batch size=16**	**batch size=32**	**batch size=64**
Wav2Vec-based	0.9151±0.0092	0.9123±0.0156	0.8895±0.0192	0.8651±0.0209
HuBERT-based	0.9736±0.0231	0.9749±0.0129	0.9737±0.0233	0.9701±0.0136
Custom	0.8661±0.0103	0.8657±0.0144	0.8251±0.0168	0.8077±0.0211

Unlike the Wav2Vec-based and Custom models, the HuBERT-based architecture maintains high performance across batch sizes, with a peak at batch size equal 16. This suggests a more stable optimization landscape or improved pretraining representations. The minor accuracy drop at batch size 64 still retains over 97% accuracy, hinting at high capacity and robustness to gradient estimation noise. For both the Wav2Vec-based and Custom models, standard deviations tend to increase with batch size, suggesting less consistent convergence behavior. This is particularly notable in the Custom model at batch size 16, which may indicate susceptibility to poor minima or suboptimal learning dynamics due to reduced update frequency and less gradient noise

5.1.2 Learning rate

The learning rate determines the size of the steps taken during gradient descent to update the model's weights. A learning rate that is too high can cause the model to overshoot the optimal solution, while a learning rate that is too low can result in slow convergence or getting stuck in local minima. In the project, learning rates of `0.01`, `0.05`, `0.001`, `0.005`, `0.0001` and `0.0005` were examined. The results for each value of learning rate are presented in the Table 2.

Table 2: Different values of learning rates and the mean accuracy for each architecture

Architecture	**lr=0.05**	**lr=0.01**	**lr=0.005**	**lr=0.001**	**lr=0.0005**	**lr=0.0001**
Wav2Vec-based	0.6196±0.0000	0.7722±0.0124	0.8910±0.0186	0.9199±0.0202	0.9273±0.0119	0.9064±0.0176
HuBERT-based	0.6197±0.0282	0.9540±0.0167	0.9605±0.0281	0.9766±0.0171	0.9740±0.0238	0.9736±0.0182
Custom	0.0361±0.0161	0.0411±0.0261	0.0672±0.0511	0.6236±0.0291	0.7761±0.0156	0.8657±0.0244

The Custom model—which presumably lacks pretraining – exhibits extreme sensitivity to learning rate. At high learning rates (e.g., 0.05, 0.01), the model fails to converge effectively (e.g., 0.0361 and 0.0411 accuracy, respectively), suggesting divergence or poor optimization. Only at lower learning rates (e.g., 0.001 and below) does its performance improve substantially. This sensitivity is consistent with findings that models trained from scratch require smaller learning rates to avoid overshooting gradients. Both Wav2Vec-based and HuBERT-based models demonstrate robust performance across a wider range of learning rates, owing to the fact that only the classification head is trained, while the powerful pretrained encoder remains frozen. Having that in mind, the value of accuracy for lr=0.05 and 0.01 being equal around 62% actually reflects models' divergence and its labeling of nearly all data points as *unknown* (see section 7 for further discussion).

5.1.3 Optimizer

The optimizer is responsible for updating the model's weights based on the computed gradients. Different optimizers have unique characteristics that can affect the training dynamics and final performance. In the project, three popular optimizers were experimented with: `Adam`, `AdamW`, and `SGD`.

Table 3: Results of using different optimizers for each architecture

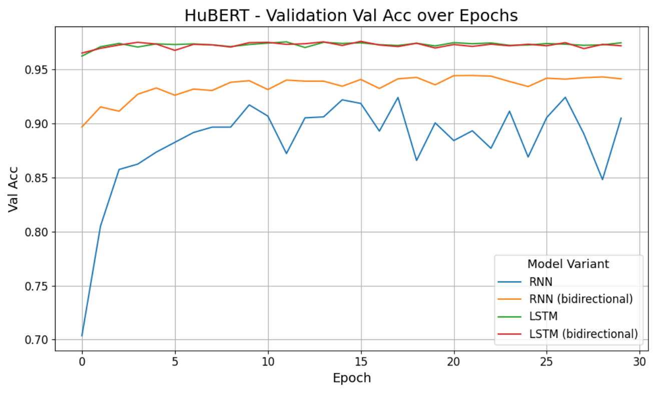
Architecture	**Adam**	**AdamW**	**SGD**
Wav2Vec-based	0.9127±0.0178	0.9161±0.0252	0.6196±0.000
HuBERT-based	0.9736±0.0162	0.9781±0.0209	0.6206±0.0161
Custom	0.8657±0.0244	0.8488±0.0188	0.6243±0.0154

For all architectures, both Adam and AdamW significantly outperform SGD by large margins. For instance, the HuBERT-based model achieves 0.97% accuracy with AdamW compared to just 62% with SGD. This disparity supports the findings of [8], which highlight that SGD requires careful tuning and is more sensitive to the loss surface than adaptive optimizers. These estimates allow for more stable and faster convergence, particularly in complex or poorly conditioned loss surfaces like those of Wav2Vec or HuBERT. For Wav2Vec-based and HuBERT-based models, AdamW yields higher accuracy than Adam – suggesting the benefit of explicit decoupled weight decay regularization.

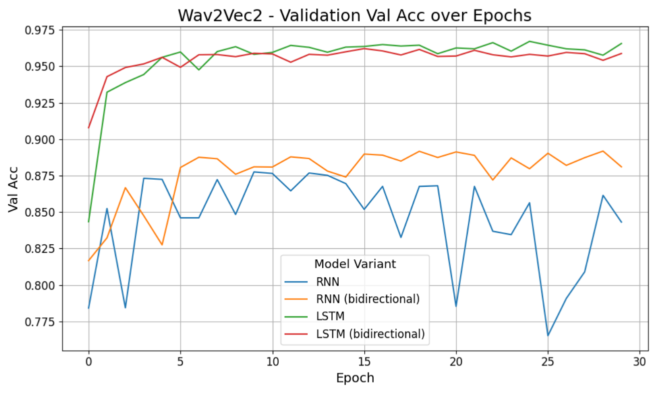
6 Architecture-specific parameters

6.1 LSTM and RNN

For both of LSTM and RNN architectures, the bididerctionality and its absence was verified in terms of its impact on the models' performances.
In a regular RNN or LSTM, the model reads the input sequence from start to end, learning only from past information. A bidirectional RNN or LSTM, on the other hand, processes the sequence in both directions – forward and backward – so it can learn from both past and future context at each step. This is especially useful in tasks like language understanding or speech recognition, where knowing what comes next can improve predictions. The bidirectional setting simply means the model has two layers working in opposite directions, and their outputs are combined to form a richer representation.



(a) HuBERT based architecture



(b) Wav2Vec based architecture

Figure 1: Training process for different architectures and bidirectional flag

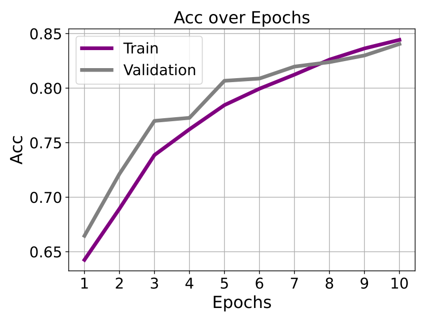
In both Wav2Vec2 and HuBERT, bidirectional RNN and LSTM consistently outperformed their unidirectional counterparts across all epochs which can be seen at Figure 1. This demonstrates that access to both past and future context significantly enhances model performance. LSTMs generally exhibited higher validation accuracy than RNNs in both bidirectional and unidirectional settings. This aligns with expectations, as LSTMs are better at capturing long-range dependencies due to their gating mechanisms. The gap between LSTM and RNN was more pronounced in HuBERT, suggesting that the choice of base architecture (HuBERT vs. Wav2Vec2) may influence the relative benefits of LSTM over RNN. Bidirectional models showed smoother convergence curves compared to unidirectional ones, particularly in the later epochs. This indicates that bidirectionality not only improves accuracy but also contributes to more stable training dynamics, likely by providing a more robust gradient signal through the additional context. While the trends were consistent across Wav2Vec2 and HuBERT, the absolute performance differed. Wav2Vec2-based models achieved higher overall validation accuracy (e.g., 0.975 for bidirectional LSTM) compared to HuBERT-based models (0.90 for the same architecture). This suggests that Wav2Vec2's pre-training may be more conducive to fine-tuning with bidirectional recurrent layers for the specific task evaluated.

6.2 Custom architecture

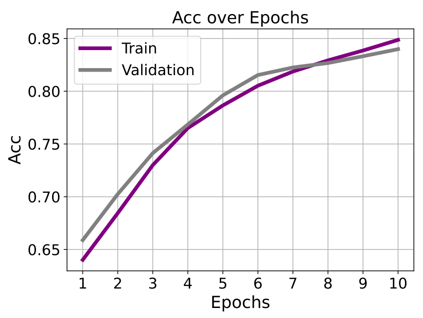
6.2.1 Number of attention heads

In the custom architecture provided, `num_heads` refers to the number of attention heads in the multi-head self-attention mechanism of the Transformer. Multi-head attention allows the model to attend to information from different representation subspaces at different positions, capturing a richer set of dependencies across the input sequence. Each head independently performs scaled dot-product attention, and their outputs are concatenated and linearly transformed. For example, with 4 heads, the model splits the embedding dimension into 4 parts (e.g., 128 $\rightarrow$ 4 heads $\times$ 32 each), enabling parallel attention over diverse patterns in the spectrogram patches, which enhances the model's ability to learn temporal and frequency-related features in the audio data.

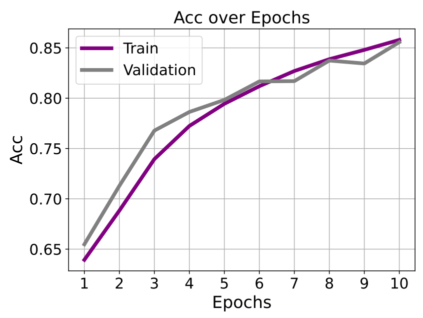
The minimal impact of varying the number of attention heads on the training process in audio classification using transformers can be attributed to several factors. Lower numbers of attention heads may already provide sufficient capacity to capture relevant patterns in the audio features. Additionally, in relatively simple or small-scale datasets (such us the one analyzed in the report), increasing the number of heads can introduce redundant computations without improving performance. The short training duration (10 epochs) may also limit the opportunity to observe any nuanced benefits from added model complexity. As a result, changes in the number of attention heads lead to



(a) 2 attention heads



(b) 4 attention heads



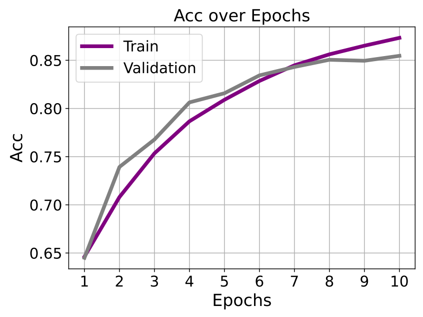
(c) 8 attention heads

Figure 2: Training process of custom architecture for different attention heads of the model

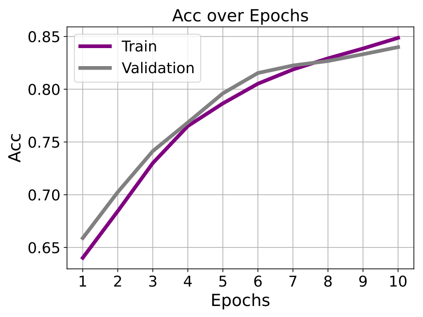
only marginal differences in training and validation accuracy. In such circumstances, the choice of lower amount of attention heads would be advised, as it significantly reduces the computational overhead.

6.2.2 Hop length

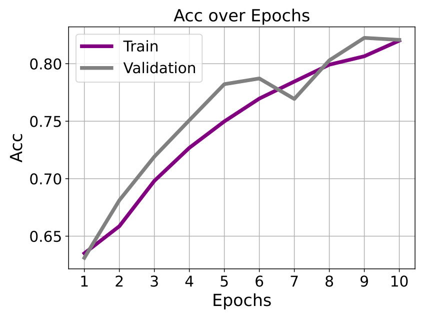
In the custom architecture, the `hop_length` parameter determines the stride or step size between consecutive frames when computing the Mel spectrogram. It controls how much the analysis window shifts over the audio signal, influencing the temporal resolution of the resulting spectrogram. A smaller hop length results in more overlap between frames, capturing finer temporal details but increasing the number of frames (and thus computational cost). Conversely, a larger hop length reduces overlap, leading to a coarser temporal representation. For example, With a hop length of 512 at a 16kHz sample rate, each frame represents approximately 32 milliseconds of audio (512 / 16000), balancing temporal resolution with efficiency for 1-second audio inputs. The remaining hop lengths of 253 and 1020 are due to model's architecture and division of 16000-long waveform array (closest to power-of-two equivalents that allow architecture's structure to remain the same).



(a) hop length = 253



(b) hop length = 512



(c) hop length = 1020

Figure 3: Training process of custom architecture for different hop length parameter values

Lower hop lengths lead to richer, more temporally precise data representations, allowing the model to learn better and generalize more effectively. However, this comes with increased computational cost and highest observable difference between training and validation performance. Conversely, larger hop lengths reduce input complexity but risk losing crucial short-term patterns, leading to reduced accuracy. The extended hop length may intuitively be more detrimental to the model's performance especially due to the nature of short (1-second) audio files, potentially losing crucial information required for classes' distinction. This phenomenon can be observed in relatively unstable learning process for the highest analyzed hop length value from validation set point of view.

7 Class imbalance

Due to the fact, that the original dataset was split into 32 different classes, the 12–fold classification task involves severely imbalanced distribution of the dataset, as shown in Table 4. Label distribution in validation and test dataset is exactly proportional. Thus far, accuracy was the metric of choice for models' comparison. This approach

Table 4: Label distribution in the training set

Label	**Count**
yes	1860
no	1853
up	1843
down	1842
left	1839
right	1852
on	1864
off	1839
stop	1885
go	1861
silence	292
unknown	32556

has its positive and negative interpretation. On one hand, the most represented class in the training dataset is also the most common one in the test data, making it important for models to be able to recognise it correctly. That ability is well-captured by accuracy as it is susceptible to the models' performance on the dominant classes. On the other hand, the accuracy metric does not reflect the model's ability to predict other scarcely-represented classes. That is why, as accuracy may be used to compare models' performance between one another, different model validation techniques shall be investigated in this chapter, allowing absolute, not comparative evaluation of the models.

7.1 Model evaluation

An important value of accuracy that appears often in the evaluation of hyperparameters' impact is around 62% (see Wav2Vec and HuBERT at lr=0.05 and 0.01 in Table 2, SGD in Table 3). This value is roughly the fraction of the most dominant class (*unknown*) in the test dataset. In particular, the exact value of acc=0.6196 denotes this exact situation, as shown on Figure 4. In order to tackle this problem several approaches can be implemented, few of which will be introduced in following sections.

7.2 Loss weights

A natural approach of dealing with class imbalance is to set appropriate weights to each label, that would correspond to its proportion in training dataset's representation [1]. Investigated loss weights were in a form of a vector derived

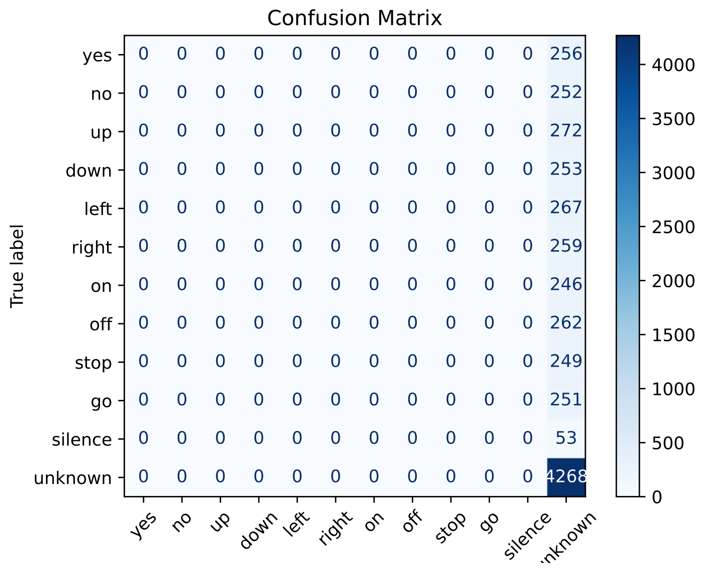


Figure 4: Confusion matrix of all models with denoted accuracy of 0.6196

according to the equation:

$$\left(\frac{\text{median}(C)}{c_1}, \frac{\text{median}(C)}{c_2}, \ldots, \frac{\text{median}(C)}{c_n} \right)$$

The experiments below were conducted on the the Wav2Vec architecture core with lr=0.05 and lr=0.0001 in order to capture its impact on Table 5 presents the accuracy and recall scores for Wav2Vec models trained with different

Table 5: Accuracy and F1 scores for different architectures

	Unweighted loss		Weighted loss	
Architecture	Accuracy	Recall	Accuracy	Recall
Wav2Vec+lr=0.0001	**0.9064**	0.8055	0.8097	**0.9045**
Wav2Vec+lr=0.05	**0.6196**	0.0638	0.0357	**0.0833**

learning rates, both with and without loss weighting. As seen, the model trained with a low learning rate of 0.0001 achieves high accuracy (0.9064) and recall (0.8055) under unweighted loss, and similarly strong performance under weighted loss, although with a slight drop in accuracy (0.8097) but a notable improvement in recall (0.9045). In contrast, the model trained with a high learning rate of 0.05 suffers from a drastic decline in performance, achieving only 0.6196 accuracy and 0.0638 recall without weighting, and even worse results (0.0357 accuracy) under weighted loss, despite a modest improvement in recall (0.0833). The phenomenon behind the 62% accuracy without weighting occurs because the "unknown" class constitutes the majority of the dataset, skewing the model's predictions when trained under suboptimal hyperparameters like an excessively large learning rate. Introducing weighted loss partially addresses class imbalance by penalizing errors on minority classes more heavily, thus increasing recall for these classes, although at the cost of overall accuracy. These results highlight the importance of proper learning rate selection and loss rebalancing when working with highly imbalanced datasets.

7.3 Undersampling

A cost-efficient and common approach would be to undersample the most represented class [3]. The impact of the severity of undersampling was verified for four different values p, that represent the fraction of the *unknown* class occurances was used in the training process. The model chosen for the verification was best performing configuration of Wav2Vec architecture from Table 2. The table shows how the performance of the Wav2Vec+MLP model changes with different undersampling coefficients p. As p increases, there is a clear trend of improvement in accuracy, precision, and F1 score, suggesting that reducing the degree of undersampling helps the model make more reliable and balanced predictions. However, recall behaves differently: it is initially very high at low p values but gradually declines as p increases. This indicates that while the model becomes more precise and overall better at correct classifications with less undersampling, it sacrifices some ability to capture all positive cases. Overall, higher values of p seem to yield better general performance, but with a trade-off between accuracy and recall that should be carefully considered depending on the application's priorities.

Table 6: Wav2Vec+MLP model results for different undersampling coefficient p

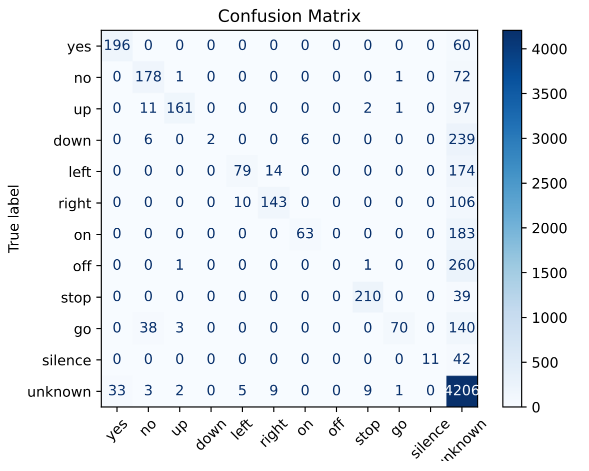
p	**Accuracy**	**Precision**	**F1 score**	**Recall**
0.05	0.7796±0.0102	0.7198±0.0155	0.7864±0.0128	0.8966±0.0094
0.1	0.8651±0.0091	0.7749±0.0137	0.8341±0.0115	0.9033±0.0082
0.2	0.8997±0.0088	0.7937±0.0121	0.8440±0.0104	0.9120±0.0076
0.5	0.9162±0.0074	0.9031±0.0093	0.8832±0.0087	0.8695±0.0101
1	0.9273±0.0067	0.9231±0.0081	0.8959±0.0079	0.8738±0.0090

8 Metrics

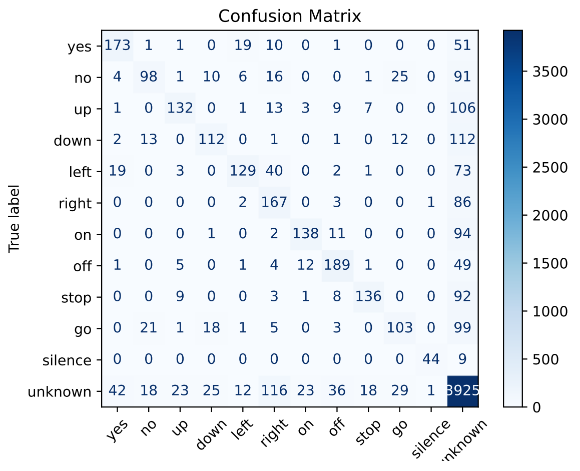
While accuracy is a commonly reported metric for model performance, the following comparison of two models demonstrates why relying solely on accuracy can be misleading and why additional metrics such as F1-score, precision, recall, and confusion matrices should be considered:

Architecture	**Accuracy**	**Recall**	**F1 score**	**Precision**
Wav2Vec-based architecture lr=0.01	0.7761	0.5924	0.6415	0.7253
Custom architecture lr=0.0005	0.7722	0.4576	0.5262	0.8197

Both models achieve nearly identical accuracy scores (Model 1: 0.7761, Model 2: 0.7722 Table **??**), which might suggest comparable performance. However, a deeper analysis reveals significant differences in their behavior. Model 1 exhibits a substantially higher F1-score (0.6415 vs. 0.5262) and recall (0.5924 vs. 0.4576), indicating better balance between precision and recall.



(a) Confusion matrix for Model 1



(b) Confusion matrix for Model 2

Figure 5: Confusion matrix for both models

- **Reliability:** Model 1 provides more consistent performance, while Model 2 has severe but isolated weaknesses
- **Error Distribution:** Model 1's errors are spread across similar classes, whereas Model 2's errors are concentrated in specific failure modes
- **Unknown Class Handling:** Model 2's excessive use of "unknown" classification artificially preserves accuracy while masking detection failures

This comparison demonstrates why accuracy alone can be misleading. Model 1's consistent performance across all classes makes it preferable for most applications, despite Model 2's excellence in specific categories. The analysis underscores the importance of examining complete confusion matrices and secondary metrics (especially recall and F1-score) when evaluating model performance for real-world use cases.

9 Conclusion and Future Work

This study explored the application of transformer-based architectures for speech command classification, comparing their performance against traditional approaches and analyzing the impact of key hyperparameters. The experiments demonstrated that pretrained transformer models like Wav2Vec and HuBERT, combined with simple downstream classifiers (MLP, RNN, LSTM), achieved superior performance—with HuBERT-based models reaching up to 97.8% accuracy—compared to custom transformer architectures trained from scratch. The robustness of pretrained models was evident in their stability across varying batch sizes and learning rates, while the custom model required careful hyperparameter tuning to avoid divergence.

Class imbalance emerged as a critical challenge, with the dominant *unknown* class skewing accuracy metrics. Techniques like loss weighting and undersampling improved recall for minority classes but highlighted trade-offs in overall accuracy. The analysis of bidirectional RNN/LSTM architectures confirmed their advantage in capturing temporal dependencies, while experiments with attention heads and hop lengths in the custom transformer revealed the importance of balancing computational efficiency with feature resolution.

Several directions merit further investigation:

- **Hybrid Architectures**: Combining the strengths of pretrained transformers (e.g., HuBERT) with lightweight recurrent or convolutional layers could enhance efficiency for edge devices while preserving accuracy.
- **Dynamic Class Balancing**: Techniques like curriculum learning or adaptive sampling could mitigate class imbalance without sacrificing minority class performance.
- **Attention Mechanisms**: Exploring sparse or memory-efficient attention variants (e.g., Longformer, Performer) may reduce the computational overhead of the custom transformer while maintaining performance.
- **Multilingual Adaptation**: Extending the framework to non-English speech commands, leveraging multilingual pretrained models like XLS-R.

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015. URL: `http://arxiv.org/abs/1511.00561`, `arXiv:1511.00561`.

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

[3] Haibo He and Edwardo A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009. `doi:10.1109/TKDE.2008.239`.

[4] Arzo Mahmood and Utku Köse. Speech recognition based on convolutional neural networks and mfcc algorithm. *Advances in Artificial Intelligence Research*, 1(1):6–12, 2021.

[5] Mohammad Soltanian, Junaid Malik, Jenni Raitoharju, Alexandros Iosifidis, Serkan Kiranyaz, and Moncef Gabbouj. Speech command recognition in computationally constrained environments with a quadratic self-organized operational layer. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2021.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[7] Pete Warden. Speech commands: A public dataset for single-word speech recognition. *Dataset available from* $http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz$, 2017.

[8] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL: `https://proceedings.neurips.cc/paper_files/paper/2017/file/81b3833e2504647f9d794f7d7b9bf341-Paper.pdf`.