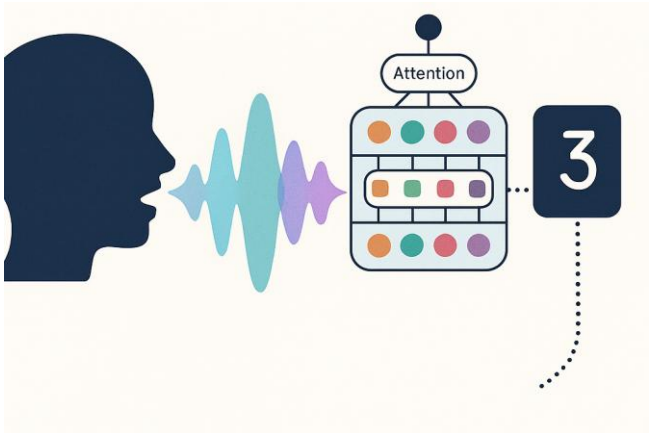

TRANSFORMER-BASED MODEL FOR SPOKEN DIGIT RECOGNITIO

Mahdi Mohammadi
Bigdata Processing

System Architecture

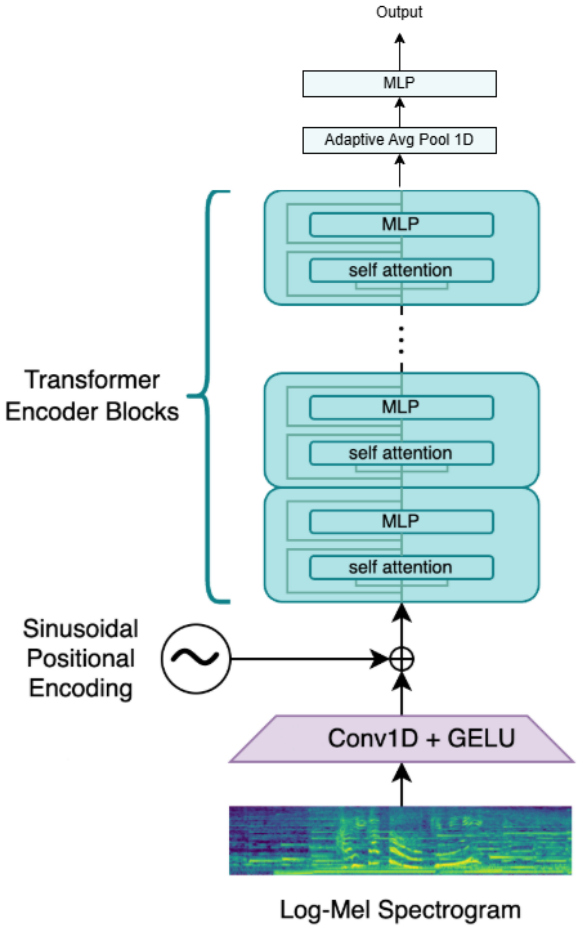

Figure 1 - Transformers Structure

The structure of this system is based on the Encoder section of the Whisper model. In fact, for the given task, it is sufficient to extract the features of each audio file and train a simple neural network to classify the output based on each numerical feature vector. The function of each component is briefly described below:

Model Input

The Mel Frequency Cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel frequency scale. Mel-frequency cepstral coefficients are commonly used as features in speech recognition systems.
The input dimensions are batch_size × 80 × time_steps, where 80 represents the number of filters used in the mel filter bank.

Convolution Layer + GELU

Conv1d is designed for processing sequential data (such as audio signals). Here, using Conv1d allows us to detect local patterns in the mel-spectrogram, such as frequency variations over short time intervals, and extract key features from the input signal. GELU is an activation function defined as:

$$\mathrm{GELU}(x) = x \cdot \Phi(x)$$

where Φ(x) is the cumulative distribution function of the standard normal distribution (Figure 2). This function determines how much of the input should be passed to the output. Unlike ReLU, which abruptly zeroes out negative values, GELU provides a smooth activation that continuously adjusts the output, leading to more stable gradients during backpropagation and helping to avoid issues like vanishing gradients. Additionally, by not discarding negative values, it allows important negative inputs to be preserved in the output.

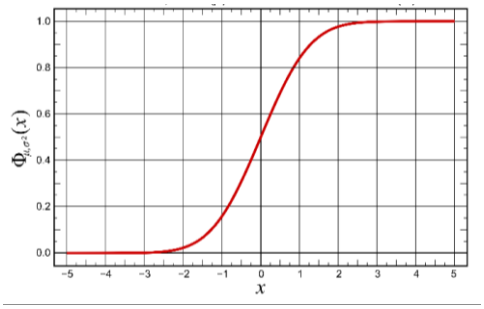

Figure2 - Standard Normal CDF

This layer increases the number of features to d_model (here, 128). The output of this layer has dimensions batch_size × d_model × time_steps.

Positional Encoding Layer

This component adds the position of time steps to the input features so that the model can understand the temporal order of the data. Sinusoidal and Cosinusoidal functions with different frequencies are used for positional encoding. The input to this component has dimensions batch_size × time_steps × d_model, and the output retains the same dimensions.

Transformer Encoder Blocks

This block includes:

- **Self-Attention** for modeling long-term dependencies between time steps
- **MLP (Multi-Layer Perceptron)** for learning complex feature combinations
- **Dropout** to prevent overfitting

The input and output dimensions of this section are batch_size × time_steps × d_model. After extensive experimentation, the final model uses 4 encoder blocks. Each block uses 8 attention heads (values 2 and 4 were also tested). For the MLP, after testing 64, 128, and 256, the final choice was 256 nodes in the hidden layer with ReLU activation (the input and output layers have d_model = 128 nodes).

Adaptive Average Pooling Layer

This layer compresses the time sequence output from the Transformer Encoder Blocks along the time axis into a summarized feature vector. Each sequence is reduced to a vector of shape batch_size × d_model by averaging over the time dimension.
Using this layer ensures that the output has a fixed size regardless of input length, eliminating the need for manual dimension adjustment—especially useful for training datasets or future inputs. Additionally, averaging smooths out small fluctuations in the data, reduces dimensionality, and helps the model learn more effectively.

Fully Connected Layer

This layer maps the compressed output from the Adaptive Average Pooling layer to the number of target classes (here, 10 classes). The input dimension is batch_size × d_model, and the output is batch_size × num_classes.

Preprocessing Audio Files and Building the Dataset

Since the goal here is to classify audio files, the first step is to build a dataset that pairs each file's label with its feature vector. To do this, we extract the label of each .wav file from its filename and store it in a list. We also convert the corresponding audio file into a log-mel spectrogram and store the resulting vector in another list. Using these two lists, we construct the main dataset. Next, we randomly split the data into 80%, 20%, and 20% for training, testing, and validation, respectively.
Then, we divide the data into batches of size 32. While this number can be tuned for better results, based on prior experience and time constraints, we chose 32 as the batch size. Finally, we pad the feature vectors with zeros to make them the same size within each batch. To do this, we find the longest vector in each batch and pad the others to match its size.

Model Training

To prevent model overfitting, we implemented early stopping with a patience mechanism set to 5. This means that in each epoch, the validation accuracy is calculated, and if this value does not improve over the next few iterations (as determined by the patience parameter), training is halted. The best model (corresponding to the best validation accuracy within the patience window) is then loaded from a saved file and used for testing. After extensive trial and error, the learning rate was set to 1e-4. Figure 3 shows the accuracy curves for the training and validation datasets:

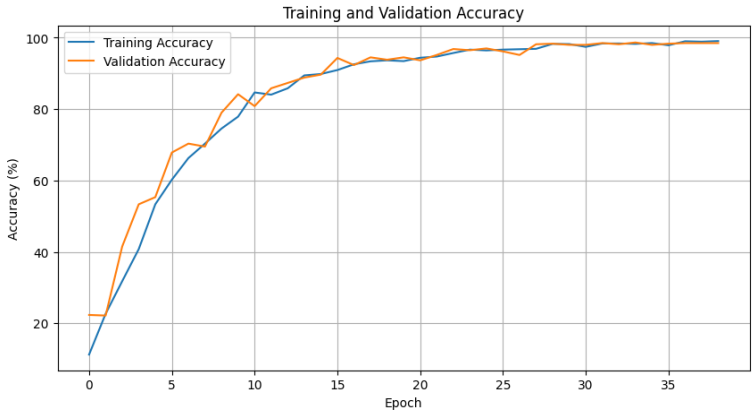

Figure3 - Models Accuracy for Validation and Test Sets (Clean Dataset)

In this process, training stopped at epoch 34.

Model Performance Evaluation

In this training process, the test accuracy reached 99.5%, which is considered a high level of precision. The confusion matrix for the model's performance is shown below:

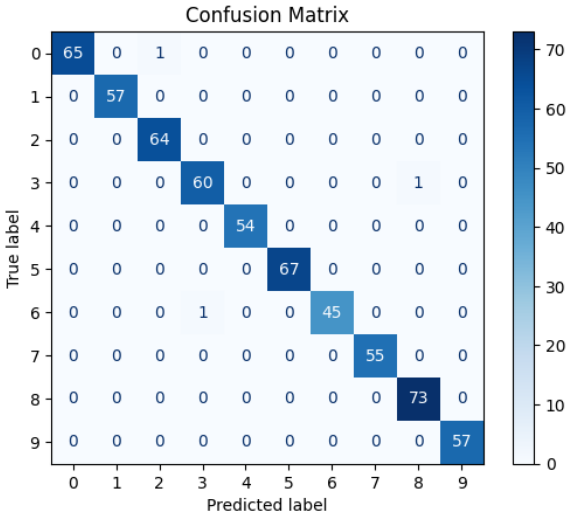

Figure 4- Confusion Matrix (Cleaned Dataset)

Noise Sensitivity Analysis

SNR (Signal-to-Noise Ratio) is the ratio of signal power to noise power in a system and serves as a measure of signal quality or clarity in the presence of noise. It indicates how dominant a useful signal is compared to the existing noise and is typically expressed in decibels (dB), calculated as follows:

$$\mathrm{SNR_{dB}} = 10 \cdot \log_{10}\left(\frac{\mathrm{P_{signal}}}{\mathrm{P_{noise}}}\right)$$

We generate white noise for SNR_dB values in the range [10, -10] with a step of 5, and add it to each feature vector in the training dataset. More precisely, in the first step, we compute the appropriate noise power using the formula:

$$\mathrm{P_{noise}} = \frac{\mathrm{P_{signal}}}{10^{\frac{\mathrm{SNR_{dB}}}{10}}}$$

Then, we sample a random vector from the distribution $N\left(0, \sqrt{P_{noise}}\right)$ with the same size as the feature vectors and add it to the original features. After applying this process, the model's accuracy for each SNR value is as follows:

Table 1 - Accuracy of the Model Under Different Amount of SNR

SNR(dB)	***Accuracy***
-10	*19.17%*
-5	*29.5%*
0	*49.33%*
5	*80.50%*
10	*92.00%*
15	*96.83%*
20	*98.33%*

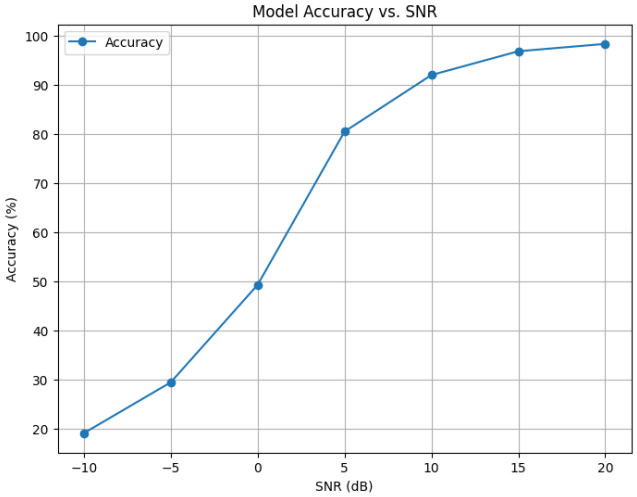

Figure5 - Model's Accuracy Considering Different Amount of SNR Trained on Clean Dataset

As shown, in the case of SNR = 0—where the noise power equals the signal power—the model's accuracy drops to 50%. In the next section, we perform operations to improve noise robustness.

Noise Robustness

There are various techniques in the literature for making a model robust to noise. One widely used method, especially in image processing models, is the technique of **Data Augmentation**. In this

approach, for each vector in the dataset, noisy versions are generated and added to the dataset so that the model is trained to handle noisy inputs as well.

In this stage, for each training sample, we generate 20 noisy samples where the SNR is randomly drawn from the distribution $U(-20, 20)$, and we add them to the training set. Additionally, to allow the model to train for the required number of epochs, we also generate 5 noisy samples for each validation sample using the same method and add them to the validation set. The training and validation accuracy during this process is shown below:

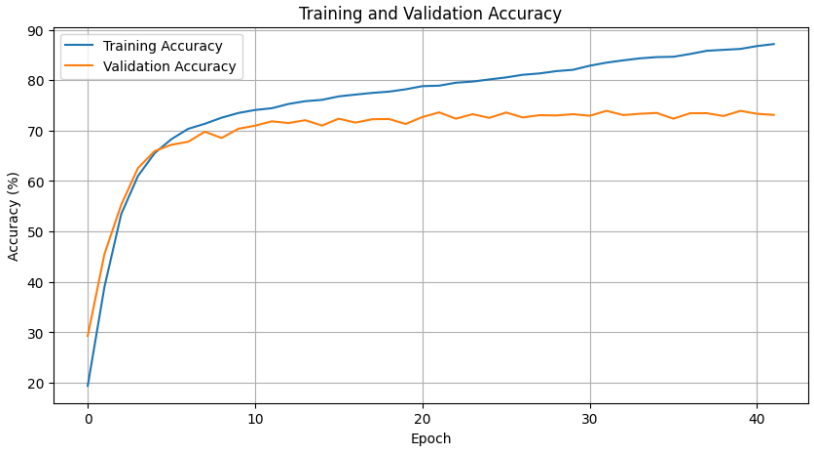

Figure 6 - Noise Robust Models Accuracy for Validation and Test Sets

In this training, the patience was set to 10 to prevent the model from getting stuck in a local optimum. Training concluded after 22 epochs. In this case, the test accuracy (on clean data) slightly decreased compared to the previous model, reaching 98.1%. The corresponding confusion matrix is shown below:

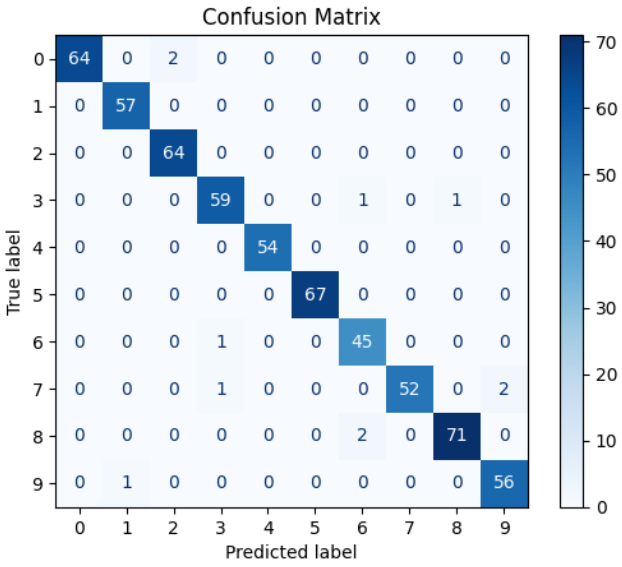

Figure 7 - Confusion Matrix (Noisy Dataset)

We repeated the experiment from (Model Performance Evaluation) using the new model. The results are as follows:

Table2 - Accuracy of the Noise Robust Model Under Different Amount of SNR

SNR(dB)	***Accuracy***
-30	*10.33%*
-25	*11.83%*
-20	*13.83%*
-15	*21.33%*
-10	*39.67%*
-5	*71.83%*
0	*89.83%*
5	*96.67%*

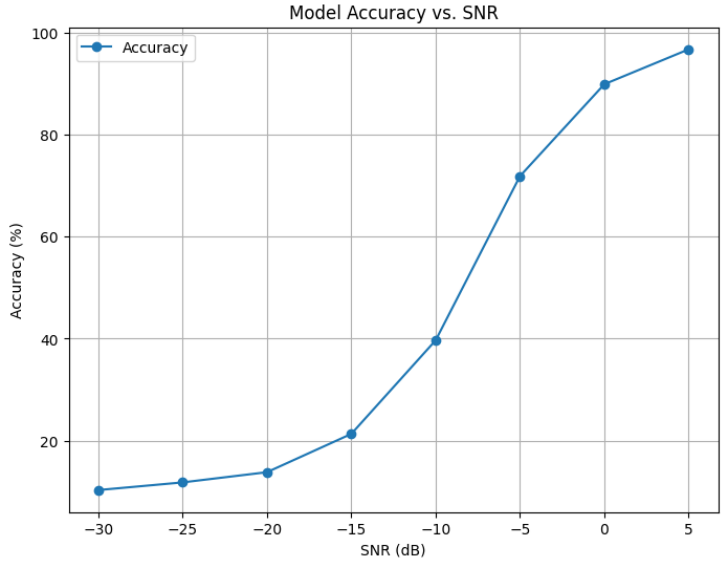

Figure8 - Noise Robust Model's Accuracy Considering Different Amount of SNR

As shown, the model's accuracy has significantly improved compared to the previous version. In cases where the noise power is less than or equal to 0.2 times the signal power, the model's performance remains unaffected. Even when the noise power equals the signal power, the accuracy improved from around 50% to approximately 90%, indicating the effectiveness of the applied method in scenarios where the noise is not excessively strong. However, in cases where the noise power exceeds five times the signal power, this method becomes ineffective. In such scenarios, other techniques mentioned in the Whisper paper should be considered.