Homework 2 - Team 17

Francesca Cossu s305746@studenti.polito.it
Niccolò Dimonte s303439@studenti.polito.it
Michele Presti s290453@studenti.polito.it

Training and Deployment of a "Go/Stop" Classifier

1 Introduction

In this work, we built a model for speech recognition of "Go/Stop" keywords from a dataset of 1600 audios. To do this, we made use of MFCCs for audio signals and a DS-CNN model, and through TensorflowLite we developed a tflite model for edge devices that remains in the constraints:
$> 97\%$ *accuracy,* < 25*kb TfLite Size,* < 8 *ms latency*.

2 Methods

For the preprocessing part, we used MFCCs. MFCCs are a commonly used feature representation for speech signals, and they capture the spectral characteristics of speech in a compact form.

Hyper-parameters tuning

For the hyper parameters tuning, we proceed as the following:

- *downsampling rate*: in this case, we don't want to apply downsampling because that would result in increased latency, so downsampling rate is set equal to original one.
- *frame length*: as we know, tf methods are more efficient if we use window sizes that are powers of two.
- *frame step*: to reduce latency, we set the frame step equal to the frame length to have 0% overlapping.
- *num of mel bins*: in speech recognition, the number of bins refers to the number of frequency bands into which the audio spectrum is divided. A common value is 40 bins but the optimal number may depend on signal characteristics. A large number can provide more improvements in accuracy but increase also the computational costs. For our task, an optimal choice was 25.
- *lower and upper frequency*: the frequency range should cover the range of frequencies present in the speech signals that you are interested in recognizing. For human speech, a common frequency range is 0-8 kHz.
- *num of coefficients*: in the cepstral domain, the influence of the vocal cords (source) and the vocal tract (filter) in a signal can be separated since the low-frequency excitation and the formant filtering of the vocal tract are located in different regions in the cepstral domain. Depending on the sampling rate and estimation method, 12 to 20 cepstral coefficients are typically optimal for speech analysis.

The chosen hyper-parameters are in Table 1.

downsampling rate	16000
frame length in s	0.032
frame step in s	0.032
num mel bins	25
lower frequency	0
upper frequency	8000
num of coefficients	13

Table 1: Hyper Parameters for Preprocessing

3 Model Architecture and Optimizations

DS-CNN Model and Parameters for Training

For this task, we employed a DS-CNN model using different techniques together, like width scaling and pruning. Regarding the model, we used Conv2D, BatchNormalization, DepthwiseConv2D and ReLU (each repeated three times), followed by a Dense layer and a Softmax function.
In a CNN, the number of channels in the output of a convolutional layer is determined by the number of filters applied to the input data. In a depthwise CNN, each convolutional filter is applied to a single channel of the input data, rather than all channels. This allows the network to learn different filters for different channels of the data, which can be useful for tasks such as speech recognition where different channels may contain different types of information. One advantage of using depthwise CNNs for speech recognition is that they can be more efficient than traditional CNNs, as they have fewer parameters and require less computation. This can be particularly useful for tasks such as speech recognition, which often require real-time processing and may be implemented on devices with limited resources.
Concerning the convolutions, due to the constraints imposed on the size of the model and because of the ease of the task, we maintained a low number of filters for the 3 Conv2D, specifically [90, 45, 45]. The schema of the model is represented in Fig. 1.
Regarding number of training epochs and batch size, the number of epochs was set at 20 as it is a sufficient number to achieve good accuracy without overfitting, while due to the low complexity of the model and the need to decrease computational costs, we opted for a batch size of 8.

Pruning Low Magnitude Weights

Using the function *keras.pruning_low_magnitude* from *tensorflow_model_optimization* we performed some pruning on the least relevant weights. The weights of the last layers are the ones less trainable with respect to the first ones, so we impose the *initial_sparsity* to a value of 0.5 so that the pruning would start after the first layers.
With a grid search was found a trade-off between the number

of convolutional filters and the final sparsity of the pruning, with positive results in terms of accuracy and size.

Optimizations Methods

Given the constraints on model size, it was essential to optimize the model. We chose dynamic gamma post-training quantization as the method. Post-training quantization is a method of quantizing a neural network after it has been trained, rather than during the training process. Dynamic quantization involves quantizing the weights and activations of the network to lower bit depths (e.g. 8-bit) using a process called "min-max" quantization.
In min-max quantization, the minimum and maximum values of the weights and activations are first determined. Then, a quantization scale is calculated based on these values and used to quantize the weights and activations to the desired bit depth. Overall, post-training quantization with dynamic range is a method of reducing the precision of a neural network in order to reduce its size and computational requirements, while also trying to minimize the impact on the accuracy of the network.

batch size	8
epochs	20
initial learning rate	0.01
final learning rate	1.e-5
initial sparsity	0.5
final sparsity	0.85

Table 2: Hyper Parameters for Training and Pruning

4 Results

Given the imposed constraints, we try to maximize the results maintaining high performance. It was important to do a trade-off between the amount of pruning trying to not make drop accuracy too much and maintaining low the number of parameters, which directly impacts the size of the model. Even though latency is a fundamental factor for real-time applications, it was not an object of improvement because it remains below the value requested thanks to the window size (that are powers of 2) and the absence of overlapping. Thanks to the ease of the task it was not difficult to reach an accuracy over 97%. The final solution we choose reaches in 20 epochs the 98.94% of training accuracy and 97.5% test accuracy, keeping the latency at 5.8 ms and the zipped model's size at 23.66 kb (as shown in Table. 3).

Our approach to the trade-off between accuracy and size tends towards the increase of accuracy, keeping the constraints satisfied. Another interesting solution, especially for maintaining the size small, was given by lowering the number of filters in convolutions, obtaining a size of only 6 kb and 97% test accuracy.

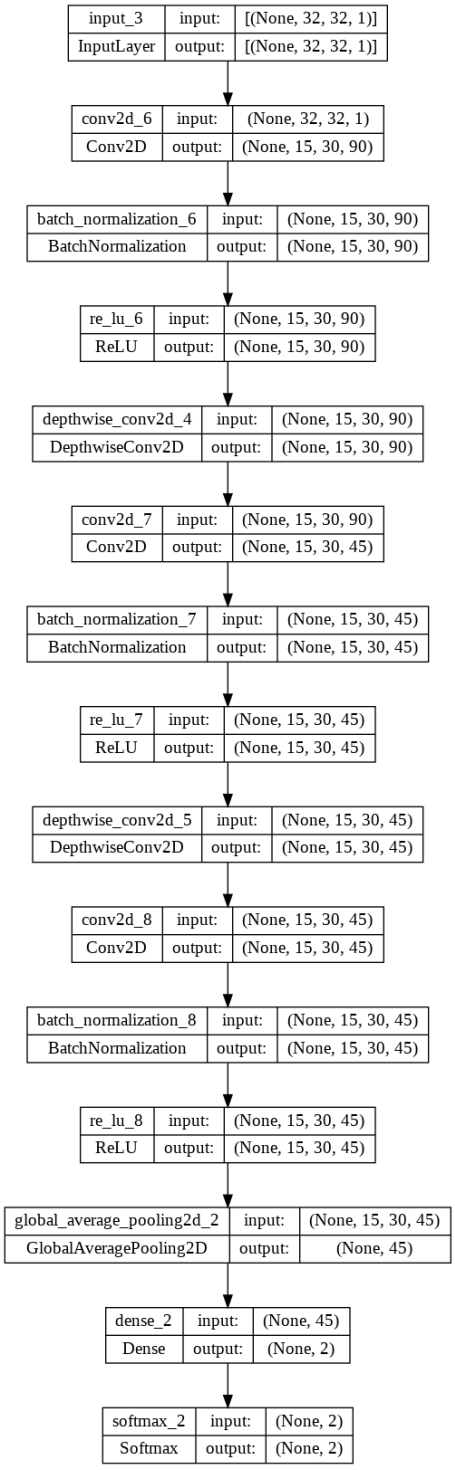

Figure 1: Model Architecture

Test Accuracy (%)	*97.5*
TfLite Model Size (zipped) (kb)	23.66
Total Latency (ms)	*5.8*

Table 3: Final solution