Word recognition using machine learning on Arduino Nano 3 BLE.

Introduction:

The main aim of this project is to develop a system for recognizing specific trigger words using an Arduino Nano 33 BLE, TensorFlow, TensorFlow Lite, and the Tiny Conv model. This process, known as hot word recognition, involves the system's ability to detect and identify certain keywords or phrases from spoken input. For this project, the target hot words are "all," "none," "never," and "must." To achieve this, the project will utilize the processing power of the Arduino Nano 33 BLE, combined with the capabilities and adaptability of TensorFlow and TensorFlow Lite. The Tiny Conv model architecture will be used to ensure efficient performance and resource management on the Arduino Nano 33 BLE platform.

Experiment:

In the experiment, the procedure for gathering data entailed capturing audio recordings for each specified hot word through open speech recording. I personally recorded 60 audio clips for each hot word. These recordings were then combined with a dataset supplied by the professor, creating a comprehensive collection for the project.

The final dataset contains 5,264 files in total. The breakdown of instances for each hot word category is as follows:

1. "all": 967 instances.
2. "must": 1140 instances.
3. "never": 1038 instances.
4. "none": 1099 instances.
5. "only": 1020 instances.

This distribution ensures a well-balanced representation of the hot words in the dataset. It guarantees that the model is trained and evaluated with a diverse range of instances. The addition of 60 audio clips per word enhances the dataset's diversity, contributing to the hot word recognition system's overall effectiveness and its ability to generalize.

Algorithm:

The design, architecture, training process, and associated parameters of the machine learning algorithm for recognizing hot words encompass several crucial elements.

Data Preprocessing:

The first step involves preprocessing the .ogg files by transforming them into .wav format, utilizing a specialized dataset script included in the assignment. The model functions on spectrograms, which are two-dimensional arrays that consist of slices of frequencies extracted from separate time intervals.

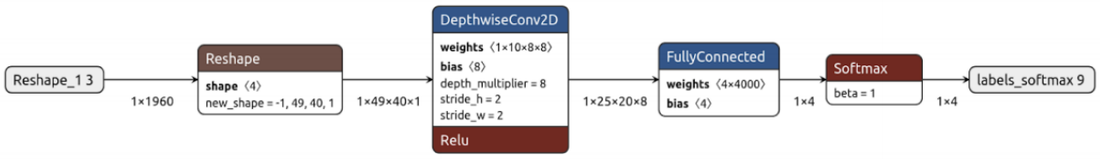
Spectrogram Generation:

Spectrograms are generated by using the Fast Fourier Transform (FFT) on a 30ms segment of audio data. This data is considered as real numbers ranging from -1 to +1. The analysis uses a 20ms step for the audio sampling, leading to 10ms of overlap in each window. The FFT process generates 257 values, and every group of six values is combined to form 40 frequency groups for each segment. These results are then processed through steps like downscaling, noise reduction, automatic gain control, and another round of down-scaling. This method is continuously applied to sequential time segments, resulting in a one-dimensional image with a size of 40 pixels in width and 49 rows in height.

Model Architecture:

The selected model structure is known as 'tiny_conv.' It is a straightforward architecture that includes a 2D Convolutional layer, followed by a Fully Connected Layer, also known as a Matrix Multiplication (MatMul) Layer, which generates logits, and concludes with a Softmax layer. The configuration of this model is laid out in the following sequence:

- A Conv2D layer equipped with weights.
- A BiasAdd operation that incorporates bias.
- The application of a Relu activation function.
- A MatMul operation, also equipped with weights.
- Another BiasAdd operation including bias.



Training Parameters:

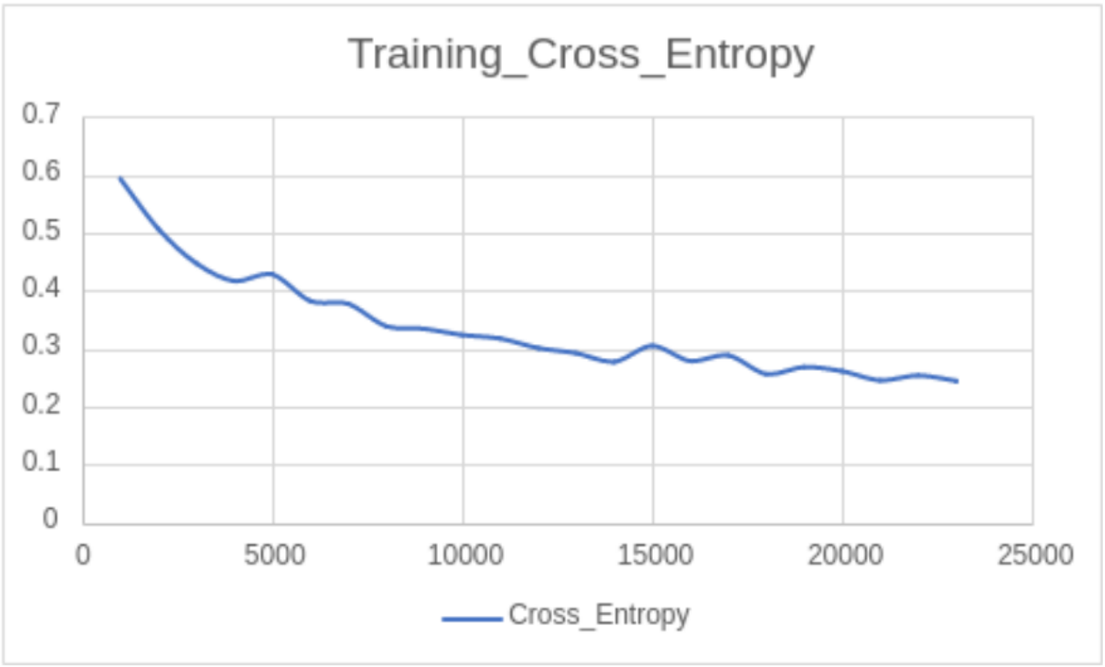
The algorithm's configuration includes an initial training phase consisting of 20,000 steps, followed by a fine-tuning phase of 3,000 steps. The learning rate begins at 0.001 and is reduced to 0.0001 during the fine-tuning phase. To enhance the dataset's variety, silent and unknown samples constitute 10% of the data. The model undergoes evaluations and data saving at intervals of every 1000 steps throughout the training process. Additionally, the quantization parameters are set to determine the range for input data quantization. This setup, including the model's architecture and training parameters, is tailored for efficient processing of spectrogram data, aiming to achieve precise recognition of specific keywords or "hot words" in the dataset.

Results:

Training Performance:

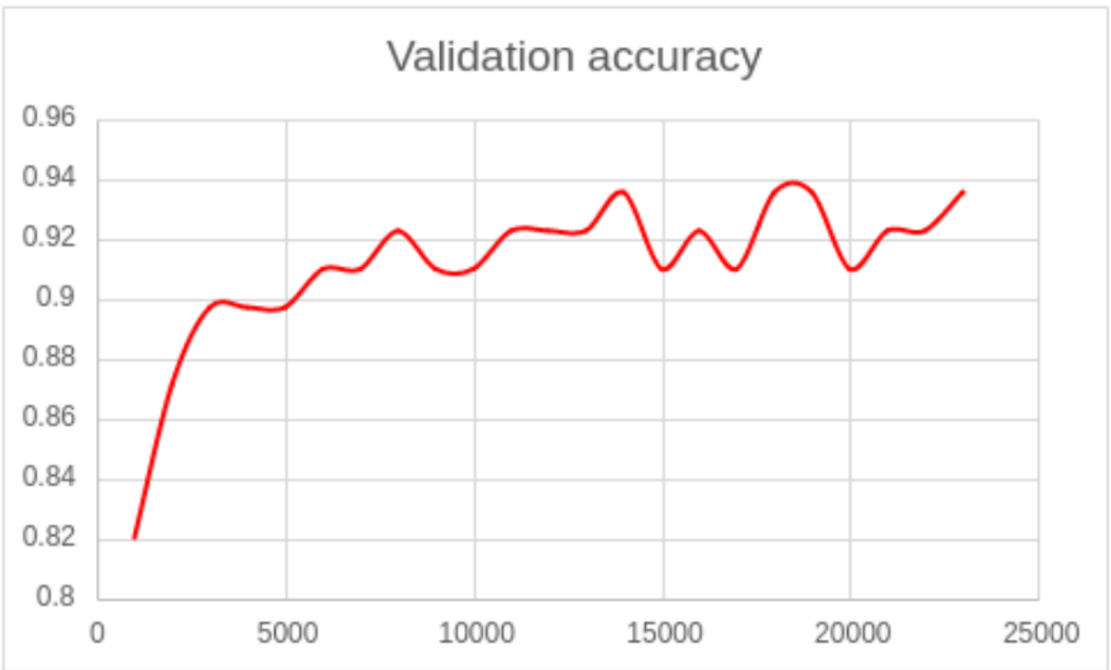
- Final Training Accuracy: 94.99%.

- Final Training Cross-Entropy: 0.2926.

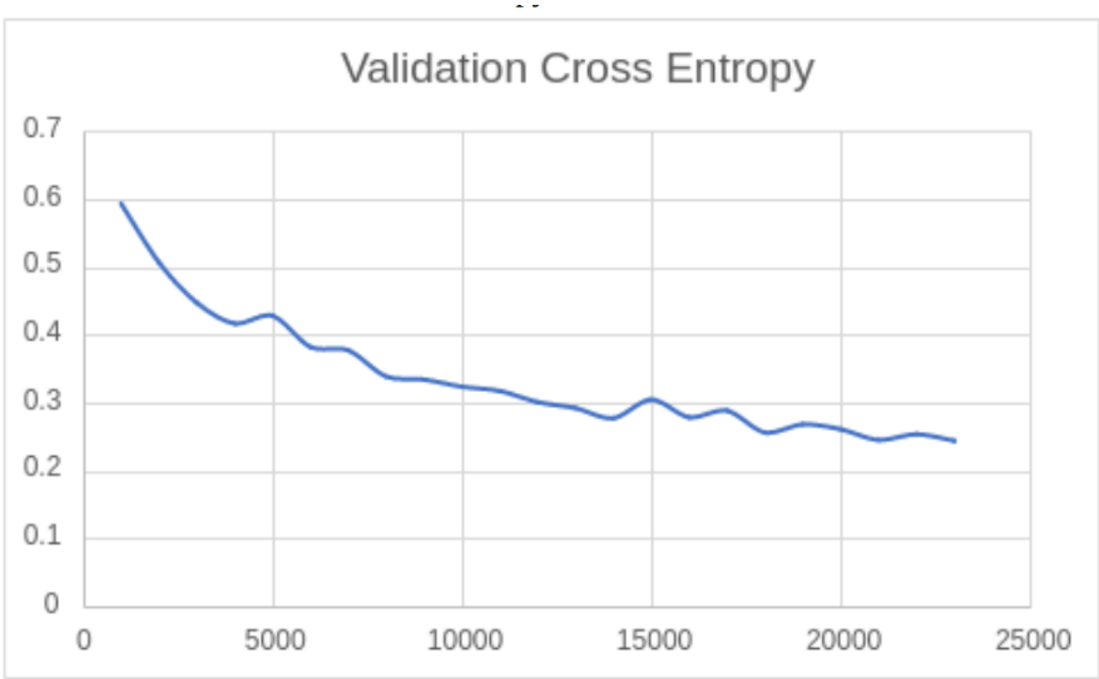


Validation Performance:

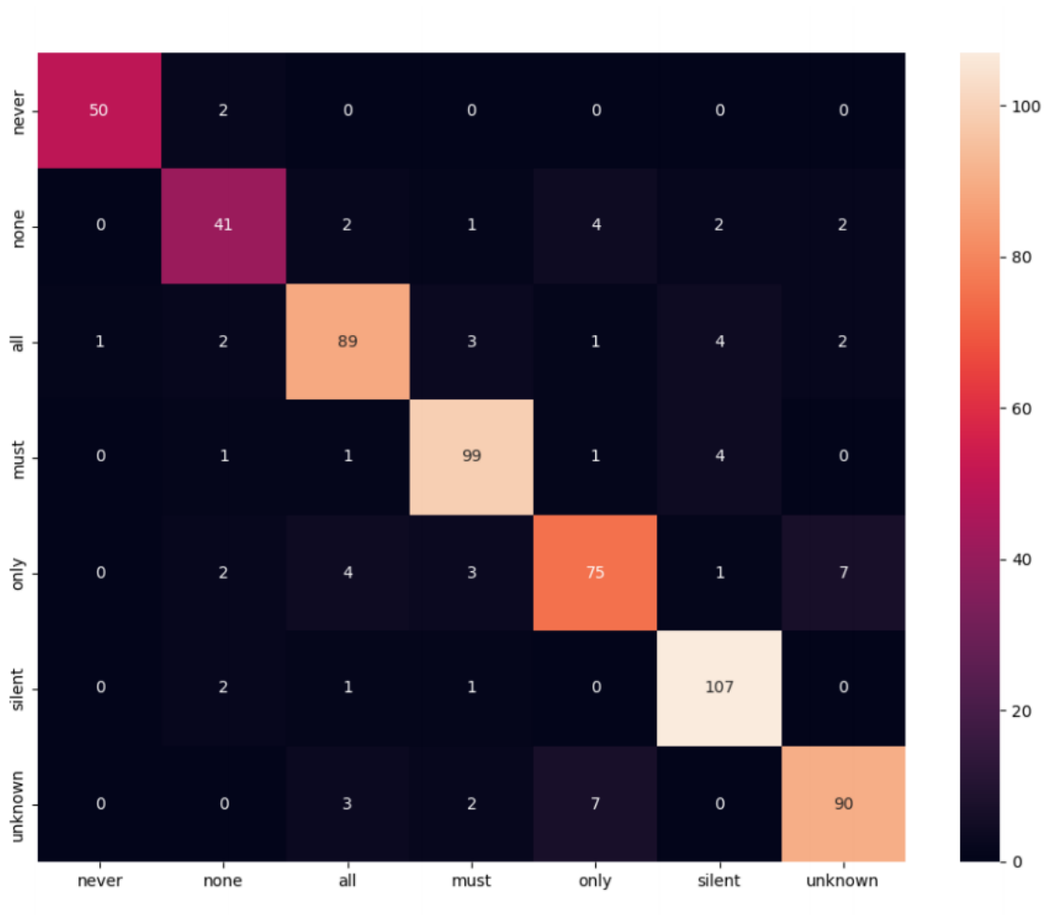
- Final Validation Accuracy: 93.59%.



- Final Validation Cross-Entropy: 0.2442.



Confusion Matrix:



Test Performance:

- Final TensorFlow Test Accuracy: 89.3% (N=617).

After model conversion:

- Float Model Accuracy: 90.19% (Number of test samples=2752).
- Quantized Model Accuracy: 90.15% (Number of test samples=2752).

Real-Time Prediction:

Predictions made in real-time closely match the general accuracy of the model. During the demonstration, the code was transferred to an Arduino Nano board, which used a microphone for input. The recognized command was displayed on the serial plotter. For visual cues, RGB lights were used:

- Cyan indicated "Unknown".
- Purple signified "never".
- Orange represented "none".
- Blue denoted "all".
- Green was used for "must".
- Red signaled "only".

Discussions:

The project focusing on hot word recognition achieved impressive outcomes, with a notable training accuracy of 94.99% and validation accuracy of 93.59%. The confusion matrix provided deep insights, highlighting the model's strong performance with various hot words. The model also demonstrated its reliability in dynamic settings through its real-time prediction capabilities, closely aligning with the test accuracy.

During the deployment phase on the Arduino Nano, the team encountered significant hurdles due to version inconsistencies, primarily caused by the original development in TensorFlow 1.15. These issues were skillfully resolved by integrating a specific code segment into the micro_speech.ino file, ensuring compatibility:

```
if (micro_op_resolver.AddConv2D() != kTfLiteOk)
{
  return;
}
```

A critical aspect of the project was generating a diverse dataset, which was crucial for training the model to achieve high accuracy and generalization. Despite challenges, the team successfully acquired the needed data.

The real-time predictions on the Arduino Nano met the project's expectations, efficiently processing audio inputs and accurately identifying commands. Looking ahead, future improvements could focus on enhancing the real-time demonstration for better usability and reducing computational demands for wider application on devices with limited resources. Overall, the project not only highlights the effectiveness of hot word recognition in real-time scenarios but also showcases the model's versatility in various deployment contexts.
