Keyword spotting - Detecting commands in speech

Dhanya Girdhar
B.Tech CSE (Batch of '27)
Shiv Nadar University
Delhi, India
dg218@snu.edu.in
Roll No: 2310110463

Raashi Sharma
B.Tech CSE (Batch of '27)
Shiv Nadar University
Delhi, India
rs708@snu.edu.in
Roll No: 2310110566

***Abstract*—Keyword Spotting (KWS) enables speech-driven devices to detect short voice commands with low latency and computational cost, making it fundamental to applications such as smart speakers, mobile assistants, IoT devices, and automotive control systems. Real-world deployment, however, demands robustness against acoustic noise, microphone distortion, and speaker variability. This paper presents a comparative evaluation of classical and deep learning approaches for noise-resilient KWS. We examine: (i) Convolutional Neural Networks trained on raw audio waveforms, (ii) MFCC-based models including HMM-GMM, LSTM, and Attention-enhanced RNNs, and (iii) a proposed BiLSTM-DenseNet hybrid architecture.**

I. INTRODUCTION

Keyword Spotting (KWS) is a core component of modern voice-controlled systems such as smart speakers, mobile assistants, IoT devices, and automotive interfaces, where rapid and accurate detection of short voice commands is essential. Real-world deployment introduces significant challenges due to background noise, microphone variability, and differences in speaker accents and pronunciation. Developing models that remain robust under such conditions is therefore critical for reliable performance.

Recent advances in machine learning offer multiple architectural choices for KWS, ranging from lightweight statistical models to deep neural networks capable of learning complex time-frequency patterns. However, the trade-offs between model complexity, feature representation, and noise robustness are not fully understood.

This work presents a comparative evaluation of classical and deep learning approaches-including HMM-GMM, LSTM, Attention-RNN, BiLSTM-DenseNet, and raw-waveform CNNs-to analyze their effectiveness for noise-resilient keyword detection and to provide insights for practical deployment in real acoustic environments.

II. METHODOLOGY

A. *MEL FRQUENCY CEPSTRAL COEFFICIENT (MFCC)*

Mel-Frequency Cepstral Coefficients (MFCCs) provide a compact, perceptually motivated representation of speech. Since the human auditory system is more sensitive to lower frequencies than higher ones, a Mel-scaled triangular filterbank is applied to the short-time Fourier magnitude spectrum to emphasise frequency regions most relevant for phoneme discrimination.

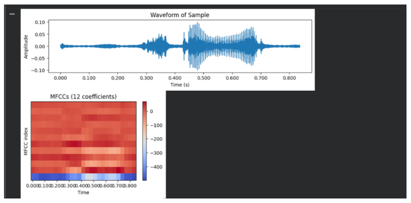

Fig. 1. 12 MFCC coefficients

To compute MFCCs, the audio waveform is first segmented into short overlapping frames (20-25 ms), each assumed to be locally stationary. Every frame is windowed and transformed using the Fast Fourier Transform (FFT). The resulting spectrum is passed through the Mel filterbank, and the logarithm of each filter's energy is taken to stabilise amplitude variations. A Discrete Cosine Transform (DCT) is then applied to decorrelate the filterbank outputs, producing cepstral coefficients; the first 12 coefficients capture the broad spectral envelope of speech.

To model temporal dynamics, first-order (Δ) and second-order ($\Delta\Delta$) derivatives are computed and concatenated with the static MFCCs, yielding a 36-dimensional feature vector per frame. This MFCC+(Δ)+($\Delta\Delta$) representation is used as the input for all MFCC-based models evaluated in this work.

B. *DATASET*

All experiments are conducted using the Google Speech Commands v2 dataset, containing one-second audio clips of spoken keywords recorded by multiple speakers under varied acoustic conditions. We split our data into training, validation and test sets with an 80-10-10 split. Since the goal of our task is to enable the model to correctly identify the commands, we use accuracy as our metric for evaluation. The test accuracy is calculated for all model frameworks and reported together as a comparative analysis.

C. *PREPROCESSING*

All MFCC-based models in this study follow a unified preprocessing pipeline using the Speech Commands v2 dataset. Each audio file is resampled to 16 kHz, normalized to unit amplitude, and trimmed for leading and trailing silence. We

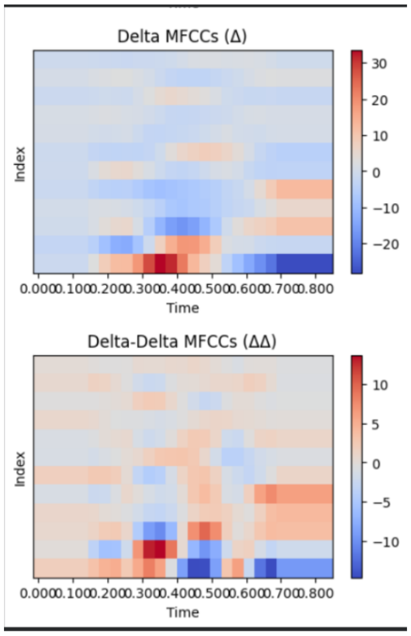

Fig. 2. First-order & Second-order derivatives of MFCCs

extract 12 MFCCs per frame and compute their and derivatives using a small regression window. These are stacked to form a 36×T feature map that serves as the input to all MFCC-based architectures.

D. NOISE AUGMENTATION

To assess robustness, a controlled noise-augmentation scheme is applied. With a 0.10 probability, a random segment from the dataset's background-noise files is mixed with the clean waveform at a 9:1 ratio after amplitude normalization. MFCCs are then recomputed from the augmented signal to maintain feature consistency. All processed samples are stored as NumPy tensors, and a metadata file records class labels, original filenames, speaker IDs, and the standardized train/validation/test split.

$$y_{\text{aug}}[n] = 0.9\, c[n] + 0.1\, n[n]$$

Among all architectures evaluated, the CNN is the only model trained directly on raw waveforms; all others (HMM-GMM, LSTM, Attention-RNN, and BiLSTM-DenseNet) operate on MFCC features derived from the standardized preprocessing pipeline. Unlike sequence models, CNNs do not require handcrafted spectral representations: they learn local filters that naturally extract meaningful structures from the raw signal. For this reason, the CNN is fed with the resampled waveform rather than MFCCs. All audio is downsampled to 8 kHz prior to training, which reduces input dimensionality and removes high-frequency components that contribute little to keyword recognition while retaining the essential speech information.

III. MODEL ARCHITECTURES

A. Hidden Markov Model with Gaussian Mixture (HM-MGMM)

We use the HMM–GMM model as our baseline. In this framework, we define N hidden states and M distinct observation components per state. The model is parameterized by:

- The state–transition matrix A, where each entry A_{ij} denotes the probability of transitioning from state i to state j.
- The emission probabilities B, where each $B_i(k)$ represents the likelihood of emitting observation k in state i, modeled using a Gaussian Mixture Model (GMM).
- The initial state distribution π, where π_i denotes the probability of beginning in state i.

Let the model parameters be denoted by $\theta = \{A, B, \pi\}$, and let z represent a sequence of hidden states. The goal is to maximize the marginal likelihood of the observed sequence x:

$$P(x \mid \theta) = \sum_{z} P(x, z \mid \theta).$$

We optimize this objective using the Expectation–Maximization (EM) algorithm. In the E–step, we compute:

$$q(z) = P(z \mid x, \theta^{\text{old}}),$$

the posterior distribution over hidden state sequences, using the current parameter estimate θ^{old}.

In the M–step, we update the parameters by maximizing the expected complete–data log-likelihood:

$$Q(\theta, \theta^{\text{old}}) = \sum_{z} P(z \mid x, \theta^{\text{old}}) \log P(x, z \mid \theta),$$

while keeping $P(z \mid x, \theta^{\text{old}})$ fixed.

In our experiments, we use $N = 3$ hidden states and a mixture of 2 Gaussians per state.

Training consists of an *outer* loop of Baum–Welch iterations, which refine the state transition probabilities and state occupancies, and an *inner* loop of EM updates for re-estimating the GMM emission parameters. The configuration used is:

- 5 Baum–Welch iterations (outer loop),
- 3 states per HMM,
- 8 EM iterations per state's GMM (inner loop),
- 36-dimensional acoustic features.

Thus, each keyword model undergoes:

$$\text{Total GMM updates per word} = 5 \times 3 \times 8 = 120.$$

B. Convolutional Neural Networks (CNN)

The convolutional model used in this work is a 1D CNN inspired by the M5 architecture, designed specifically for keyword spotting directly from raw audio. Each utterance from the Speech Commands dataset is resampled to 8 kHz. This downsampling preserves the temporal structure of keywords while discarding high-frequency regions dominated by silence, resulting in a compact and noise-reduced representation.

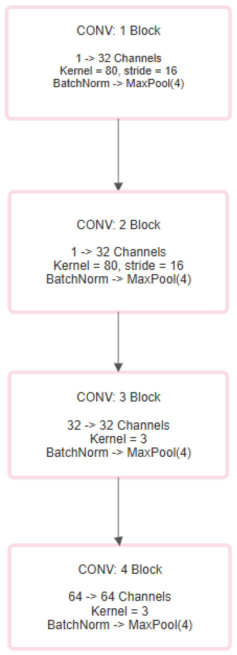

Fig. 3. CNN Architecture

After the final pooling layer, a global average pooling operation compresses the temporal dimension into a fixed-size embedding independent of input length. A fully connected layer maps this embedding to the final 35-class keyword output space, and the model outputs log-probabilities via log-softmax.

The 1D CNN (M5) model is trained directly on raw waveform inputs using the following configuration:

- **Optimizer:** Adam optimizer with learning rate 0.01 and weight decay 10^{-4}.
- **Batch size:** 256.
- **Epochs:** 10.
- **Learning rate scheduler:** StepLR with step size 20 and decay factor $\gamma = 0.1$.
- **Loss function:** Negative Log-Likelihood Loss (NLLLoss), applied after the model's log-softmax output layer.

C. Recurrent Neural Networks (RNN)

We also adopt RNN for our task considering the sequential nature of the inputs. RNN is a form of deep neural nets that exhibit temporal dynamics. RNN differs from the CNN in that at each time step of the input sequence, a hidden state is computed by taking both the input and the previous hidden state into account, and this hidden state is then used produce the output. Figure 1 illustrates this structure. Instead of the vanilla RNN cell, we adopt two variants of it. We start by building a RNN with unidirectional LSTM and modify this model by adding Bidirection LSTM cell and the attention mechanism.

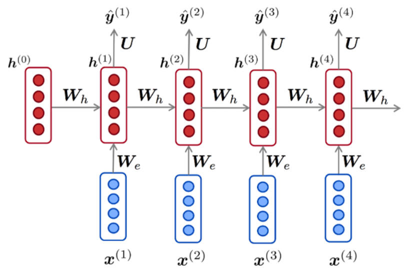

Figure 1: RNN Unrolled

- **Long Short-Term Memory** The Long Short-Term Memory (LSTM) network addresses the vanishing gradient problem and enables long-range temporal dependencies through its gated architecture. [**?**]. The gating mechanism regulates how information is forgotten, updated, and exposed to the next time step.

  a) Forget Gate.: The forget gate decides which information from the previous cell state should be discarded:

$$f_t = \sigma(U_f h_{t-1} + W_f x_t),$$
$$k_t = c_{t-1} \odot f_t,$$
$$g_t = \tanh(U_g h_{t-1} + W_g x_t),$$

  where σ is the logistic sigmoid and $\odot$ denotes element-wise multiplication.

  b) Input (Add) Gate.: The input gate determines which new information will be added to the cell state:

$$i_t = \sigma(U_i h_{t-1} + W_i x_t),$$
$$j_t = g_t \odot i_t,$$
$$c_t = j_t + k_t.$$

  c) Output Gate.: The output gate selects the relevant portion of the cell state to output as the new hidden state:

$$o_t = \sigma(U_o h_{t-1} + W_o x_t),$$
$$h_t = o_t \odot \tanh(c_t).$$

  - **Input dimension:** 36 (12 MFCC + Δ + $\Delta\Delta$)
  - **Hidden size:** 128 units
  - **Number of layers:** 1
  - **Directionality:** Unidirectional
  - **Dropout:** 0.2 (applied within the LSTM)

  Both the clean and noise-augmented LSTM models are trained using the same configuration:

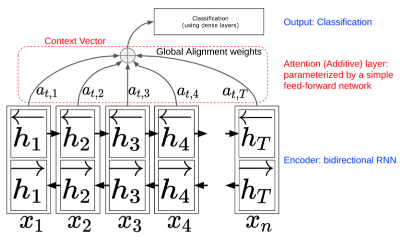
Classification (using dense layers)
Output: Classification
Context Vector
Global Alignment weights
Attention (Additive) layer: parameterized by a simple feed-forward network
$a_{t,1}$ $a_{t,2}$ $a_{t,3}$ $a_{t,4}$ $a_{t,T}$
$\overleftarrow{h_1}$ $\overleftarrow{h_2}$ $\overleftarrow{h_3}$ $\overleftarrow{h_4}$ $\overleftarrow{h_T}$
$\overrightarrow{h_1}$ $\overrightarrow{h_2}$ $\overrightarrow{h_3}$ $\overrightarrow{h_4}$ $\overrightarrow{h_T}$
Encoder: bidirectional RNN
x_1 x_2 x_3 x_4 x_n

Figure 2: BiLSTM RNN with attention mechanism

  - **Optimizer:** Adam
  - **Batch size:** 64
  - **Epochs:** 25
  - **Learning rate:** 0.001
  - **Scheduler:** StepLR with step size 5 and $\gamma = 0.5$
  - **Loss:** Cross-entropy

- **Attention-RNN + BiLSTM** Attention mechanisms have become an essential extension of RNN architectures, enabling models to focus on the most informative parts of a sequence. In keyword spotting, the spoken word may appear anywhere within the one-second audio clip, making frame-level relevance crucial. Attention allows the network to identify and emphasise the region containing the keyword rather than relying on a single summary vector.

  Traditional encoder-RNN models use only the final hidden state for prediction, forcing all temporal information into one representation. With attention, the model instead accesses all hidden states and assigns importance weights to each time step. This produces a weighted context vector that highlights speech-rich frames and suppresses silence and noise.

  In our architecture, a BiLSTM encoder combined with dot-product attention enables the model to localise the keyword and improve classification accuracy.

  The model is trained using the following configuration:

  - **Optimizer:** Adam with learning rate 1×10^{-3} and Weight decay: 1×10^{-5}
  - **Batch size:** 64
  - **Epochs:** 25
  - **Learning-rate scheduler:** StepLR with step size: 8 and Decay factor: $\gamma = 0.5$
  - **Loss function:** Cross-entropy loss

D. *BiLSTM + DenseNet*

The BiLSTM-DenseNet hybrid architecture combines recurrent temporal modelling with deep hierarchical time-frequency feature extraction.

1) Bidirectional LSTM Front-End: The first stage of the model is a two-layer Bidirectional LSTM (BiLSTM) that captures both forward and backward temporal dependencies. This is critical for keyword spotting, as phonetic cues often depend on context from both directions in time.

- Input dimension: 36
- Hidden size: 128
- Number of layers: 2
- Directionality: Bidirectional
- Dropout: 0.2 (0.3 in the modified configuration)
- Effective output size: $128 \times 2 = 256$ features per timestep

The BiLSTM converts the MFCC sequence into a rich temporal embedding that is then passed into the DenseNet convolutional module.

2) DenseNet1D Feature Extractor: The output of the BiLSTM is reshaped into a (B, C, T) tensor and processed with a 1D DenseNet architecture. DenseNet blocks encourage efficient feature reuse by concatenating outputs from all preceding layers, enabling both local and global time–frequency representations to be learned.

Two configurations of the DenseNet module were evaluated:
Configuration A (default):
- Dense blocks: 2
- Layers per block: 3
- Growth rate: 32
- Dropout: 0.2
- Epochs: 25
- Batch size: 64
- Learning rate: 0.001

Configuration B (modified):
- Epochs: 40
- Growth rate: 24
- Dropout: 0.3
- Other parameters unchanged

Both configurations achieved similar accuracy, demonstrating the stability of the architecture.

Each Dense Block contains three DenseLayer1D modules. Each DenseLayer1D consists of:
- Batch Normalization
- ReLU activation
- 1D Convolution (kernel size = 3, padding = 1)

Each layer adds a fixed number of growth channels and concatenates its output depth-wise with all previous outputs:

$$X_{l+1} = \mathrm{Concat}(X_0, X_1, \ldots, X_l).$$

This progressively increases channel dimensionality while maintaining temporal resolution.

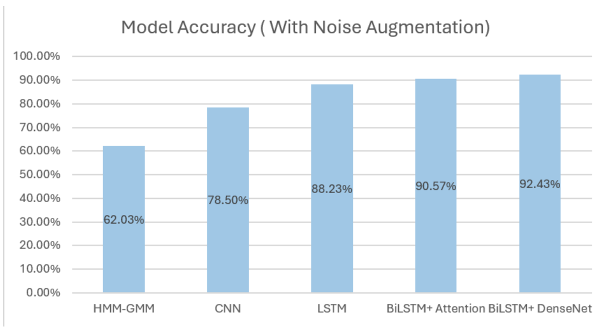

Fig. 4. Model Accuracy (With Noise Augmentation)

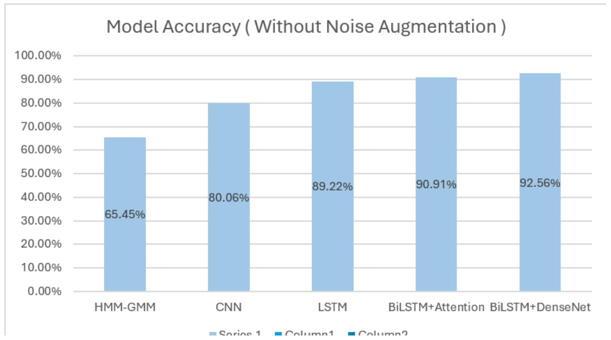

Fig. 5. Model Accuracy (Without Noise Augmentation)

a) Transition Layer: After each dense block, a Transition1D layer reduces the number of channels through:

- BatchNorm
- ReLU
- 1×1 Convolution

The number of output channels is halved (with a minimum of 16), reducing model size while preserving feature richness.

3) Temporal Pooling and Classification: Following the DenseNet feature extractor, an Adaptive Average Pooling layer collapses the temporal dimension:

$$(B, C, T) \rightarrow (B, C, 1),$$

allowing the network to handle variable-length input sequences.

The final classification head consists of:

- Flatten operation
- Dropout (0.2 or 0.3 depending on configuration)
- Fully connected layer mapping $C \rightarrow 35$ keyword classes

IV. EXPERIMENTAL RESULTS

Figure 3 (with augmentation) and Figure 4 (without augmentation) summarise the performance of five keyword-spotting models: HMM-GMM, CNN, LSTM, BiLSTM + Attention, and BiLSTM + DenseNet. Across both conditions, the results follow a consistent ranking:

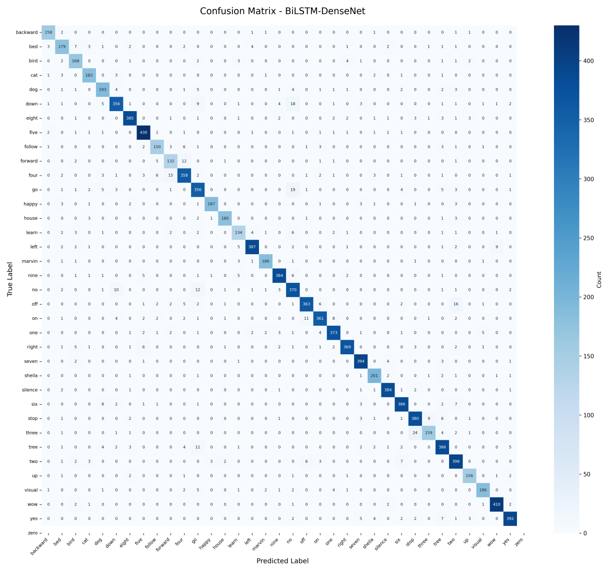

Fig. 6. Confusion Matrix BiLSTM + DenseNet

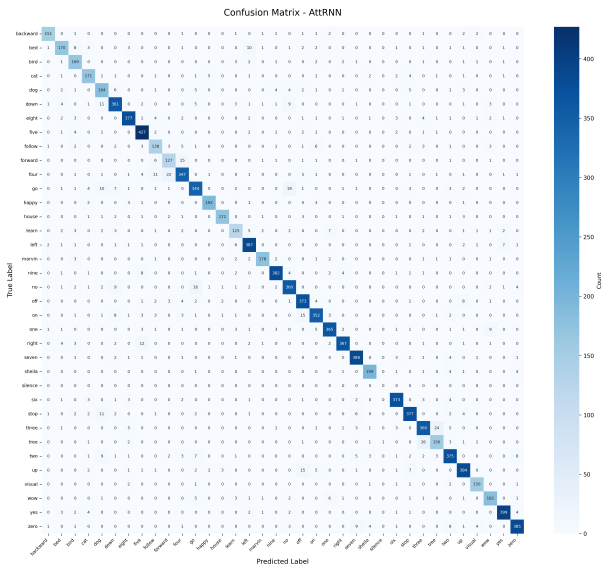

Fig. 7. Confusion Matrix ATNN-RNN + BiLSTM

HMM–GMM < CNN < LSTM < BiLSTM+Attention < BiLSTM+DenseNet

Without noise augmentation, the accuracies range from **65.45%** (HMM-GMM) to **92.56%** (BiLSTM + DenseNet). With noise augmentation, the accuracies range from **62.03%** to **92.43%**, indicating minimal degradation for deep neural models but a noticeable drop for classical models.

Noise augmentation was applied mildly (10% probability, 10% mixing ratio) and uniformly across all dataset splits. These results show that:

- Classical models such as HMM-GMM are more sensitive to added noise.

- Neural architectures demonstrate high stability, with performance changes typically within $\pm1\%$.

Overall, deep recurrent-convolutional hybrid architectures consistently achieve the strongest performance in both clean and noise-augmented settings.

V. Discussion

A. Model Capacity vs. Accuracy

Performance improves consistently with increasing model complexity. The HMM–GMM model, which relies solely on frame-level MFCC likelihoods without deep temporal modelling, shows the lowest accuracy.
CNNs, despite operating directly on raw waveforms, outperform HMM–GMM by learning localized temporal filters. LSTMs further improve performance by capturing long-term temporal dependencies in MFCC sequences. BiLSTM models, which incorporate both forward and backward temporal context, extract richer acoustic structure.
Finally, adding DenseNet convolutional layers on top of BiLSTM outputs enables feature reuse and multi-scale representation, yielding the highest accuracy overall.

B. Impact of Noise Augmentation

Noise augmentation affects the models unevenly:

- **HMM-GMM**: experiences a degradation of roughly 3–4%, indicating weak robustness to additive noise.
- **CNN / LSTM**: show only mild fluctuations (approximately 1–2%), reflecting moderate noise resilience.
- **BiLSTM + Attention / BiLSTM + DenseNet**: exhibit negligible performance drops ($< 1\%$), demonstrating strong robustness.

These results highlight the inherent ability of deep neural architectures to learn noise-invariant representations even when trained primarily on clean data.

C. Effects of Attention and Dense Connectivity

The attention mechanism allows the model to focus selectively on the relevant temporal regions containing the keyword, improving performance relative to plain LSTMs. DenseNet layers further enhance feature propagation and reuse, explaining why the BiLSTM + DenseNet model consistently achieves the strongest results under both clean and noise-augmented conditions.

VI. Conclusion

The experiments conducted in this work provide a comprehensive comparison between classical acoustic modelling techniques and modern deep learning architectures for keyword spotting. The results demonstrate several clear trends:

- **Deep learning models substantially outperform traditional HMM-GMM systems**, achieving improvements of approximately 25–30 percentage points in accuracy.
- **Hybrid architectures that combine BiLSTMs with DenseNet-style convolutional blocks achieve the strongest overall performance**, consistently exceeding 92% accuracy across both clean and noise-augmented conditions.
- **Attention mechanisms and dense connectivity play a significant role** in boosting performance by allowing the model to emphasize speech-rich temporal regions and efficiently reuse multi-scale time-frequency features.
- **Noise augmentation has only a limited effect on deep models**, indicating that modern architectures learn inherently robust representations on the Speech Commands dataset.

Overall, the findings confirm that deep recurrentconvolutional -hybrid models provide the most effective and reliable solution for keyword spotting. They offer both high accuracy and resilience to mild acoustic perturbations, making them suitable for real-world deployment in low-latency, noise-prone environments.

References

[1] S. Rai, T. Li, and B. Lyu, *Keyword Spotting: Detecting Commands in Speech Using Deep Learning*, 2023. Available: arxiv.org/pdf/2312.05640.

[2] A. Karpathy, *The Unreasonable Effectiveness of Recurrent Neural Networks*, 2015. Available: karpathy.github.io.

[3] A. Author et al., *Effective Combination of DenseNet and BiLSTM for Keyword Spotting*, 2019. Available: researchgate.net.

[4] T. Sainath and R. Parada, *Convolutional Neural Networks for Small-Footprint Keyword Spotting*, 2015. Available: semanticscholar.org.

[5] Author(s), *Keyword Spotting in Continuous Speech Using Deep Neural Networks*, Speech Communication, 2022. Available: dl.acm.org.

[6] Author(s), *Hybrid Neural Architectures for Keyword Spotting (DenseNet + BiLSTM / Attention)*, Available: researchgate.net.

[7] Author(s), *Generalisation Gap of Keyword Spotters in a Cross-Speaker Study*, 2021. Available: pmc.ncbi.nlm.nih.gov.