Homework 1

GROUP 17

Juan Aragon (s291466) Hadi Nejabat (s246601) Rostislav Timuta (s280238)

I. Temperature and Humidity TFRecord

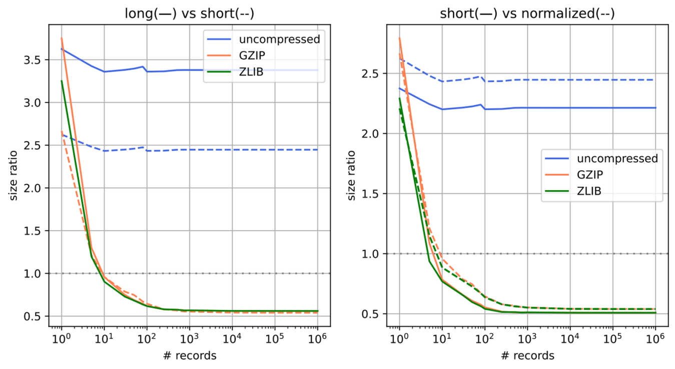

Fig. 1. File-size comparison expressed as the relative size to the original file.

The goal of this exercise was to work with *TFRecord*, a more efficient data format used to serialize and deserialize structured data, able to occupy less space in memory than traditional formats while boosting the speed of the information transfer process. For this exercise we worked with a structured dataset, containing time (in POSIX format), humidity and temperature records, obtained from a *DHT11* sensor.

For the first part (***long***), the data-set was read line by line, transforming each record into a dictionary of collected named features, using *tf.train.Feature()*. The 3 features of interest were named: *datetime*, *temperature* and *humidity*. Each of them encoded as FloatList. At this point 2 compression modalities were computed (*GZIP* and *ZLIB*). Obtaining the results shown in Fig.1.

A second experiment (***short***) was computed, similar to the previous one, just replacing the encoding features, using *Int64List* for sensor's data, and changing the names for each feature to *D*, *T* and *H*, respectively. A noticeable improvement was observed for the uncompressed versions.

Finally a last experiment (***normalized***) was done. By taking the *short* version, all sensor's data were normalized according to the max-min values for each feature and stored as FloatList.

For the uncompressed versions, noticeable variations were observed, the worst scenario was obtained at *long*, increasing the size of the original file on average by a factor of 3.4. However, all experiments showed an improvement in size reduction for any of the compression modalities, reaching almost a 50% of reduction for the large files.

II. Audio pre-processing optimization

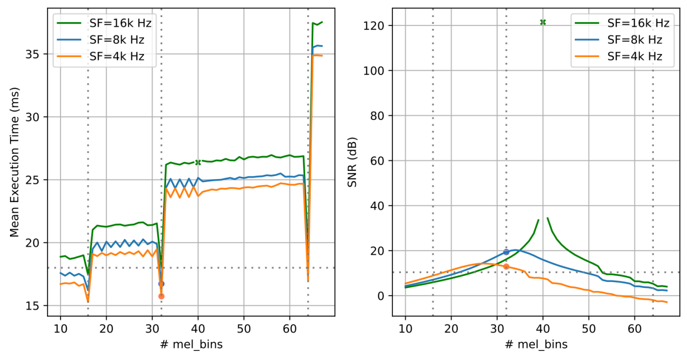

Fig. 2. Pipelines' performances for varying parameters.

The goal of this exercise was to accelerate the pre-processing pipeline for computing the *mfcc* of an audio file. Considering a standard implementation of it, using *Tensorflow*'s algorithms, the original pipeline was made with 4 stages: reading the audio file, computing the *stft*, calculating the mel's spectrum, and obtaining the mfcc. It was needed to choose the proper parameters for achieving on average 18ms of execution time for each transformation and an SNR above 10.4 dB with respect to its standard version.

The experiment consisted of optimizing 3 parameters: The sampling rate, which originally was 16kHz. The maximum frequency of the signal, which according to *Nyquist Shannon Theorem*, must be half the sampling frequency. And lastly, the number of mel_bins, being always greater than the number of desired coefficients (10), and initially set to 40.

For this purpose, all possible options were analyzed, obtaining the results shown in Fig.2. For the mean execution time, it was observed a reduction at the last stage of the pipeline by using values in the form 2^n, with n being an integer number. On the other hand, the SNR performed as a bell-curve, except for the case when the parameters were identical with the standard ones.

At the end of the experiments, two feasible configurations were obtained, satisfying both the constraints; as a result of the following parameter sets (SF=8kHz, MF=4kHz, mel_bins=32) and (SF=4kHz, MF=2kHz,mel_bins=32). Between these two solutions, it was considered the one with SF=8kHz as the optimal one, since it preserved a higher audio quality in terms of SNR. With this solution it was obtained an **ET** $\approx$ **16ms** (initially 29ms) and a **SNR = 15.06 dB**.

Homework 2

GROUP 17

Juan Aragon (s291466) Hadi Nejabat (s246601) Rostislav Timuta (s280238)

I. Multi-Step Temperature and Humidity Forecasting

Model	α	Sparsity	Compressed File size (kB)	T MAE (°)	Rh MAE (%)
A	0.04	0.90	1.20	0.26	1.20
B	0.04	0.90	1.72	0.64	2.46

TABLE I: Results ex 1

The goal of this exercise was to implement 2 multi-output models for temperature and humidity forecasting, to infer multi-step predictions while satisfying some constraints regarding the output's accuracy and memory consumption. Both models were trained, validated and tested using the Jena Climate Dataset[1], with a 70%/20%/10% split ratio. Initially, in both versions it was considered the standard models introduced on lab3: *MLP* and *CNN*. In general, it was proven that *CNN* outperformed *MLP* for the given constraints, since it was capable of retrieving more accurate results even with significant reductions on its number of filters (less memory space).

For the first model, a modified version of the original *CNN* was implemented, by removing the 1D-Convolutional layer and decreasing the number of filters by a factor of α, so the total amount of filters in the Dense layer was reduced from 64 units into 2 units. For the second model, all layers were considered as in the standard form, but the number of available filters was also modified by using the same value for the parameter α.

Then, in order to increase the sparsity, it was used the approach of *magnitude-based pruning*, with a *PolinomialDecay* to schedule how much sparsity to enforce in a training step. In both cases the sparsity started from an initial value of 0.3 and acheived a final value of 0.9.

After training our model in float32 we performed a *F16 quantization* in TFlite, the weights were converted to float16. In the end, to see the model size advantage due to pruning we compress our model with *zlib*.

From the TABLE I we can observe the different hyperparameters chosen and the results obtained for the two model versions.

[1]https://storage.googleapis.com/tensorflow/tf-keras-datasets/jena_climate_2009_2016.csv.zip

II. Keyword Spotting

Model	α	Compressed File size (kB)	Accuracy(%)	Latency (ms)
A	0.25	94.8828	0.922	-
B	0.23	47.1904	0.925	38.81
C	0.15	23.3018	0.912	37.64

TABLE II: Results ex 2

The purpose of this exercise was to train three different models for keyword spotting on the mini speech command dataset[2] seen on lab 3. The models were required to meet some constraints regarding the output accuracy, model size and the latency measured on the edge device.

For all three models it was chosen to apply the MFCC algorithm at the preprocessing step since it offered a higher accuracy than the STFT version. For the MFCC parameters, it was noticed that by using *frame_length = 256*, *frame_step = 128* and the rest of the default values[3], the latency constraints were satisfied on versions B and C.

Regarding the models, for all scenarios it was considered a modified version of the DS-CNN. The inner blocks of the model were repeated 3 times (originaly it was done only 2 times), but the kernel size was changed from 3x3 to 2x2. It was added also a Dropout layer at the end of every block, since it was noticed a huge overfitting effect on the training step. As in the previous exercise, here it was introduced the parameter α to reduce the number of filters for each model.

In version-A, to reduce the size of the final file, it was used $\alpha = 0.25$, and a *Float16 Quantization* approach. No magnitude-based prunning was introduced in any of the models since the accuracy of the model was badly affected by its implementation.

On the contrary, for versions B and C, rather than using the *Float16 Quantization* it was used a *Weights + activations quantization* approach, declaring a generator function for providing the representative data.

From TABLE II we can observe the different hyperparameters chosen and the results obtained for the three model versions. All models have been compressed into zlib.

[2]http://storage.googleapis.com/download.tensorflow.org/data/mini-_speech_commands.zip

[3]'sampling_rate': 16000, 'frame_length': 256, 'frame_step': 128, 'mfcc': True, 'lower_frequency': 20, 'upper_frequency': 4000, 'num_mel_bins': 40, 'num_coefficients': 10

Homework 3

GROUP 17

Juan Aragon (s291466) Hadi Nejabat (s246601) Rostislav Timuta (s280238)

I. Model Registry

The purpose of this exercise was to develop a Model Registry in order to manage models for temperature and humidity on the Raspberry Pi providing some services relying on a RESTful API.

In particular, the offered services are:

- Storage of a tflite model in a local subfolder called models located at the same path of the service script.
- List the names of all the models stored in the models folder
- Measure and predict temperature and humidity values with the *DHT-11* sensor.
- Sending alarms through MQTT protocol to specific subscribed clients to monitor the accuracy of the predicted measures.

HTTP METHODS

The HTTP protocol defines a set of request methods that indicate the action to be performed on a given resource. As it pertains to this specific application,

- PUT method is used to store a tflite model with a given name. Both parameters included in the body of the request.
- GET method is used both for retrieving the list of saved models and for executing predictions with respect to the provided path (/list or /predict).

REASONS BEYOND THE CHOICE

In order to handle the /add functionality, PUT method was preferred over POST method in order to offer to clients the possibility of modifying the already stored models, avoiding to create multiple copies of models having the same name in the file system.

MQTT ALARM SYSTEM

In the initialization of the server it was also implemented an MQTT publisher connection with the public broker *test.mosquitto.org*, to publish the alarm messages regarding the accuracy of the predictions obtained at the */predict* sent following the SenML+json format.

Additionally, it was developed an MQTT client to be connected to the same broker, subscribed to the same topic where the server is sending the alarms (*/ML4IoT/Group17/H3/e1/alert*). The subscriber client is an inherited class of *DoSomething* with a modification on the *notify* method in order to display on the terminal window the topic and the alarm message received.

II. Edge-Cloud Collaborative Inference

Parameters	Accuracy(%)	MET (ms)	C. costs (MB)
sp:16k, mel_bins:16, th:0.25	**92**	61.09	**0.20**
sp:12k, mel_bins:16, th:0.25	79	43.43	**0.59**
sp:12.8k, mel_bins:32, th:0.50	85	**30.59**	2.81
sp:12.8k, mel_bins:40, th:0.25	84	**33.77**	**1.13**

TABLE I: Results ex 2

The goal of this exercise was to develop a Collaborative Interface framework for Keyword Spotting comprised of a Slow inference pipeline running on a notebook and a Fast inference pipeline on a Raspberry Pi.

SLOW PRE-PROCESSING PIPELINE

For the cloud service it was decided to leave the parameters on their original values, using a sampling frequency=16kHz, resolution=16bit, frame length=40ms, frame step=20ms, #Mel bins=40, lower frequency=20Hz, upper frequency=4kHz, #MFCCs=10.

SUCCESS CHECKER POLICY

On the edge side, it was decided to calculate the difference between the probabilities related to the two most likely labels from the predicted output. Calling the cloud service through a GET method sending the the audio under analysis after a base64 and string conversion.

FAST PRE-PROCESSING PIPELINE

Similar to the second part of the first homework of the course, the variable parameters for developing a faster pipeline were the sampling frequency and the number of Mel bins. Additionally it was also modified the threshold related to the success checker.

CONSTRAINTS & FAST PRE-PROCESSING PIPELINE

After running different simulations, trying different combinations of the variable parameters previously mentioned, it was decided to present on this paper the summary of the top-4 best options, which are described on Table I.

By using the same pipeline as in the SLOW version, the only constraint not met was related to the mean execution time being 50% higher than the limit. However, by reducing the sampling frequency we can see a drop in the accuracy of the predictions and consequently an increment in the communication costs. Surprisingly, by reducing the sampling frequency into 12.8KHz the mean execution time is drop almost by half, affecting less to the accuracy than other options.