Deep Learning. Project 2
Transformers
Report

Michał Iwicki 320556
Michaił Legczylin 320640

April 2025

Contents

1 Introduction

In this project, we investigated word classification based on one-second sound files using several different deep learning models. We also aimed to identify the best model and its configuration for this specific task.

We implemented two transformer architectures — `Mel_transformer` for the mel spectrogram representation of sound data and `RawAudioTransformer` for the time-amplitude representation — and also utilized a pretrained `EfficientNet` model to assess the performance of our custom models.

2 Types of used neural networks

2.1 CNN

CNN (Convolutional Neural Network) is a type of deep learning (DL) model primarily used for image classification. It extracts features by applying convolutional layers, batch normalizations, and activation functions (most commonly `ReLU`), repeating these steps several times. After many transformations, it outputs a numerical vector corresponding to the probabilities of how confident the model is that a specific image belongs to a given category.

2.2 Transformers

Originally, transformers were used in natural language recognition and translation. They consist of two parts — encoders and decoders. Encoders convert words into tokens — numerical vector representations — allowing the model to analyze the importance of specific words. These tokens are then passed to decoders, processed again, and translated into another language via a word embedding table.

In our project, however, we used encoders only, processing and analyzing data already in numerical representation, giving us the ability to classify incoming sound data. An example transformer architecture: 1

Key components of transformers include:

- **Input embedding** — converts words into numerical vectors,
- **Positional encoding** — adds position information to these vectors,
- **Multi-head attention** — applies the multi-head attention algorithm to contextualize a given token within the scope of the context window with other unmasked tokens,
- **Output embedding** — converts output words into numerical vectors,
- **Masked multi-head attention** — applies the multi-head attention algorithm with masked tokens.

2.3 Sound data representations

2.3.1 Raw

Raw sound data is represented as amplitudes of sound in the time domain.

In our project, each audio file has $16,000$ samples, which means a vector of this length containing values corresponding to a combination of different amplitudes. Example Figure 2.

2.3.2 Mel Spectrogram

A mel spectrogram is created from sampled audio using the Fast Fourier Transform (FFT) algorithm and applying the mel scale to it. It is a three-dimensional object. In the following example (Figure 3), for every time domain value, we obtain the result of the FFT algorithm. The FFT algorithm is a numerical approximation used to calculate the Fourier Transform. The Fourier Transform itself is a brilliant mathematical achievement by the French mathematician Fourier, allowing functions to be converted from the time domain into the frequency domain. It enables the identification of frequencies with the highest amplitude, and thus, the greatest impact.

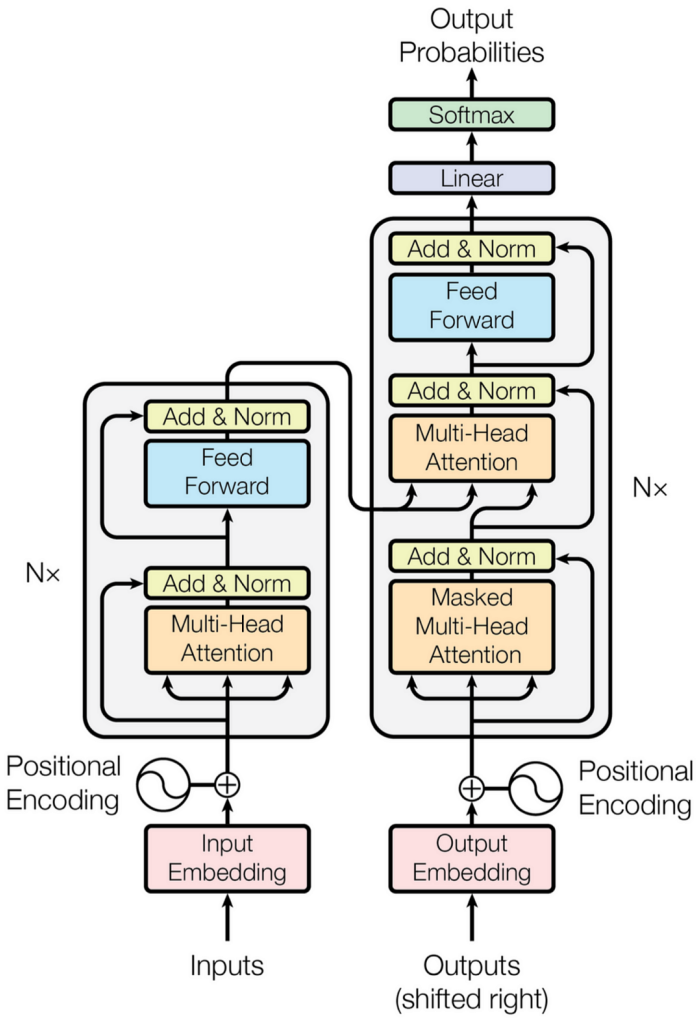

Figure 1: Transformer example (both encoders and decoders).

3 Used Architectures

In this project, we used three different architectures: two based on image processing using mel spectrograms scaled into decibel scales, and one operating directly on raw sequences of audio signals.

1. **Mel_transformer** — a transformer based on the mel spectrogram representation of sound data. First, it applies a 2D CNN model with two convolutional layers (kernel sizes equal to 3 and paddings equal to 1), batch normalization, and the `ReLU` activation function. This is followed by a standard linear layer connected to four transformer encoder layers, topped with an output layer. This architecture was inspired by AST architecture, but instead of splitting spectrogram into blocks we used 2 standard convolutional layers.

2. **RawAudioTransformer** — operates on a vector-based sound data representation, similar to `Mel_transformer`. It uses a 1D CNN model with two convolutional layers (kernel sizes equal to 16 and 8, strides equal to 4 and 2, paddings equal to 6 and 3 repectively), batch normalization, and the `ReLU` activation function. After the transformer encoder layer, the outputs are averaged along the time dimension and connected to the output layer. It is our implementation of idea of Wav2Vec, which uses 1D convolutional layers followed by transformer layers.

3. **EfficientNet** — a pretrained model available in the Python `torch` package. Among the available architectures, we used B0 because it is the baseline model with the only available

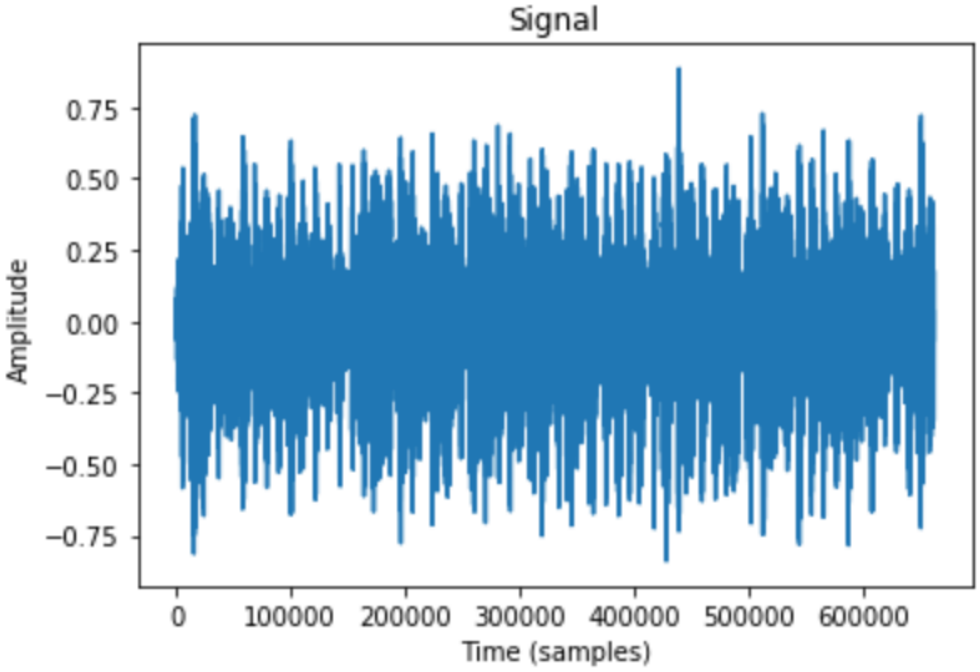

Figure 2: Raw format example.

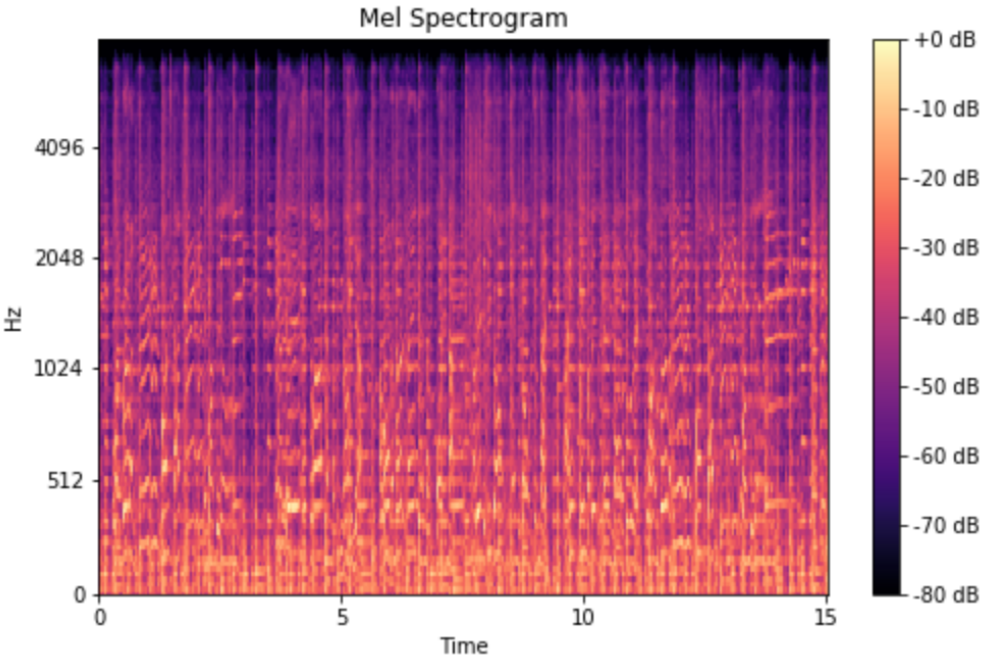

Figure 3: Mel Spectrogram format example.

pretrained weights option (`EfficientNet_B0_Weights.IMAGENET1K_V1`). We modified the first layer, as by default it expects a three-channel color image.

4 Preprocessing

According to the provided lists, we split the data into train, validation, and test subsets. Each audio file was read with `torchaudio` and either clipped or padded to exactly 16,000 samples. After

addressing potential differences in sample rates, the files were saved as tensors in two copies: raw audio and mel spectrogram in the decibel scale.

4.1 Denoising

We also experimented with a denoising method as part of the data preprocessing. The implementation we used was from the `noisereduce` package. Due to its compatibility with PyTorch and CUDA, we did not observe significant computational overhead. In the experiments, we tested both versions: with and without denoising.

5 Experiments and Results

5.1 Experimental Setup

- **Dataset**: All training and testing were conducted on the full dataset. The training set was used to fit the models, the validation set was employed for early stopping and for identifying the best-performing parameter state, and the test set was used to measure the model's final quality.
- **Training Configuration**: The training was conducted with the following setup:
  - **Epochs**: 15 epochs, with the patience parameter set to 3.
  - **Optimizer**: Adam optimizer with weight decay for model selection.
  - **Learning Rate**: default value of 0.001.
  - **Batch Size**: default value of 16.
  - **Denoising**: data processed with the default parameters of the `reduce_noise` function from the `noisereduce` package.

5.2 Testing Training Parameters

In this section, we tested the influence of learning rate and weight decay on models based on mel spectrograms. Due to high computational requirements, we abandoned these experiments for the raw audio transformer. For each model, we selected the largest learning rate at which the model successfully converged and tested two smaller values. For weight decay, we tested standard values of 0.01, 0.001, and 0.0001. We used data with denoising preprocessing, and each test was conducted three times.

5.2.1 Mel_transformer

The learning rate has minimal influence on the model's accuracy but can affect its stability. This architecture requires much smaller learning rates than CNNs and does not converge with a learning rate of 0.0005. Weight decay appears to have a negligible effect, although excessively large values can cause convergence issues. The results can be seen on figures 4 and 5.

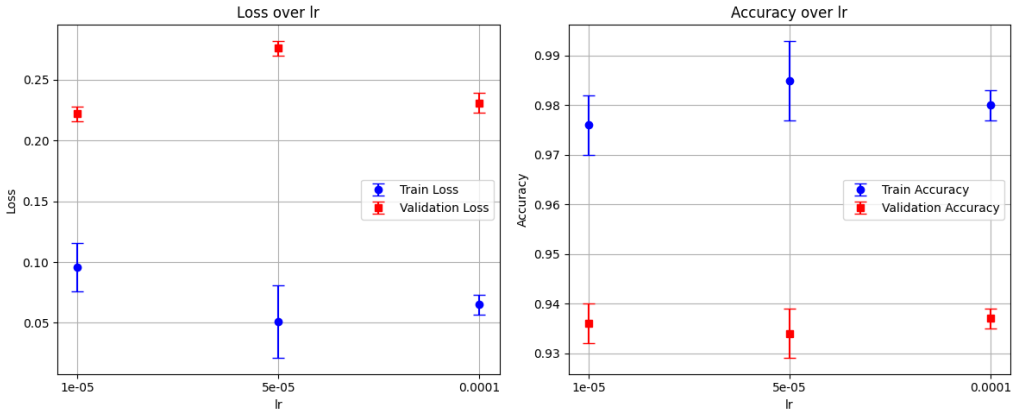

Figure 4: Metrics values vs. learning rate for Mel_transformer.

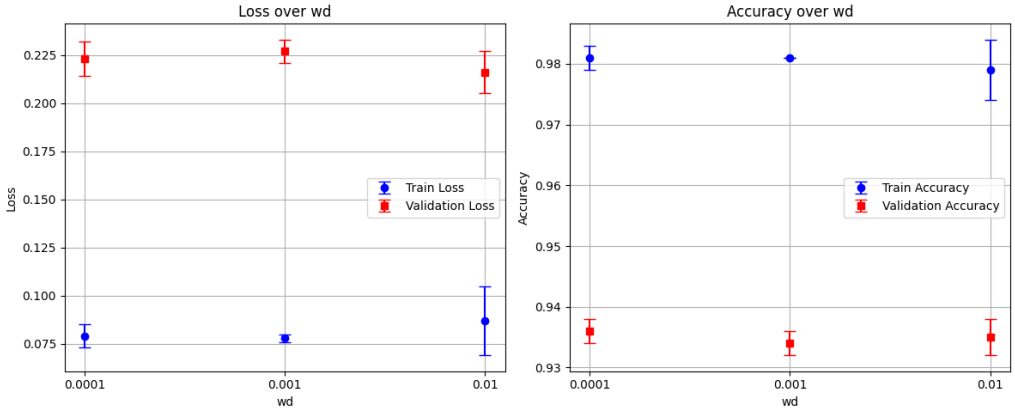

Figure 5: Metrics values vs. weights decay rate for Mel_transformer.

5.2.2 EfficientNet

Interestingly, the best parameters seem to be identical to those found in the previous project. A small learning rate can lead to underfitting, whereas a larger weight decay may help the model find a better solution compared to no regularization. However, this comes at the cost of increased model instability. The results can be seen on figures 6 and 7.

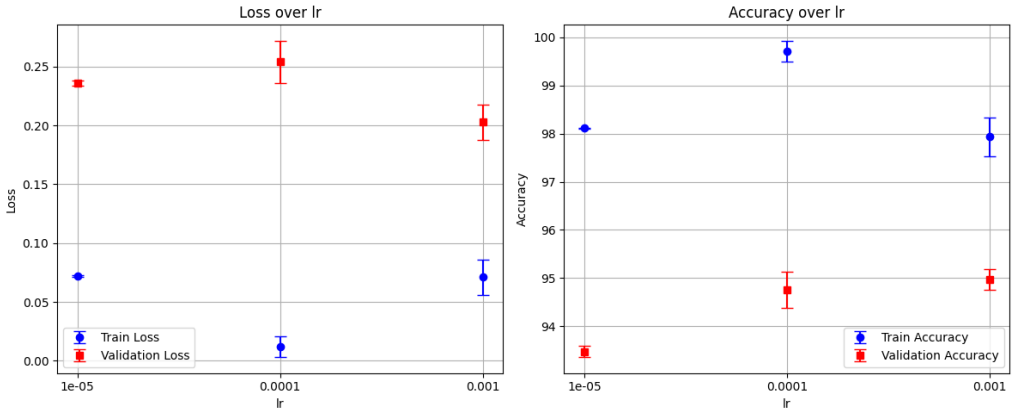

Figure 6: Metrics values vs. learning rate for EfficientNet.

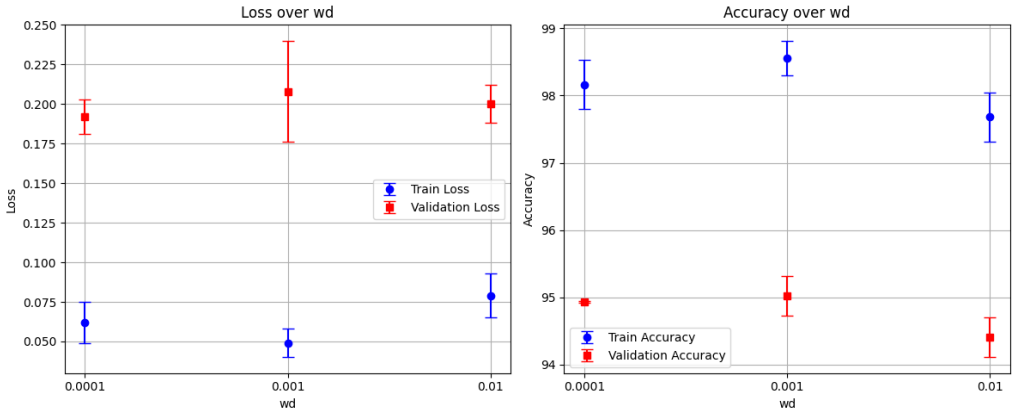

Figure 7: Metrics values vs. weights decay rate for EfficientNet.

5.3 Noise vs Voice Detection

The dataset provided a few recordings of noise. Unfortunately, after splitting these into segments of the same size as the main dataset, we obtained around 450 samples, which is relatively low even for single-class data. Therefore, we applied data augmentation techniques and increased the number of observations to 1751. These were split into training, validation, and test sets in proportions of 0.6, 0.2, and 0.2, respectively. For the voice samples, we randomly selected three times more observations from random classes of the main dataset.

Generation of New Noise Data

- Each recording was segmented into 16,000-length samples with three different uniform shifts.
- An additional 25% of the dataset was generated through random convex combinations of two existing samples.
- An additional 15% of the dataset was created by adding Gaussian noise to existing observations.

The transformer architecture based on mel spectrograms performed exceptionally well on this task, achieving over 99% accuracy across all datasets, as can be seen on figure 8.

```
Epoch 1: Train Loss: 0.0698 | Train Acc: 0.9860 | Val Acc: 0.9986

Epoch 2: Train Loss: 0.0192 | Train Acc: 0.9971 | Val Acc: 0.9986

Epoch 3: Train Loss: 0.0168 | Train Acc: 0.9964 | Val Acc: 0.9986

Epoch 4: Train Loss: 0.0096 | Train Acc: 0.9981 | Val Acc: 0.9979
Early stopping triggered at epoch 4
Best Val Acc: 0.9986
```

Figure 8: Training evaluation of noise vs. voice model.

5.4 Word Classification

The primary objective of this project was to predict the spoken word from a given audio file. Based on the results of the training parameter experiments, we selected optimal hyperparameters for each model, monitored the training process, and evaluated performance on the test dataset. We tested on both standard and denoised data in order to check if there is any advantage gained from denoising audio in terms of accuracy of converging.

5.4.1 Mel_transformer

Best accuracies achieved for Mel_transformer are 94.76 (for standard) and 94.22 (for denoised). Figure 9 represents change of loss and accuracy for training process of this model. Figure 10 presents typical training speed. And figure 11 shows confusion matrix, i.e. predicted vs. expected classes.

Figure 9: Training evaluation of Mel_transformer model.

Figure 10: Training speed for Mel_transformer model.

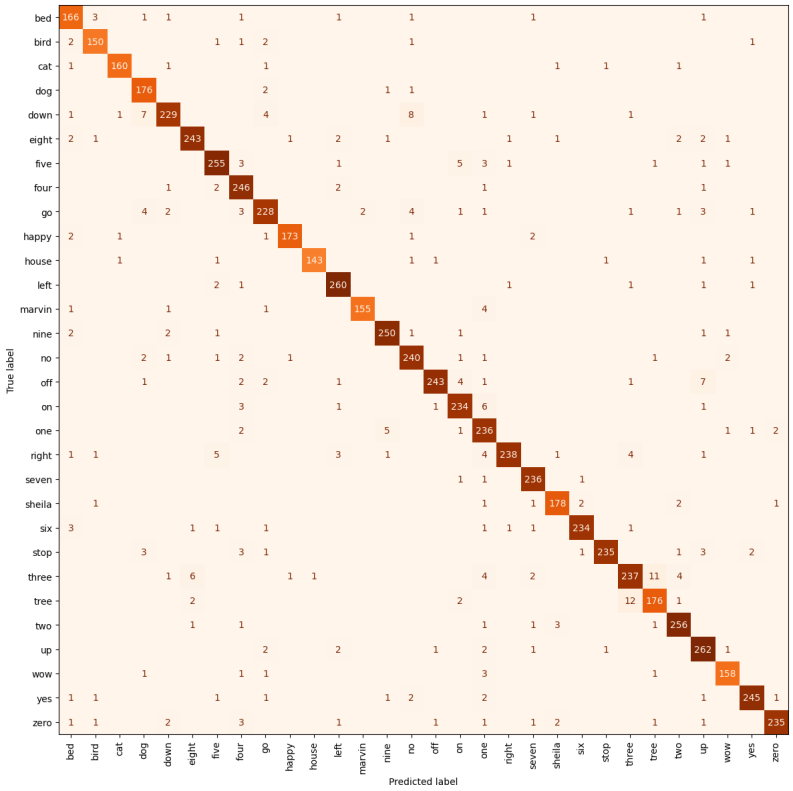

Figure 11: Confusion matrix of best performance of Mel_transformer model.

5.4.2 RawAudioTransformer

Best accuracies achieved for RawAudioTransformer are 83.64 (for standard) and 85.5 (for denoised). Figure 12 represents change of loss and accuracy for training process of this model. Figure 13 presents typical training speed. And figure 14 shows confusion matrix, i.e. predicted vs. expected classes.

Figure 12: Training evaluation of RawAudioTransformer model.

```
Epoch 3/20:   9%|▉          | 303/3193 [00:45<07:14,  6.65it/s, loss=1.07]
```

Figure 13: Training speed for RawAudioTransformer model.

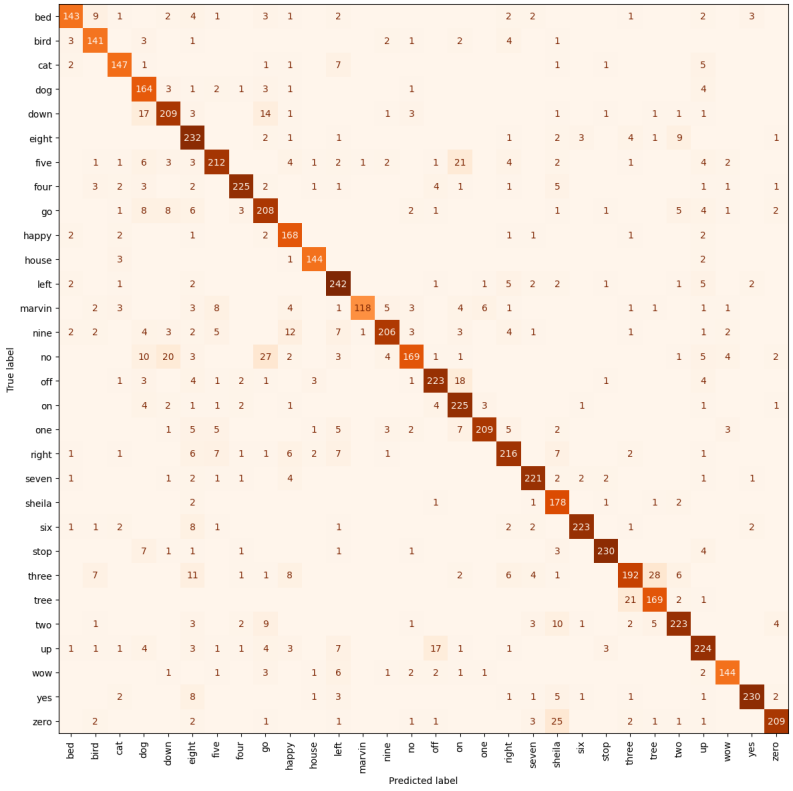

Figure 14: Confusion matrix of best performance of RawAudioTransformer model.

5.4.3 EfficientNet

Best accuracies achieved for EfficientNet are 95.82 (for standard) and 95.96 (for denoised). Figure 15 represents change of loss and accuracy for training process of this model. Figure 16 presents typical training speed. And figure 17 shows confusion matrix, i.e. predicted vs. expected classes.

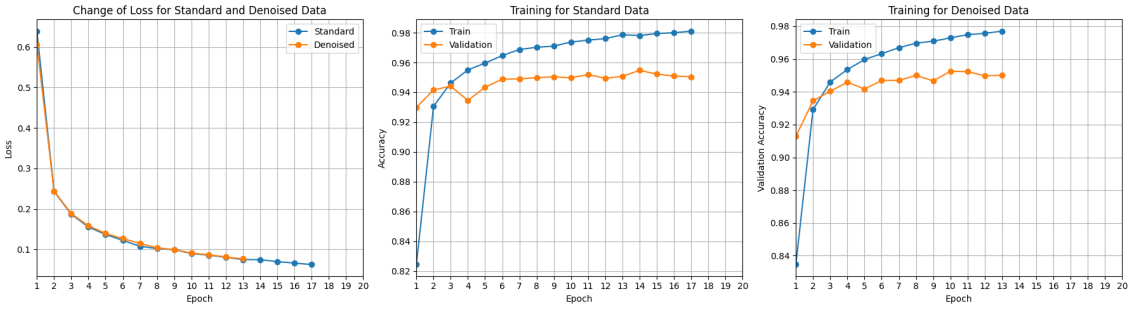

Figure 15: Training evaluation of EfficientNet model.

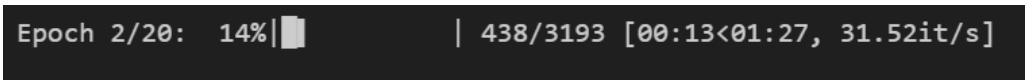

Figure 16: Training speed for EfficientNet model.

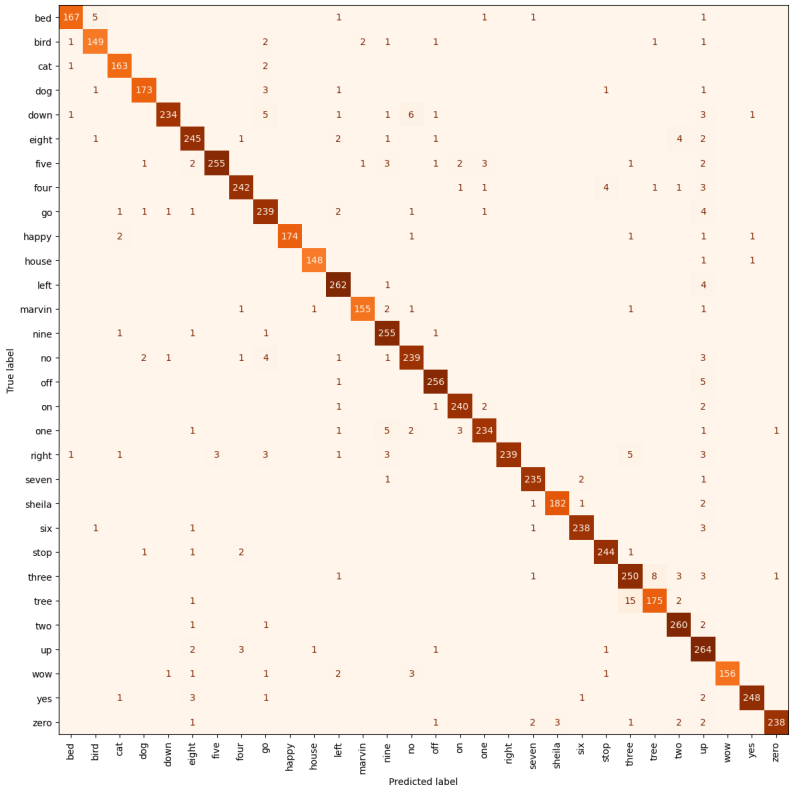

Figure 17: Confusion matrix of best performance of EfficientNet model.

5.4.4 Performance Discussion

Our testing platform consisted of an RTX 4070 Super GPU, a Ryzen 7500F processor, 32 GB of DDR5 RAM, and an SSD with read speeds of up to 7000 MB/s. It is a high-performance workstation capable of training one epoch of the mel spectrogram model in under one minute. However, it has a high power consumption, with the GPU alone drawing approximately 120 W, resulting in a total system consumption exceeding 150 W.

The models differed significantly in terms of computational speed. Disappointingly, the RawAudioTransformer trained approximately ten times slower than the spectrogram-based model, achieving fewer than 7 iterations per second, compared to around 67 iterations per second for the Mel

spectrogram model. On the other hand, EfficientNet was unable to fully utilize the GPU's potential, reaching approximately 32 iterations per second, with less than 50% GPU usage, whereas the transformers consistently maintained over 90% GPU utilization.

Interestingly, after loading a new dataset, during the first epoch the number of iterations per second was consistently limited to around 8, with reduced GPU usage. Only from the second epoch onwards was the GPU able to operate at full capacity. This behavior suggests that some form of caching or data preprocessing occurs during the initial run, which subsequently enables faster data loading into the GPU.

5.5 Conclusions

Our experiments demonstrated a significant advantage in using a pretrained model, which converged very rapidly, although the `Mel_transformer` also exhibited outstanding performance. The `RawAudioTransformer`, on the other hand, performed poorly both in terms of accuracy and computational efficiency. These results suggest that spectrograms represent human perception of sound much better than raw amplitude values.

The types of misclassifications observed in the spectral models were quite similar. All models confused the words *three* and *tree*, which is natural and may result from unclear pronunciation in the recordings. In the confusion matrices, vertical lines are visible, indicating that when the model is uncertain about a word, it tends to assign it to a few preferred labels associated with ambiguous observations.

Training on denoised data doesn't seem to bring significant advantages. However, we observe greater variance in accuracy on the validation dataset during training. This could be beneficial in the context of early stopping by providing a wider range of potential results. This is an interesting area for further research.

6 Final Summary

- The experiments demonstrate the superiority of using spectrograms compared to analyzing raw audio. Both the results and the computational efficiency were incomparably better.
- Voice recognition proved to be an easier problem when transformed into an image recognition task rather than working directly with raw audio files, likely because spectrograms better represent how humans perceive sound.
- Distinguishing voice from noise presented no significant challenge. More interesting research could be conducted by adding noise to the voice data as well and analyzing how the accuracy changes with different noise variances.
- Utilizing pretrained image classification models offers substantial advantages in both convergence speed and result quality. However, CNNs cannot fully exploit GPU potential compared to transformers.
- Transformers are less sensitive to training parameters, but this also means that there are fewer gains from extensive hyperparameter tuning.
- Denoising audio data can potentially accelerate convergence and help prevent overfitting, but further experiments with different problems and architectures are necessary to confirm this.

References

- Wikipedia on transformers in DL.
- Explanation of Fourier Transform, FFT and mel scale.
- Article "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks".
- Denoising data article.
- Description of AST inspiration for our mel spectogram transformer architecture.
- Article of Wav2Vec that inspired us for our raw audio tranformer.