Machine Learning for IoT

Homework 2

Matteo Donadio
Student ID: s303903
Politecnico di Torino

Alessandro Maria Feri'
Student ID: s305949
Politecnico di Torino

Mattia Sabato
Student ID: s305849
Politecnico di Torino

I. Evolutionary Neural Architecture Search

A. Encoding and Mapping

The task of identifying a lightweight and reliable architecture, along with determining the optimal hyper-parameters for a fast pre-processing and effective training, does not come with a simple solution. Being these aspects greatly intertwined, we opted to conduct a Neural Architecture Search (NAS) utilizing a Genetic Algorithm (GA) to efficiently optimize them in parallel.

Operation	Shape	#Parameters
Input	(61, 8, 1)	0
Conv [2, 2, valid, L2]	(30, 4, 32)	128
BatchNorm	(30, 4, 32)	128
RElU	(30, 4, 32)	0
Conv [2, 2, same, L2]	(15, 2, 32)	4,096
BatchNorm	(15, 2, 32)	128
RElU	(15, 2, 32)	0
DW-Conv [3, 1, same, -]	(15, 2, 32)	288
BatchNorm	(15, 2, 32)	128
RElU	(15, 2, 32)	0
Global AvgPool	(32)	0
Dense	(2)	66
SoftMax	(2)	0
Total Parameters:	4,962	(19.38KB)
Trainable Parameters:	4,770	(18.63KB)
Non-trainable Parameters:	192	(768.00 Byte)

TABLE I: *Best Performing Architecture*

We define a triple $x = (\textit{Preprocessing, Architecture, Training})$ as a single vector $x \in \mathbb{N}^n$, whose components consist of integer values carrying encoded information, and we denote it as an *individual*. Each value serves as a *gene*, representing an encoded tunable hyper-parameter, whereas the ordered sequence of all genes corresponding to an individual forms the *genome*. For instance, the gene at position 1 may be an integer value representing a multiple of 4, indicating the `batch_size`, while the gene at position 12 may be a binary value indicating the presence or absence of a kernel regularization for a convolution. We then define a mapping $f(x) : \mathbb{N}^n \to \mathcal{H}$ which decodes an individual x into a general function $h = f(x)$, wrapping pre-processing, training and architecture into a unified object. The best performing architecture we have identified is presented in Table I, where we report the kernel size, stride, padding and kernel regularizer for convolutions. Table II shows the hyper-parameters associated with the architecture's remaining genome, along with their corresponding performances after training for 60 epochs.

B. Strategy

Having established the encoding, mapping, and the supported values for the genome, we adopted a (μ, λ) strategy with $\mu = 6$ and $\lambda = 10$. In each *generation*, 3 individuals are chosen using a roulette-wheel approach, where the probability is proportional to their relative *fitness*. Subsequently, a total of λ offsprings are generated by applying randomly chosen uniform/one-cut/two-cuts crossovers and one-bit mutations with a probability of 0.4. In each generation, the fitness of each individual is lazily computed as the maximum accuracy on the test set over a total of 15 epochs. Notice how a multi-objective approach, where the fitness is alternatively computed as the reciprocal of the size with a certain probability, could yield a *Pareto front* where both the accuracy and the model size are extremely optimized.

	Hyper-parameter / **KPI**	Value
Pre-processing	sampling_rate	16,000
	frame_length	0.032
	frame_step	0.016
	mel_bins	14
	lower_freq	0
	upper_freq	4,000
	MFCC	8
Training	batch_size	16
	epochs	60
	optimizer	RMSprop
	initial_lr	10e-3
	final_lr	10e-5
	schedule	PolynomicalDecay
	power_decay	2
Performance	**Accuracy (%)**	99.00
	TFLite Size (KB)	21.57
	Savings	57.04

TABLE II: *Best Hyper-parameters and Performances*

II. Results

Notably, no additional optimization was required, thanks to the thorough self-optimization carried out from the beginning and the careful design of the search spaces. The incorporation of depthwise convolutions, crucial for reducing the number of weights, was restricted to the third layer alone, since their application to other layers proved to be detrimental to accuracy. Moreover, the adoption of Mel-Frequency Cepstral Coefficients, widely utilized in speech processing, has shown to be very effective and able to enhance the accuracy despite a slowdown in the pre-processing. Finally, the use of a GA has incredibly reduced the amount of time needed to find an acceptable solution, avoiding the burden of conducting a grid search and allowing solutions to self-optimize themselves.