Extending SpArSe: Automatic Gesture Recognition Architectures for Embedded Devices

J. Borrego-Carazo, E. Buschermöhle, D. Castells-Rufas, J. Carrabina, E. Biempica, C. Müller

Abstract

SpArSe

- http://ictai2020.org/index.html : 10 Junio. Comunican 16 Agosto Rango CORE B
- http://islab.di.uminho.pt/ideal2020/ : 5 Junio. COmunican 10 Julio Rango CORE C
- https://mike2020.sigappfr.org/: 29 Junio COmunican 24 agosto NO RANGO

1 Introduction

Traditionally, optimization and tuning of machine learning model hyper-parameters was carried by means of expert knowledge, grid search or random search [1]. However, due to the long evaluation times of machine learning models, specially neural networks, it was costly and inefficient to implement the two latter strategies in terms of capital, energy and time ([2], [3]).

An alternative is to use Bayesian Optimization [4] for performing a result oriented search of the hyper-parameter space. There are two main components of a Bayesian Optimization framework [5]: the surrogate function, which models the objective function (i.e. the performance of the original model), and the acquisition function, which provides the next sampling points. Traditional surrogate models are Gaussian processes or decision trees. However, gaussian process based Bayesian optimization is not directly suited for conditional spaces. To enable this possibility, new kernels were developed [6].

Additionally and as a parallel problem, the training and use of deep neural networks was computationally demanding. In order to make neural networks more efficient and also to be able to embed them into resource constrained devices several strategies have been developed by researchers: pruning ([7], [8]), quantization [9] and efficient operations [10], among others. Along these improvements new software tools have begun to appear to port networks to resource constrained environments ([11], [12], [13], [14], [15]).

However, these procedures are focused in modifying an already trained structure and avoiding huge drops in performance; the neural architecture is not modified. The Neural Architecture Search (NAS) was a different problem: it focused on finding best performing architectures by modifying the network itself, usually by means of an evolutionary or genetic algorithm [16]. Lately, NAS algorithms have focused their attention into developing architectures which are both well performing but constrained in resource consumption and usage [17]. However, although delivering state of the art performance and efficiency, these methods still require high computational resources.

From these three problems, hyper-parameter optimization, NAS and making NNs resource efficient, stems off SpArSe [18]. The main objective is not only building state of the art performance CNNs, but also managing both the size of the model and feature maps. While controlling the size of the model allows to work with resource constrained devices with limited ROM, controlling the size of the feature maps allows to adapt to different RAM memory sizes. It is built upon four main components. First, the hyperparameter search space, which defines the possible network topologies. Second, Multi-task Bayesian Optimization is used to pursue three objectives: performance, working size and working memory. Third, pruning is used to push the reduction of used parameters. And, finally, morphisms of the original and subsequent networks are used to avoid random initializations

and costly Gaussian process fitting.

Altogether with SpArSe, there have been more efforts centered in providing NAS solutions for resource constrained environments ([19], [20], [21], [22]). However, most of these efforts center their attention in CNNs and do not include RNNs, and when they include them [23], the focus is not centered in minimizing the memory and size footprint of the architectures found. This lack, altogether with the explosion of new technologies and environments enabling or demanding gesture recognition (Virtual reality, smart cars, Kinekt sensors, wearables, among others), establishes an excellent opportunity for automatizing the building of gesture recognition networks for embedded environments.

In this work, we extend the purpose and usage of Sparse to recurrent neural networks and combinations with convolutional neural networks. We also include latency as a new objective, for tasks which have a time constraint on performance. With this we are able to develop gesture recognition solutions specially suited for embedded devices and time constrained tasks. We apply our implementation in a popular gesture classification dataset, the Corpus of Social Touch (CoST [24]). In addition, we modify the search procedure introducing a low cost fidelity by running less epochs at the beginning and increasing them as the program advances. Hence, this forces the search algorithm to be more exploratory at the beginning and employ more time with Pareto solutions by the end of the procedure. Also, we include an analysis of the kernel employed to account for conditional spaces [6], and another analysis for the importance of latency inclusion as an objective.

2 Related work

Regarding NAS algorithms, they are based on three components [25]: search space, search strategy and performance measure.

The search space defines the allowed or accessible components for building the network. In most cases, the search restricts to CNNs and hence is focused primarily on images. However, modifications have already been made to account for other types of inputs, such as in the case of [26].

Regarding the search strategy, there are three main approaches in the literature: evolutionary algorithms (EA) ([27], [28]), bayesian optimization [29] and reinforcement learning (RL) [30].

All those previously mentioned methods focus primary in one metric performance, usually the performance on the validation set according to the task metric. Lately, modifications of the following types of search strategies or new developments have been carried out. In [17] an evolutionary based NAS algorithm is implemented taking into account resource and outcome constraints. In [31], authors use a combination of RL and EA and optimize to find suitable architectures both on the error rate and the latency (or FLOPS during inference). In [32] they shrink and expand a established network in order to improve accuracy but also account for latency.

However, all these procedures are not oriented specifically for heavily constrained devices, such as microcontrollers, and the architectures developed cannot be deployed in such platforms. Hence, efforts have been made to develop procedures to deliver constrained networks. One of such techniques is to set off from a predefined network and conduct an optimization procedure for delivering a reduced but performing version ([19], [20]). Other techniques, such as in SpArSe [18], define a constrained search space and define different objectives both for ROM, RAM and accuracy.

3 SpArSe Modifications

SpArSe is built upon two major components: the search space and the search strategy or procedure. As performance, it has a three objective target: accuracy, Model Size and RAM memory.

The search space, which is detailed in the original implementation, consists in sequential convolutional layers organized in blocks altogether with the possibility of branching through fully connected layers, as detailed in Figure 1. It also includes pruning parameters. In our case we have deleted fully connected layers since they add more complexity and certainly make the network heavier. Although, as detailed in other studies [33], this could help discovering new structures inside the network, in this case the internal structure would have to depend on this fully connected branching, which adds more weight without explicit improvement. Regarding the pruning, the original paper uses structured and

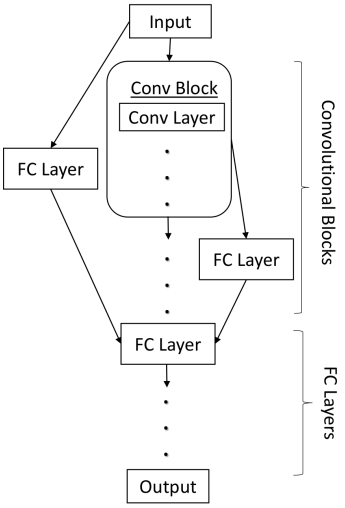

Figure 1: Original search Space made primarily with Convolution Blocks consisiting of several layers each, fully connected layers as branches and as final step before classification. Our modified space is exactly the same with the exception of branches.

structured pruning, while we have used only unstructured pruning.

The search procedure in the original implementation is based in three stages: in the first, new networks are sampled randomly or morphed from a previous one. In the second stage, the morphs are restricted to pruning. In the third step, the reference configurations are restricted optimal Pareto Points. The sampling process of architectures is through Thompson Sampling (TS). In general we conserve the three stage procedure, but with changes with regards sampling and model training. In the first stage, we follow a low fidelity scheme, as detailed in [34], consisting in training for only one epoch and sample models following a Sobol random procedure. In this case there are no morphisms, since we want to explore the space and obtain architectural performance information. In the second stage, we continue without morphisms but the sampling procedure is led by a Gaussian process (GP) with [6] as kernel and with Monte Carlo based Expected Improvement (EI) as acquisition function (AF). Thus, we rely in the AF for

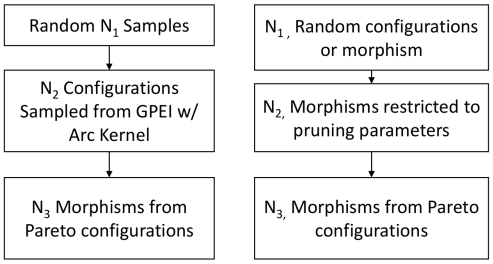

Figure 2: Left: Search procedure for our version of SpArSe. Right: original search procedure. In the original procedure, each configuration is either sampled randomly with probability ρ, or drawn as a morphism with probability $1 - \rho$. As far as the authors are aware, the GP is only in charge of fitting the data and enable the acquisition function to choose the best point from the sampled configurations.

the proposal of new points, while in the original paper this was relegated to morphisms or TS. The reason is the evidence of greedy exploration by TS ([35], [36]), while EI, among others, balances better the exploration/exploitation trade-off. We also increase the epochs through which the models are trained. In the final stage, we base new sampling points only in morphisms directed through the AF and the training epochs are extended to a normal training procedure. An illustration of our search procedure is detailed at Figure 2.

As an additional change, we have changed the RAM computation to the maximum value for the sum of parameters of input, output and weight along the network. That is, $MaxRAM = \|y_l\|_0 + \|x_l\|_0 + \|w_l\|_0$, where l is the layer index, y is the output of the operation, x is the input and w stands for the weight (including bias). In the original paper only input and output, or input and weight were included in working memory. The authors of the current work think that the three elements suppose a better approximation to the RAM computation in a real scenario.

4 Experimental Design

First, we compare the original implementation with our implementation after modifications. The com-

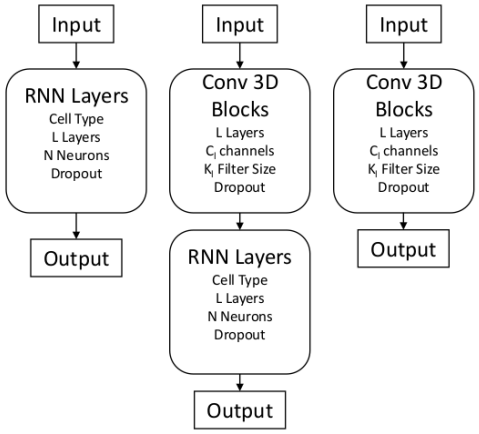

Figure 3: Different search space possibilities for the search space expansion of SpArSe. Overall they form a unique search space, however, there is the possibility to sample from each of them depending on the task. Left: a search space only composed of recurrent neural networks.Middle: a combination of 3D convolutions and recurrent networks. Right: a search space only for 3D convolutions.

parison is made with the MNIST and CIFAR2 [37] datasets, with same preprocessing and splits as in the original paper.

4.1 Sparse Extension and CoST Dataset

The second step is to extend the SpArSe to gesture classification with temporal structures For this purpose, latency is included as another objective. We specify the latency of the network with the number of floating operations needed to predict a sample input. Then, we let the user specify the frequency or speed of its platform to finally obtain a latency measure. To be able to capture temporal dependencies, RNNs, GRU and LSTM, are also incorporated into the search space, as detailed in Figure 3.

In order to test out the RNN and latency inclusion, and the overall suitability for gesture classification, the next task is to trial the extended version of SpArSe with a suited dataset: the Corpus of Social Touch [24]. It consists of sequences coming from a pressure sensor arranged in a $8x8$ grid.

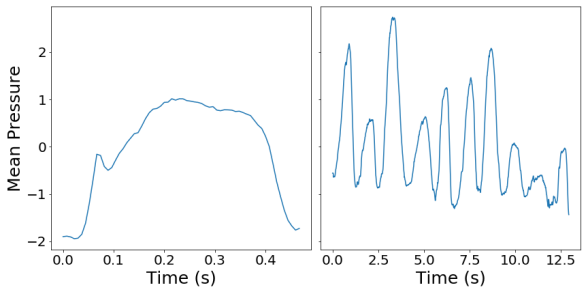

Figure 4: Example of CoST samples. Left: grab gesture. Right: tickle gesture. Values of rpessure are standardized by mean subtraction and standard deviation division.

Each of the values of the sensor is between 0 and 1023 and the sampling frequency is 135 Hz. Hence, each sequence has dimension $Nx8x8$, where N is its length. The lengths of the dataset vary from 10 to 1747 samples. The dataset was recorded with the participation of 31 subjects and comprises 14 classes: grab, hit, massage, pat, pinch, poke, press, rub, scratch, slap, stroke, squeeze, tap, and, tickle. In Figure 4, a sample from the dataset averaged over channels is presented.

4.1.1 Dataset Split and Preparation

Regarding the dataset split there are inconsistencies among the papers regarding the data division. For example, [38] and [39] divide the dataset randomly into 21 subjects for the training set and 10 for the test set. The authors of the present study have not been able to find the specific ID of this partition. [40] seems to follow the same structure since the authors participated in the original competition. However, in [41] and [42], they use leave-one-subject-out cross validation for showing main results. In the present study, the dataset has been divided in two different manners. For the NAS procedure, the dataset has been split randomly between 20 subjects for training, 5 for validation and 6 for test. After finding the best performing network, we apply leave one out subject cross validation to obtain the final results.

As stated in the previous section, the lengths of the sequences are quite large. These induces problems both for RNN and CNN based architectures. For CNNs, saving and using whole sequences would

be a problem for embedded deployment since we would be storing a really large array. For RNNS, in the training phase, problems such as vanishing gradients might occur, while in the prediction phase, there is a dichotomy: if the RNN cell prediction is faster than the sampling rate there is no problem for real time prediction. However, if it is slower you are forced to store all the data in RAM, which can be impossible for most microcontroller capabilities (a sequence of, for example, 500 samples, equals 125 KB, in 32 bit float numbers). Hence, we devise two strategies: first, partition the sequences and predict on the sub-sequences, as done in previous studies ([41], [38]), and second, pick samples from inside the sequence. With the first method, the whole sequence is not processed by the model, while in the second, although intermediate information is lost, the whole sequence is perceived. This is important since for some sequences the gesture is recorded at the end [40]. Both the number of chops or the sampling are included in the search procedure.

The only preprocessing applied to the dataset is standardization by mean subtraction and standard deviation division.

4.2 Kernel Analysis and Latency Proxy

The Arc Kernel [6] is defined as follows. Given an N dimensional space, X, with dimension bounds $[u_i, l_i] \in \mathbb{R}$, and some delta functions $\delta_i : X_i \rightarrow \{True, False\}$, where, $i \in 1, ..., D$, an embedding $g_i : X_i \rightarrow \mathbb{R}^2$ is defined as:

$$\begin{array}{ll} [0,0]^T & \text{if } \delta_i(\mathbf{x}) = false \\ w_i[\sin \pi\rho_i \frac{x_i}{u_i - l_i}, \cos \pi\rho_i \frac{x_i}{u_i - l_i}]^T & \text{otherwise} \end{array}$$

where $\omega_i \in \mathbb{R}^+$ and $\rho_i \in [0, 1]$. The, with a covariance function, for example the Radial Basis Function (RBF), we can define the Arc Kernel.

$$K(x, x) = \prod_i \sigma^2 \exp(-\frac{1}{2} d_i(\mathbf{x}, \mathbf{x}'))^2 \qquad (1)$$

where σ is a scale parameters and $d_i = \|g_i(\mathbf{x}) - g_i(\mathbf{x}')\|$

We can see that the capabilities for capturing conditionalities reside in the embeddding and the delta functions. These functions annihilate the dimensions which are not relevant in each case. For example, if a network can have 6 layers but the current samping only uses 4 it is obvious that the number of neurons in layers 5 and 6 are not useful. Delta functions control the activation of the relevant dimensions. As a side analysis of this work, we assay the performance of the Arc Kernel capturing conditional dependencies. For this purpose, we use a convolutional neural network with up to five layers with variable number of fiters targeted towards the Cifar10 dataset.

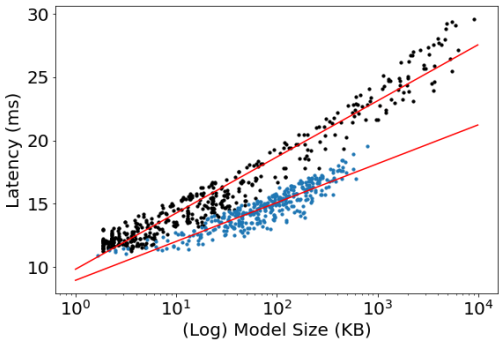

Figure 5: Latency against Model Size for RNN Search Space (black) and 2DCNN (blue), along regression lines (red) for each one. $R^2_{RNN} = 0.9490$, $R^2_{2DCNN} = 0.8247$. Model Size axis is logarithmic.

Finally, and also as an added study, we asses the utility of including latency as another objective. This test stems from the idea of model size being a proxy for latency [19]. However, if this should be the case, a relationship between size and latency should be established to avoid including it as an objective but allowing the user to restrict the networks sampled.

5 Results

The network builder and trainer has been developed using Pytorch, as well as the pruning and quantization procedures. The search algorithm is mainly build with Ax [43], although some parts that needed more detail such as the Kernel have been implemented through BoTorch [44] and GPyTorch [45]. Each of the models and search procedures have been carried out with a single RTX 2070 Ti.

Framework	CIFAR-10 Binary			MNIST		
	Accuracy	Size	MaxRAM	Accuracy	Size	MaxRAM
DeepMaker [19]	-	-	-	95.9	507.81	-
SpArSe [18]	73.84	0.78	1.28	96.49	1.44	1.33
SpArSe Mod	71.74	5.32	6.29	96.85	2.67	5.35

Table 1: Results for the original version of SpArSe and our modified version. Accuracy is in % and Size corresponds to the model weight in KB taking into account only weights and not code. The maximum RAM is also in KB and corresponds, in our case, to the computation specified in Section 3.

5.1 SpArSe

In Table 1, results for the modified version can be compared with the original implementation. Results for our version have similar accuracy performance and a little higher memory requirements. This could be both due to the difference in the computation procedure and the maximum RAM calculation method. Another possibility is the architecture change. The reason might be, following the findings in ([33], [20]), the reduced search space and the impossibility of the pruning strategy to prune further compared to the original search space. That is, the bigger the search space the greater the probability of finding a well performing network with a smaller size and feature maps.

A point that was not noted in the original paper is the utility of the Pareto frontier as final result. With the frontier we are able to choose among a range of different combinations suiting our direct hardware constraints. For example, and as illustrated in 6, we could choose the heavier network in the CIFAR2, with 74.55 % accuracy, 19.31 KB in size and 14.39 KB in RAM. Or, if we were really constrained for those resources, we could choose the other extreme with 57.83 % accuracy, 0.12 KB in size and 1.44 KB in RAM.

5.2 CoST

First, we have compared the results between splitting or insampling the data for a pure RNN model, as estabished in Section 4.1.1. Results are illustrated in Table 2 and compared to other implementations. As seen, cutting the signal delivers better results than sampling points from it, probably due to loss of signal information when sampling. As seen the RNN method provides state of the art results for all metrics, making SpArSe solutions directly usable in resource constrained environments while obtaining the maximum performance for the configured search space.

Comments on 2D and 3DCNN results

5.3 Size as proxy for latency

In the methodology, Section 4.2, the doubt was raised about size being a proxy for latency. In Figure 5, latency is plotted against model size in logarithmic scale, for both RNN and 2DCNN spaces. It seems to be a linear relationship after this transform, that is, $L = A \cdot \log MS + B$, with regression coefficient $R^2 = 0.9493$. However, several concerns should be raised. Models developed are sequential and not complex and branchlike graphs, hence more observations for more complex inference procedures should be obtained. Moreover, search spaces employed are by construction small compared to other tasks not related to resource constrained devices. Hence, observations for bigger networks should be obtained. A final point is that this relationship is space dependent. Hence, latency has to be in any case measured, as points out results for the 2DCNN space, which has another slope. For this reason, it is worth including it as an objective if the task is latency limited.

5.4 Arc Kernel

In Figure 6, one can observe the progress of the seach procedure with respect to accuracy and model size. However, the fundamental question is if the progress made is due to the capability of the GP and Kernel to model the conditional nature of the network or only due to the lower fidelities, that is, because we increase epochs with time.

To clarify this, we have compared the Matern Kernel and the Arc Kernel to a simple problem: a neural network with one to five convolutional lay-

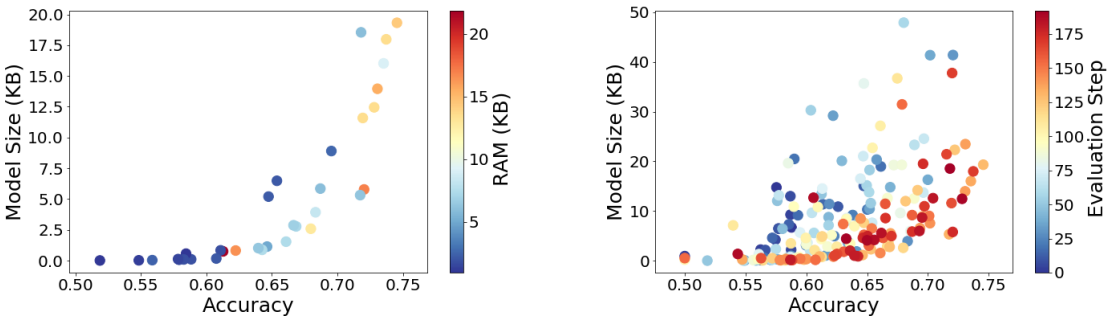

Figure 6: Left: Pareto frontier for our implementation of Sparse and Cifar 10 Binary. Notice the different possibilities to choose among regarding RAM, size and performance. Each point corresponds to a specific configuration of the search space. Right: full search procedure for CIFAR-10 Binary with the modified SpArSe. The color degradation shows the progress of the search.

Model	Model Type	Accuracy	Model Size (KB)	Max RAM (KB)	Latency (ms)
Albawi *et al.* [41]	CNN	63.71	3556.28*	2105.34*	-
Ta *et al.* [40]	Random Forest	60.8	-	-	-
Hughes *et al.* [38]	CNN + RNN	52.86	73.05	25.25	-
Ours	RNN Cut	**65.12**	**15.77**	**13.86**	15.68
Ours	RNN Insample	58.38	32.20	31.89	15.29
Ours	2DCNN	55.68	17.91	16.31	**13.17**

Table 2: Results for the CoST dataset with hold one subject out cross validation, as in [41] and [39]. * indicates that it has been estimated from the netowrk details in original .paper

ers and different neurons for each layer targeted to the CIFAR 10 dataset. In Figure 7 we can observe the performance of both the Matern and Arc kernels. Arc Kernel seems to be more noisy although final performance is better than Matern due to this more explorative nature. With regards the space complexity capture, Arc Kernel ends assuming the conditional structure of the space, due to the latter QQ plot fitting. However, it shows difficulties during the process, as observed by the vertical lines.

One important point, in the case of the Arc Kernel, is the need to define the delta functions, $\delta_i(\mathbf{x})$ with regards the original conditional space, as defined in Section 4.2. An improvement could be to learn these conditionalities by including them in the search space. However, this could overload the GPs and care should be taken.

6 Conclusions and further work

In this study we have modified and extended the procedure and architecture introduced in [18], obtaining similar results. These results act as a validation of the general NAS procedure since the modifications were based in the addition of lower fidelities and modification of sampling methodologies. By adapting the search space, including 3DCNN and RNNS, we have been able to develop state of the art methods for a Gesture Recognition task. An analysis of the Arc Kernel [6] has also been presented, showcasing its slightly better performance to other available kernels and proposing the inclusion. Finally, regarding using size as a proxy for latency, although it could be used as such, the problem resides in the different relationship in every kind of network topology, which forces to learn it to be able to use it. Hence, we conclude that the measure of latency is unavoidable.

References

[1] J. Bergstra and Y. Bengio, "Random Search For HyperParameter Optimization," vol. 13, pp. 281–305, 2012.

[2] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," pp. 1–16, 2016.

[3] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 8697–8710, 2018.

[4] J. Mockus, V. Tiesis, and A. Zilinskas, "The application of Bayesian methods for seeking the extremum," *Towar. Glob. Optim.*, vol. 2, no. September 2014, pp. 117–129, 1978.

[5] P. I. Frazier, "A Tutorial on Bayesian Optimization," no. Section 5, pp. 1–22, 2018.

[6] K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, and M. A. Osborne, "Raiders of the Lost Architecture: Kernels for Bayesian Optimization in Conditional Parameter Spaces," pp. 1–6, 2014.

[7] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," pp. 1–14, 2015.

[8] Y. L. Cun, J. S. Denker, and S. A. Solla, "Advances in neural information processing systems 2," ch. Optimal Brain Damage, pp. 598–605, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990.

[9] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," 2017.

[10] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2017.

[11] "Tensorflow Lite." `https://www.tensorflow.org/lite`. Accessed: 2019-07-19.

[12] N. Rotem, J. Fix, S. Abdulrasool, G. Catron, S. Deng, R. Dzhabarov, N. Gibson, J. Hegeman, M. Lele, R. Levenstein, J. Montgomery,

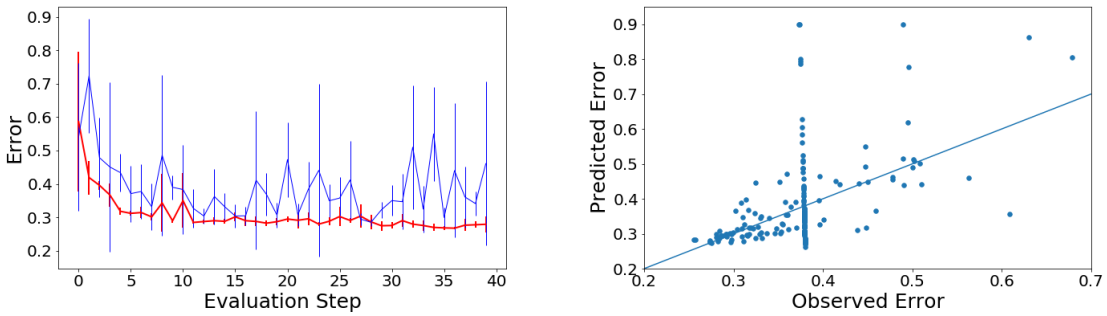

Figure 7: Left: evolution of performance of a CNN with variable layers in a bayesian optimization procedure with a GP based on Arc (blue) and Matern (red). Vertical bars indicate standard deviation of the different evaluations inside the same step. Right: Prediction values of the GP for each evaluation step and the real observed values (by evaluating the network)

B. Maher, S. Nadathur, J. Olesen, J. Park, A. Rakhov, M. Smelyanskiy, and M. Wang, "Glow: Graph Lowering Compiler Techniques for Neural Networks," 2018.

[13] "ARM NN." `https://developer.arm.com/ip-products/processors/machine-learning/arm-nn`. Accessed: 2019-07-19.

[14] "NVIDIA tensor rt." `https://developer.nvidia.com/tensorrt`. Accessed: 2019-07-19.

[15] "uTensor." `https://github.com/uTensor/uTensor`. Accessed: 2020-04-20.

[16] S. K.O. and M. R., "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[17] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution," pp. 1–23, 2018.

[18] I. Fedorov, R. P. Adams, M. Mattina, and P. N. Whatmough, "SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers," pp. 1–26, 2019.

[19] M. Loni, S. Sinaei, A. Zoljodi, M. Daneshtalab, and M. Sjödin, "DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems," *Microprocess. Microsyst.*, vol. 73, p. 102989, 2020.

[20] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-All: Train One Network and Specialize it for Efficient Deployment on Diverse Hardware Platforms," pp. 1–13, 2019.

[21] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," *MECOMM 2018 - Proc. 2018 Work. Mob. Edge Commun. Part SIGCOMM 2018*, pp. 31–36, 2018.

[22] B. Lu, J. Yang, L. Y. Chen, and S. Ren, "Automating deep neural network model selection for edge inference," *Proc. - 2019 IEEE 1st Int. Conf. Cogn. Mach. Intell. CogMI 2019*, pp. 184–193, 2019.

[23] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient Neural Architecture Search via parameter Sharing," *35th Int. Conf. Mach. Learn. ICML 2018*, vol. 9, pp. 6522–6531, 2018.

[24] M. M. Jung, R. Poppe, M. Poel, and D. K. Heylen, "Touching the void - introducing CoST: Corpus of social touch," *ICMI 2014 - Proc. 2014 Int. Conf. Multimodal Interact.*, pp. 120–127, 2014.

[25] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search," vol. 20, pp. 63–77, 2019.

[26] Y. Wang, Y. Yang, Y. Chen, J. Bai, C. Zhang, G. Su, X. Kou, Y. Tong, M. Yang, and L. Zhou, "TextNAS: A Neural Architecture Search Space tailored for Text Representation," 2019.

[27] L. Xie and A. Yuille, "Genetic CNN," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2017-Octob, pp. 1388–1397, 2017.

[28] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[29] H. Zhou, M. Yang, J. Wang, and W. Pan, "BayesNAS: A Bayesian Approach for Neural Architecture Search," 2019.

[30] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing Neural Network Architectures using Reinforcement Learning," pp. 1–18, 2016.

[31] X. Chu, B. Zhang, R. Xu, and H. Ma, "Multi-Objective Reinforced Evolution in Mobile Neural Architecture Search," 2019.

[32] A. Gordon, E. Eban, O. Nachum, B. Chen, and T.-J. Yang, "FluidNets: Fast & Simple Resource-Constrained Structure Learning of Deep Networks," no. 1, pp. 1586–1595, 2017.

[33] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *7th Int. Conf. Learn. Represent. ICLR 2019*, pp. 1–42, 2019.

[34] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search," vol. 20, pp. 63–77, 2019.

[35] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[36] B. Shahriari, Z. Wang, M. W. Hoffman, A. Bouchard-Côté, and N. de Freitas, "An Entropy Search Portfolio for Bayesian Optimization," pp. 1–10, 2014.

[37] C. Jose, P. Goyal, P. Aggrwal, and M. Varma, "Local deep kernel learning for efficient nonlinear SVM prediction," *30th Int. Conf. Mach. Learn. ICML 2013*, vol. 28, no. PART 2, pp. 1523–1531, 2013.

[38] D. Hughes, A. Krauthammer, and N. Correli, "Recognizing social touch gestures using recurrent and convolutional neural networks," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 2315–2321, 2017.

[39] M. M. Jung, X. L. Cang, M. Poel, and K. E. Maclean, "Touch challenge'15: Recognizing social touch gestures," *ICMI 2015 - Proc. 2015 ACM Int. Conf. Multimodal Interact.*, pp. 387–390, 2015.

[40] V. C. Ta, W. Johal, M. Portaz, E. Castelli, and D. Vaufreydaz, "The grenoble system for the social touch challenge at ICMI 2015," *ICMI 2015 - Proc. 2015 ACM Int. Conf. Multimodal Interact.*, no. November, pp. 391–398, 2015.

[41] S. Albawi, O. Bayat, S. Al-Azawi, and O. N. Ucan, "Social touch gesture recognition using convolutional neural network," *Comput. Intell. Neurosci.*, vol. 2018, 2018.

[42] M. M. Jung, M. Poel, R. Poppe, and D. K. Heylen, "Automatic recognition of touch gestures in the corpus of social touch," *J. Multimodal User Interfaces*, vol. 11, no. 1, pp. 81–96, 2017.

[43] E. Bakshy, L. Dworkin, B. Karrer, K. Kashin, B. Letham, A. Murthy, and S. Singh, "AE: A domain-agnostic platform for adaptive experimentation," *Conf. Neural Inf. Process. Syst.*, pp. 1–8, 2018.

[44] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy, "BoTorch: Programmable Bayesian Optimization in PyTorch," 2019.

[45] J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson, "Gpytorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration," *Adv. Neural Inf. Process. Syst.*, vol. 2018-Decem, no. NeurIPS, pp. 7576–7586, 2018.