Machine Learning for IoT

Lab 3 – Training & Deployment

Exercise 1: Keyword Spotting with Spectrogram

In Deepnote, create a Python notebook to train and evaluate a model for keyword spotting on the Mini Speech Command dataset.

1.1 Build a *tf.data* pipeline for data ingestion & pre-processing. Compute log-Mel spectrogram features with the following hyperparameters:

- Sampling rate: 16000Hz.
- STFT frame length: 40ms.
- STFT frame overlap: 50%.
- # of mel bins: 40.
- Mel lower frequency: 20Hz.
- Mel upper frequency: 4000Hz.

1.2 Develop a Convolutional Neural Network (CNN) with the architecture reported in the table below:

CNN
Conv2D(filters=128, kernel_size=[3, 3], stride=[2, 2], use_bias=False, padding='valid') **BatchNormalization**() **ReLU**() **Conv2D**(filters=128, kernel_size=[3, 3], stride=[1, 1], use_bias=False, padding='same') **BatchNormalization**() **ReLU**() **Conv2D**(filters=128, kernel_size=[3, 3], stride=[1, 1], use_bias=False, padding='same') **BatchNormalization**() **ReLU**() **GlobalAveragePooling2D**() **Dense**(units=8) **Softmax**()

1.3 Train the model using the following experimental setup:

- Select the *SparseCategoricalCrossentropy* (with *from_logits*=False) as loss function.
- Select *Adam* as optimizer and set a linear decay schedule for the learning rate.
- Select the *SparseCategoricalAccuracy* to evaluate the prediction quality.

1.4 Set the following training hyper-parameters:

- Batch size: 20.
- Initial learning rate: 0.01.
- End learning rate: 1e-5.
- # of epochs: 10.

1.5 Evaluate the *SparseCategoricalAccuracy* on the test-set.

1.6 Save the Keras model using the current timestamp as the name in a folder named *saved_models*.

1.7 Repeat the training & evaluation flow using different hyperparameters values:
- STFT frame length ∈ [10, 50] ms. Try also with power-of-two values like 8ms, 16ms, 32ms.
- STFT frame step such that overlap ∈ {0%, 25%, 50%, 75%}.
- # of mel bins ∈ [10, 40].
- Mel lower frequency ∈ [20, 80] Hz.
- Mel upper frequency ∈ [2000, 8000] Hz.

1.8 For each configuration, evaluate the *SparseCategoricalAccuracy* and report the collected results in a table. Comment the collected results:
- Which is the configuration with the highest accuracy?
- How much accuracy improved after tuning the pre-processing hyperparameters?
- Which is (are) the hyperparameter(s) that most affect(s) accuracy?

Exercise 2: Keyword Spotting with MFCCs

In Deepnote, create a modified version of the notebook of *Exercise 1* to train the CNN model with MFCC features.

2.1 Build a *tf.data* pipeline for data ingestion & pre-processing. Compute MFCC features with the following hyperparameters:
- Sampling rate: 16000Hz.
- STFT frame length: 40ms.
- STFT frame overlap: 50%.
- # of mel bins: 40.
- Mel lower frequency: 20Hz.
- Mel upper frequency: 4000Hz.
- # of MFCCs: 10.

2.2 Run the training & evaluation flow (see *Exercise 1*, 1.2 – 1.6).

2.3 Repeat the flow using different hyperparameters values:
- STFT frame length ∈ [10, 50] ms. Try also with power-of-two values like 8ms, 16ms, 32ms.
- STFT frame step such that overlap ∈ {0%, 25%, 50%, 75%}.
- # of mel bins ∈ [10, 40].
- Mel lower frequency ∈ [20, 80] Hz.
- Mel upper frequency ∈ [2000, 8000] Hz.
- # of MFCCs ∈ [10, 40].

2.4 For each configuration, evaluate the *SparseCategoricalAccuracy*. Report and comment the collected results.

2.5 Compare the results of *Exercises 1*, *2*, and *3* and answer the following questions:

- Which is the maximum accuracy achievable?
- Which is the pre-processing technique that guarantees the maximum accuracy?
- Which is the most important optimization? Feature selection (Spectrogram vs. Mel vs. MFCCs) or hyperparameters tuning?

<u>Exercise 3:</u> TFLite Conversion

In Deepnote, create a Python notebook to convert a trained model from the TensorFlow *SavedModel* format to the *TFLite* format.

- Convert the *SavedModel* using the *TFLiteConverter*.
- Save the *TFLite* model in a new folder named *tflite_models*.
- Repeat the the TFLite conversion for the models generated in *Exercises 1* and *2*.

<u>Exercise 4:</u> TFLite Testing

4.1 In Deepnote, create three Python notebooks to test the *TFLite* models generated in Exercise 4:

1) Test Inference with log-Mel Spectrogram.
2) Test Inference with MFCCs.

4.2 For each test, develop an inference pipeline that:

- Read WAV files from the MSC test-set
- Apply pre-processing
- Run inference with the *TFLite* interpreter.
- Measure the categorical accuracy.
- Measure the median latency needed for pre-processing, model prediction, and their sum (batch size=1).
- Measure the *TFLite* model size in KB (1KB = 1024 bytes).

4.3 Test the *TFLite* models and summarize the collected results in the following plots:

- Accuracy (in %) vs. Latency (in ms).
- Accuracy (in %) vs. Model size (in KB).

4.4 Comment the results:

- Which is the major contribution to inference latency? Pre-processing or model prediction?
- Which is the pre-processing configuration of the solutions belonging to the Pareto front in the Accuracy vs. Latency trade-off?
- Which is the pre-processing configuration of the solutions belonging to the Pareto front in the Accuracy vs. Memory trade-off?

<u>Note:</u> See the definition of Pareto efficiency in https://en.wikipedia.org/wiki/Pareto_efficiency.

Exercise 5: TFLite Deployment

Select one *TFLite* model from the Pareto front of *Exercise 5* and deploy it to your notebook.

5.1 Record few examples of keywords with your microphone and store the audio data on disk. Set the # of channels to 1, the resolution to *int16*, and the sampling frequency to 16kHz.

5.2 In VS Code, write a Python script that reads the recorded audio data and run inference with the *TFLite* model.

5.3 Double-check the predictions of the *TFLite* model with the collected data and comment the results:

- Is the model accurate enough with real data?
- What can be done to improve the prediction quality?