

Unsupervised Embedded Gesture Recognition Based on Multi-objective NAS and Capacitive Sensing

^{1, 2, *} Juan BORREGO-CARAZO, ¹ David CASTELLS-RUFAS,
² Ernesto BIEMPICA and ¹ Jordi CARRABINA

¹ Universitat Autònoma de Barcelona, C. de les Sitges s/n, Bellaterra, 08193, Spain

² R+D, Kostal Eléctrica S.A., Notari Jesús Led 10, 08181, Senmenat, Spain

¹ Tel.: 93 581 3358

E-mail: juan.borrego@uab.cat

Received: 5 October 2020 / Accepted: 26 November 2020 / Published: 28 February 2021

Abstract: Gesture recognition has become pervasive in many interactive environments. Recognition based on Neural Networks often reaches higher recognition rates than competing methods at a cost of a higher computational complexity that becomes very challenging in low resource computing platforms such as microcontrollers. New optimization methodologies, such as quantization and Neural Architecture Search are steps forward for the development of embeddable networks. In addition, as neural networks are commonly used in a supervised fashion, labeling tends to include bias in the model. Unsupervised methods allow for performing tasks as classification without depending on labeling. In this work, we present an embedded and unsupervised gesture recognition system, composed of a neural network autoencoder and K-Means clustering algorithm and optimized through a state-of-the-art multi-objective NAS. The present method allows for a method to develop, deploy and perform unsupervised classification in low resource embedded devices.

Keywords: Unsupervised learning, Neural networks, Neural architecture search, Capacitive sensing, Embedded electronics.

1. Introduction

Hand gestures are an efficient way of communicating simple concepts. During the last two decades, its usage in Human-Machine Interaction (HMI) has become pervasive thanks to the proliferation of low-cost depth sensors based on structured light, time of flight, and active stereo matching [1].

Depth sensors based on the analysis of light parameters can achieve a high-resolution accuracy, but since they are based on a single point of view that must be distant to the object to recognize, they are mostly used for mid-air gesture recognition [2]. Moreover, they are sensible to illumination conditions.

On the other hand, capacitive sensing stands out for its low power, highly sensitive but reliable solution for gesture recognition where close contact is required [3].

Neural Networks (NNs) have achieved outstanding performance in tasks such as image classification or speech translation. In the supervised case, however, their usage requires labeling, which can induce, among other factors, bias in the model and undermine its real-world use [4]. To partially solve this issue, unsupervised methods focus only on the information proportioned by the data to perform the task and avoid annotation and labeling, both error-prone activities which could induce noise and bias. Nevertheless, the bias or noise are not completely removed since the data itself could still be biased and noisy. In the end,

unsupervised methods do not only liberate from the task and effects of labeling but also allow for more freedom in the learning of patterns, by not being constrained to a specific purpose task. Hence, they enable the possibility of pattern discovery and identification outside the constraints of the specific task purpose.

As an added problem, neural networks have required a certain level of computing resources preventing their deployment in low resource platforms like microcontrollers. Lately, this problem has been addressed through optimization techniques like quantization and pruning. However, with the use of

such techniques, the network still has to be built manually. In the case of low resource platforms, this fact imposes severe difficulties to develop NNs which are well-performing and compliant with the different memory and latency requirements of the deployment platform. To address this problem, Neural Architecture Search (NAS) [5] was developed, allowing for the automatic building of neural networks. Later, this method was extended to a multi-objective setting [6], to account for the requirements of low resource platforms and deliver functional but also minimal neural networks.

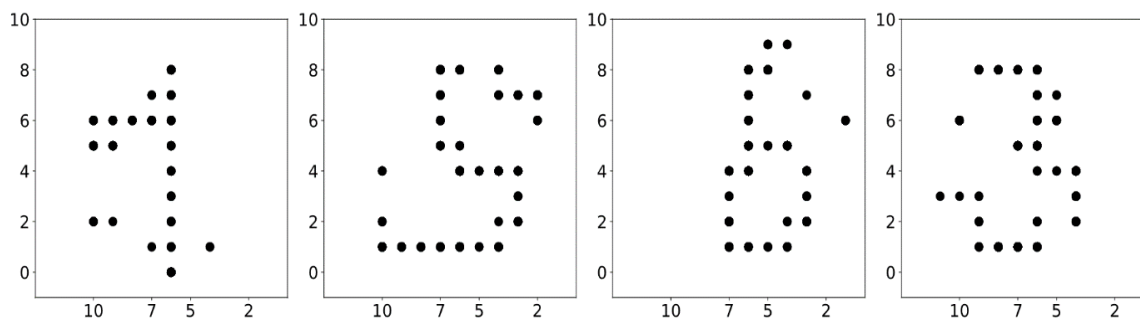


Fig. 1. Examples of numbers drawn at the sensitive surface. From left to right: a one, a five, a six and a three. The black points correspond to the electrode pairs through which the finger has passed. In an image configuration they correspond to 1 s while the background is set to 0.

In this work, we present an auto-encoder neural network that, in junction with K-Means (KM) and optimized through NAS, can perform unsupervised classification in a low resource microcontroller for recognizing gestures. To check the validity of such unsupervised learning and classification, we check our model results in a supervised evaluation setting, which proves the validity of the approach and the possibility of not needing labeled data when using it. Although there have been similar works with other models [7] or with other topics [8], to the best of our knowledge, this is the first work that presents a fully embedded and unsupervised solution for gesture recognition using neural networks and capacitive sensing.

2. Previous Work

Capacitive sensing has been used for multiple application fields including music [9], security [10], or entertainment. It is characterized by offering a low power, only electronic, and cheap alternative to other sensing methods. In the case of gesture recognition, capacitive sensing has also been used extensively; although mostly in touchable surfaces [11], there have been applications based on wearables [12] and on 3D and extended ranges [13]. Different applications require different sensitivities and ranges, capacitive sensing allows for a variety of design parametrizations that change these features. In the present case, we refer to mutual capacitance sensing according to

the classification from [3], which allows for a short-range interaction.

Several approaches can be used to classify the sensed signals into recognized gestures. Classic machine-learning methodologies were based on a two-step procedure: first, extracting the relevant features from the input signals; and, second, the classification of such features into the corresponding gestures [14]. Humans associate gestures with meanings. Gestures can be classified as being either static or dynamic. In a static gesture, a certain body part pose that is maintained for a period of time is associated with a meaning (e.g. thumbs-up is associated with OK). In a dynamic gesture, the body part movement is essential to the meaning (e.g. finger-wag is associated with NO). Common algorithms to classify static gestures are support vector machines (SVM) [15], k-nearest neighbors, and random forests, among others. Regarding dynamic gestures, Hidden Markov Models (HMM) [16] or Dynamic Time Warping models are commonly used due to the intrinsic temporal component. In order to feed the data in an appropriate form to the latter algorithms, methods such as Fisher Vectors, PCA, or other feature representations were used.

Nevertheless, with the advent and success, albeit with a higher computational cost, of deep learning this pipeline has changed. In the deep learning setting, the feature extraction step has been deleted in most cases and all the process has been substituted by a neural network. In the case of static classification, methods

based on CNN [17] stand out, differentiating between them with regards to the information they use: camera-based ([18], [19]), or signals ([20], [21]). In the latter case, 1D convolutions stand out as a novel application. In the case of dynamic classification, recurrent models are commonly used ([22], [23]), allowing for a continuous and online gesture classification. Lately, however, and in the case of videos, 3D convolutions have also been used both for continuous and static classification [24].

Deep learning methods, however, usually entail high computational costs that unable their deployment into resource-constrained platforms. To alleviate this problem several methods such as quantization [25], pruning [26], distillation, or NAS [27] have been developed on the software side, while hardware accelerators and deployment frameworks [28] have also implemented in order to speed up inference and allow ease of deployment.

Since then, applications for deploying neural in resource-constrained environments, and more specifically, microcontrollers have flourished. From keyword spotting [29], image classification, and object detection, new applications are being ported to low power and edge devices. Gesture recognition has not been an exception, with new applications and methods appearing continuously; for example, using radar signals [30] or with wearables [31]. In the case of capacitive sensing, most of the applications that use embedded deep learning are oriented to touch detection, prediction, and related tasks in smartphone surfaces [32], while works developed in the microcontroller environment are less frequent.

In most cases, the development of such embedded gesture recognition applications with neural networks requires the annotation of data [33]. There are few cases, where unsupervised methods have been used for gesture recognition with neural networks have been used. In [7] the authors employ several unsupervised methods, such as K-means, Self Organizing Maps or Hierarchical Clustering for classifying gestures sensed through a gyroscope and a geomagnetic sensor. However, they do not embed the models in the microcontroller. In [8], the authors use an approach similar to our proposal based on an autoencoder plus K-Means to classify images. However, they do not perform any kind of optimization in order to allow the model to be run in resource-constrained devices, reason why our methods stand out.

Precisely and, to the best of our knowledge, this is the first work using unsupervised gesture recognition in a microcontroller using neural networks and capacitive sensing.

3. Sensing Method and Data

3.1. Capacitive Sensing Platform

As depicted in Fig. 2, the sensing platform consists of a surface touch module with three elements: the

plastic cover, the capacitive foil, and the embedded board. The touch interface is the upper part of the plastic cover, which is immediately after the capacitive foil and glued to it. The embedded board is in the lower part and connected to the capacitive foil. The sensing component, which is the capacitive foil, consists of a central sensitive surface and 24 electrodes which run through it: 9 horizontally and 13 vertically, plus a global shield electrode.

The sensing mechanism is based on the transference of charge between two capacitors: integration and measurement condensers. More details of the sensing methodology can be found in [23].

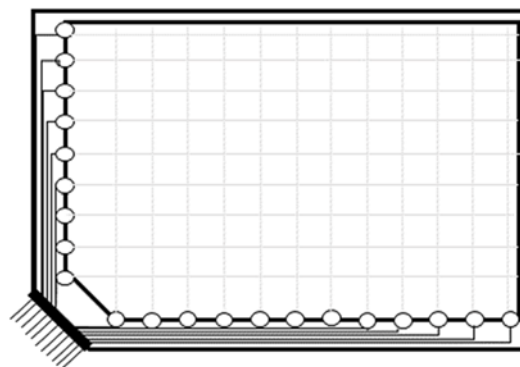


Fig. 2. Illustration of the capacitive foil. Each of the dots represents an electrode, emitting each one a distinctive measurement of its row/column in the surface. The foil is pasted underneath the plastic cover and connected to the microcontroller through the lower-left pins.

The final result of a measurement is a set of voltages sensed at the measurement capacitor, one for each electrode. Each voltage is then converted through a 10-bit ADC to obtain a final raw value for each electrode at a predefined sampling rate (in the present case, 500 Hz).

3.2. Data Collection and Preparation

The user enters the gestures by touching the sensing surface and, without withdrawing the finger, draws the gesture on it. We detect the touch by the method described in [34], and then we begin to collect the raw values of the electrodes as described in the previous section. By examining which electrodes, horizontally and vertically, have the highest value we can determine in which spatial coordinates the finger is placed. When the user withdraws the finger, we stop collecting data. It has to be noted, that the gestures described do not contain segmentation and thus we do not assess that problem. That is, we consider a gesture all the signals from a touch to a withdrawal, and force users to perform the whole gesture in such a manner.

By integrating all the coordinates through which the finger has passed we construct a greyscale image of 9×13 pixels of the figure drawn. That is, we mark the points through which the user has passed with a 1,

while the rest of the untouched points remain as background with a value of 0. Hence, the orientation of the strokes is deleted and only the final form of the gesture drawn is recorded. That is we produce a static version of the gesture. To ease the deployment in a microcontroller and due to deployment framework limitations, we end up squaring the image to 13×13 by padding it. An example of such a method can be seen in Fig. 1, where different numbers drawn on the sensitive surface can be visualized as the points through which the finger has passed.

To collect the dataset, numbers from 1 to 9 were drawn a total of 100 times per number on the sensing surface. The dataset is divided in a stratified manner in training, validation, and test, with 720, 90 and 90 samples respectively.

4. Models and NAS Framework

4.1. Models

We consider two auto-encoder (AE) model types, that is, networks that are trained to replicate the input. The first a Convolutional Auto-encoder (CAE), and the second a Dense Auto-encoder (DAE), made only of fully connected layers.

In general, the architecture consists of two differentiated parts: the first, the encoder, is in charge of extracting the information and compress it in a central vector, and the second, reconstructing the input. In the CAE case, compression is performed by a succession of convolutional or pooling layers followed by ReLU activations, while the input reconstruction is performed by deconvolution layers and final upsampling. In the DAE case, all the layers are fully connected followed by ReLU activations, except for the final layer which reconstructs the input by reordering the output of the network. In both cases, there is a central fully connected layer that produces the latent feature vector. An important note is that to reconstruct the input, as it is made of 0s and 1s, we have used a Binary Cross-entropy Loss for training the AEs.

To provide unsupervised classification, a KM algorithm is in charge of grouping the compressed feature vectors into clusters. The choice of KM is central since it needs the number of clusters to be specified. In our case, this is precisely the number of classes.

4.2. Training and Model Optimization

To train the whole unsupervised method we divide the training into three separate steps. First, we train the AE to replicate the input. The loss used is an L2 norm based reconstruction loss. The networks are trained for

a maximum of 50 epochs and the model with the best validation loss is used as the final model; the number of epochs is chosen based upon observation of previous training.

Second, once it has stopped training, we forward the entire training dataset through it and collect the latent feature vector for each input. The latent vector corresponds to the central chosen vector, which can be seen in Fig. 3. In the CAE case, it has 12 elements, while in the DAE it has 117. Hence, the feature extraction through the autoencoder acts as a form of data compression algorithm with an effective compression ratio close to 10.

Third, we train the KM algorithm with all the collected feature vectors. That is, we have a feature vector per image and each of those vectors correspond to the central vector obtained by forwarding the corresponding image through the trained autoencoder. For example, in the CAE case, we have 720 vectors of size 12. As we are classifying 9 different numbers (from digit 1 to 9, excluding 0), we establish 9 clusters for the K-Means algorithms. K-Means algorithm is initialized following the k-means ++ strategy [35] of similarly distanced cluster centers and the distance used is the L-2 norm¹. Finally, to obtain test or validation results, we forward the test or validation sets through the AE, obtain the feature vectors, and predict to which cluster they belong.

Regarding the performance measure, we have selected a clustering supervised measure, V-Measure [36], for checking the actual classification capabilities of the framework. Choosing an actual unsupervised clustering metric, such as Silhouette Score would not provide direct insight into the accuracy of the predictions. The training, however, remains fully unsupervised.

As detailed as the first step, we have to build and train the network. However, building and tuning neural networks to be well-performing but also to comply with hardware requirements is a difficult task. Hence, we employ an extended implementation of a NAS framework [6] oriented towards optimizing accuracy, model size, maximum feature map (working memory), and latency. An important note is that, as the search space is composed of conditional variables, we use an implementation of the Arc Kernel [37], which is especially suited for conditional spaces. By decomposing the space using and using a cylindrical embedding, it is able to assess the conditionality among variables. The embedding $g_i: \mathcal{X}_i \rightarrow \mathfrak{R}^2$, where \mathcal{X}_i is the original dimension i space, can be used with any distance and covariance method:

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

¹ For more details, the interested reader can check the implementation at <https://scikit-learn.org/stable/modules/clustering.html#k-means>

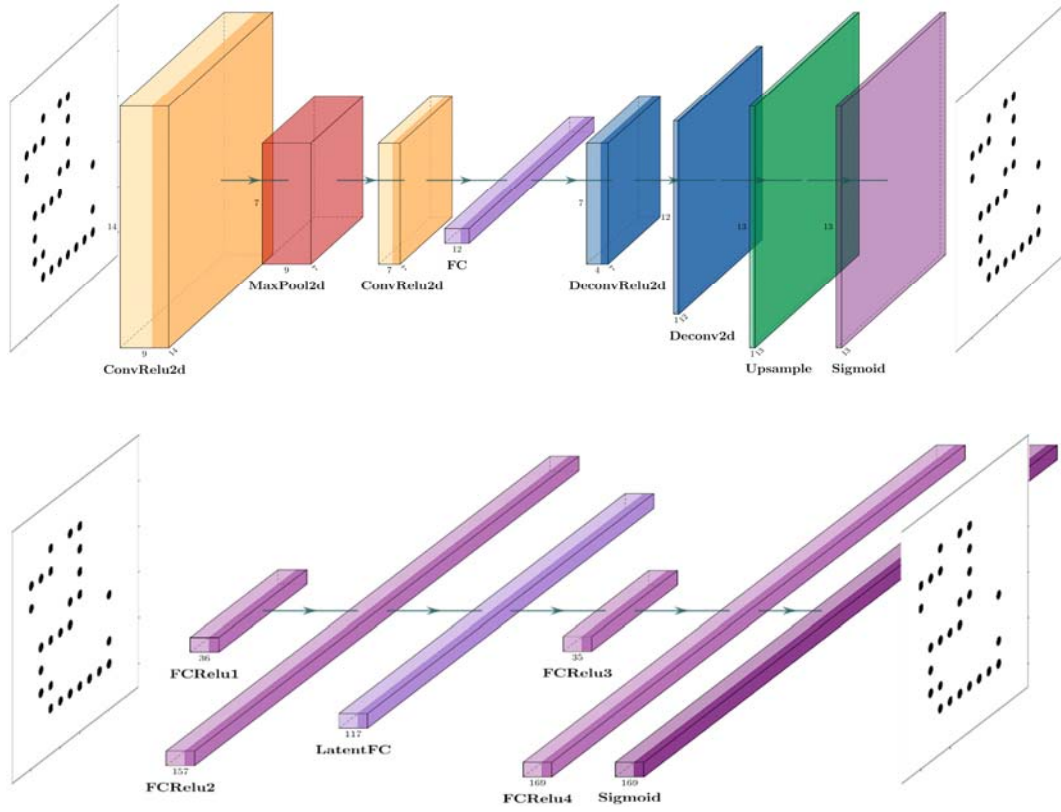


Fig. 3. Best architectures resulting from the optimization procedures for both the CNN and FC autoencoders. The elements in the image represent the feature maps. In the convolution case, the size of the feature map is indicated by the vertical and depth numbers, while the horizontal number indicates the number of feature maps. In the FC case, the number indicates the number of neurons in each step and the images are flattened at the beginning and reshaped at the end.

where $w_i \in \mathfrak{R}^+$ and $\rho_i \in [0, 1]$ are the radius and angle factor of the cylindrical space, $u_i - l_i$ establishes the length scale of the dimension i , and δ_i is a delta function establishing the existence of coupling among dimensions x .

The framework works, in each architecture case, CA or DE, with a configuration space composed of both training hyperparameters and architectural parameters, including post-training linear static quantization and L1-Norm based pruning. Once the search space has been defined, the framework sequentially searches for solutions that minimize the following objectives

$$Error(\Omega_i) = 1 - Accuracy_{validation}(\Omega_i)$$

$$ModelSize(\Omega_i) = \sum_l \|w_l\|_0$$

$$WorkingMemory(\Omega_i) = \max_l (\|x_l\|_0 + \|w_l\|_0 + \|y_l\|_0)$$

$$Latency(\Omega_i) = \frac{InferenceFLOPS}{Platform\ Frequency}$$

where Ω_i is the specific configuration of the search space, l is the specific layer of the network, w_l the weights of layer l , y_l the output of that layer and x_l the input. In our present case, the accuracy corresponds to the correct classification of each pixel of the output with regards to the input.

Once we have the best model found and trained by our NAS framework, we can proceed with the next

steps of training the KM model and obtaining test results.

5. Results

The models established, CAE and DAE, are both developed in PyTorch and the NAS procedure is developed under Ax [38] and BoTorch [39].

First, we illustrate the optimization procedure for the CAE with SpArseMod in Fig. 4. As seen, while the optimization procedure advances there is a decrease in the model size. Also, one can observe the trade-off between the model size and the error: while we continue to obtain smaller architectures the error worsens, obtaining in some cases a compromise.

The final result is an election of the best points of the search: the Pareto frontier. With the best-chosen model, we train and test the K-Means unsupervised clustering and classification. We present the optimization results for the AEs, which correspond to the results for model size, maximum feature map, and latency, and detailed in Table 1, as well as the test results for the K-Means unsupervised classification. As seen, the CAE outperforms the DAE in almost all metrics, except for latency where DAE is a little bit faster. The results for the model size and maximum feature map sizes are low enough to be able to embed the model in a low resource microcontroller.

After the optimization result and after training the KM model with the feature maps obtained, we obtain the test results for the unsupervised classification. In this case, the CAE model performs much better than the DAE, achieving a good result regarding the V-Measure. This indicates the feasibility of this method to provide unsupervised classification.

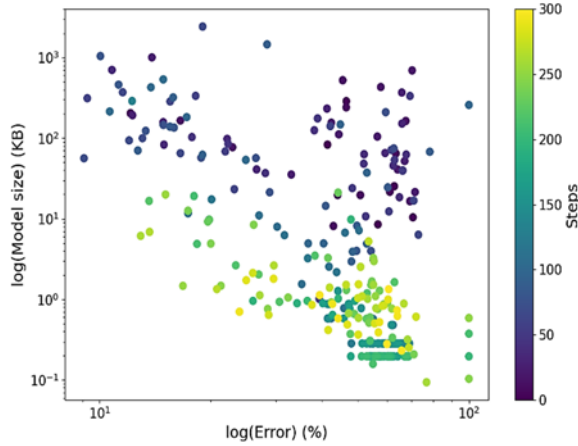


Fig. 4. Optimization procedure for CAE. The y axis represents the Model size and the x-axis the error (per pixel classification). Both axes are in logarithmic scale. The color illustrates the evolution procedure as each of the steps of the Bayesian optimization procedure.

Table 1. Results for the optimization procedure with SpArSeMod. MF corresponds to maximum feature map size, MS to model size, L to latency, and V-M to the V-Measure.

Model	MF (kB)	MS (kB)	L (ms)	V-M (%)
CNN	2.69	6.13	5.53	87.18
FC	18.32	39.67	4.08	58.63

As the final step of our development, we further embed the best model with CMSIS-NN in an NXP-S32K142 microcontroller to perform inference. To perform inference and obtain test results in the embedded implementation, we only need the encoder part of the CAE model and the centroids of KM. We proceed in the following manner: first and for each input, we obtain the feature vector by forwarding the input through the encoder, and second, we measure the distance to all the centroids and choose the nearest one to assign a class.

It is important to note that, in CMSIS-NN legacy API, the quantization used is the power of two based and in the case of PyTorch is linear, enforcing thus a loss of precision. This changes the accuracy of the model but not the size since in both cases we use 8-bit integers. We obtain a V-Measure of 84.08 %, hence resulting in a loss of around 3.10 % in the V-measure, but also validating our embedded deployment.

Finally, to have a visual representation of the clusters, we embed with t-SNE [40] all the encoded

features from the training set and also the centroids. As can be seen in Fig. 5, the clusters are well separated identifying each one as a group of gestures corresponding to a specific number.



Fig. 5. Cluster visualization of the different gesture numbers through t-SNE.

6. Conclusions

We have shown a full pipeline of work for unsupervised classification of gestures. We have employed a CAE plus KM as a model, and a multi-objective NAS to find a proper embeddable model. Finally, we have been able to embed that model into an embedded platform with CMSIS-NN, obtaining good performance results, and thus validating our approach. There are two next important steps to be able to improve the gesture recognition system. First, to change the quantization to linear instead of power of two based to reduce the drop in performance and also to improve. Second, to improve the clustering by adding a Kullback-Leibler divergence component in the loss, hence separating more the feature vectors corresponding to different numbers [41].

Acknowledgments

This project is supported by the Spanish Ministry of Science, Innovation, and Universities under grant RTI2018-095209-B-C22 and the Catalan Government industrial Ph.D. program under grant 2018-DI-3.

References

- [1]. S. Giancola, M. Valenti, R. Sala, A survey on 3D cameras: Metrological comparison of time-of-flight,

- structured-light and active stereoscopy technologies, Springer, 2018.
- [2]. F. M. Caputo, P. Prebianca, A. Carcangiu, L. D. Spano, A. Giachetti, Comparing 3D trajectories for simple mid-air gesture recognition, *Computers and Graphics*, Vol. 73, 2018, pp. 17–25.
 - [3]. T. Grosse-Puppenthal, *et al.*, Finding common ground: A survey of capacitive sensing in human-computer interaction, in *Proceedings of the Conference on Human Factors in Computing Systems (ACM CHI'17)*, 6-11 May 2017, pp. 3293–3316.
 - [4]. R. Binns, M. Veale, M. Van Kleek, N. Shadbolt, Like Trainer, Like Bot? Inheritance of Bias in Algorithmic Content Moderation, *Social Informatics*, 2017, pp. 405–415.
 - [5]. D. Stamoulis, *et al.*, Single-path nas: Designing hardware-efficient convnets in less than 4 hours, in *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2019, pp. 481–497.
 - [6]. I. Fedorov, R. P. Adams, M. Mattina, P. N. Whatmough, SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers, in *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, Vancouver, Canada, 2019, pp. 1–26.
 - [7]. A. Moschetti, L. Fiorini, D. Esposito, P. Dario, F. Cavallo, Toward an Unsupervised Approach for Daily Gesture Recognition in Assisted Living Applications, *IEEE Sens. J.*, Vol. 17, Issue 24, 2017, pp. 8395–8403.
 - [8]. C. Song, F. Liu, Y. Huang, L. Wang, T. Tan, Auto-encoder based data clustering, *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, Vol. 8258 LNCS, Issue PART 1, 2013, pp. 117–124.
 - [9]. J. A. Paradiso, N. Gershenfeld, Musical applications of electric field sensing, *Comput. Music J.*, Vol. 21, Issue 2, 1997, pp. 69–89.
 - [10]. M. Huynh, P. Nguyen, M. Gruteser, T. Vu, POSTER: Mobile Device Identification by Leveraging Built-in Capacitive Signature, in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1635–1637.
 - [11]. K. Hinckley, M. Sinclair, Touch-Sensing Input Devices, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1999, pp. 223–230.
 - [12]. A. Pouryazdan, R. J. Prance, H. Prance, D. Roggen, Wearable Electric Potential Sensing: A New Modality Sensing Hair Touch and Restless Leg Movement, in *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, 2016, pp. 846–850.
 - [13]. A. Nelson, G. Singh, R. Robucci, C. Patel, N. Banerjee, Adaptive and Personalized Gesture Recognition Using Textile Capacitive Sensor Arrays, *IEEE Trans. Multi-Scale Comput. Syst.*, Vol. 1, Issue 2, 2015, pp. 62–75.
 - [14]. S. Escalera, V. Athitsos, I. Guyon, Challenges in multi-modal gesture recognition, *Gesture Recognit.*, 2017, pp. 1–60.
 - [15]. D.-Y. Huang, W.-C. Hu, S.-H. Chang, Vision-based hand gesture recognition using PCA+Gabor filters and SVM, in *Proceedings of the Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 2009, pp. 1–4.
 - [16]. M. A. Moni, A. B. M. S. Ali, HMM based hand gesture recognition: A review on techniques and approaches, in *Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology*, 2009, pp. 433–437.
 - [17]. J. Nagi, *et al.*, Max-pooling convolutional neural networks for vision-based hand gesture recognition, in *Proceedings of the IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, 2011, pp. 342–347.
 - [18]. X. Liu, K. Fujimura, Hand gesture recognition using depth data, in *Proceedings of the Sixth IEEE International Conference on Automatic Face and Gesture Recognition*, 2004, pp. 529–534.
 - [19]. H.-I. Lin, M.-H. Hsu, W.-K. Chen, Human hand gesture recognition using a convolution neural network, in *Proceedings of the IEEE International Conference on Automation Science and Engineering (CASE)*, 2014, pp. 1038–1043.
 - [20]. V. Shanmuganathan, H. R. Yesudhas, M. S. Khan, M. Khari, A. H. Gandomi, R-CNN and wavelet feature extraction for hand gesture recognition with EMG signals, *Neural Comput. Appl.*, Vol. 32, Issue 21, 2020, pp. 16723–16736.
 - [21]. S. Y. Kim, H. G. Han, J. W. Kim, S. Lee, T. W. Kim, A hand gesture recognition sensor using reflected impulses, *IEEE Sens. J.*, Vol. 17, Issue 10, 2017, pp. 2975–2976.
 - [22]. P. Wang, W. Li, S. Liu, Y. Zhang, Z. Gao, P. Ogunbona, Large-scale continuous gesture recognition using convolutional neural networks, in *Proceedings of the 23rd International Conference on Pattern Recognition (ICPR)*, 2016, pp. 13–18.
 - [23]. D. Castells-Rufas, J. Borrego-Carazo, J. Carrabina, J. Naqui, E. Biempica, Continuous touch gesture recognition based on RNNs for capacitive proximity sensors, *Personal and Ubiquitous Computing*, 2020.
 - [24]. P. Molchanov, S. Gupta, K. Kim, J. Kautz, Hand gesture recognition with 3D convolutional neural networks, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2015, pp. 1–7.
 - [25]. B. Jacob, *et al.*, Quantization and training of neural networks for efficient integer-arithmetic-only inference, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.
 - [26]. S. Anwar, K. Hwang, W. Sung, Structured pruning of deep convolutional neural networks, *ACM J. Emerg. Technol. Comput. Syst.*, Vol. 13, Issue 3, 2017, pp. 1–18.
 - [27]. E. Liberis, Ł. Dudziak, N. D. Lane, μNAS: Constrained Neural Architecture Search for Microcontrollers, *Computer Science*, 2020.
 - [28]. L. Lai, N. Suda, V. Chandra, CMSIS-NN: Efficient neural network kernels for arm Cortex-M CPUs, *arXiv Prepr. arXiv1801.06601*, 2018.
 - [29]. Y. Zhang, N. Suda, L. Lai, V. Chandra, Hello edge: Keyword spotting on microcontrollers, *arXiv Prepr. arXiv1711.07128*, 2017.
 - [30]. M. Scherer, M. Magno, J. Erb, P. Mayer, M. Eggimann, L. Benini, TinyRadarNN: Combining Spatial and Temporal Convolutional Neural Networks for Embedded Gesture Recognition with Short Range Radars, *arXiv Prepr. arXiv2006.16281*, 2020.
 - [31]. E. Torti, *et al.*, Embedded real-time fall detection with deep learning on wearable devices, in *Proceedings of the 21st Euromicro Conference on Digital System Design (DSD)*, 2018, pp. 405–412.
 - [32]. H. V. Le, S. Mayer, N. Henze, Investigating the feasibility of finger identification on capacitive


- touchscreens using deep learning, in *Proceedings of the 24th International Conference on Intelligent User Interfaces*, 2019, pp. 637–649.
- [33]. F. Sakr, F. Bellotti, R. Berta, A. De Gloria, Machine learning on mainstream microcontrollers, *Sensors*, Vol. 20, Issue 9, 2020, p. 2638.
- [34]. U. Borgmann, Method for Detecting Contact on a Capacitive Sensor Element, US Patent App. 16/354, 699, March 2019.
- [35]. D. Arthur, S. Vassilvitskii, k-means++: The advantages of careful seeding, *Stanford*, 2006.
- [36]. A. Rosenberg, J. Hirschberg, V-measure: A conditional entropy-based external cluster evaluation measure, in *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007, pp. 410–420.
- [37]. K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, M. A. Osborne, Raiders of the lost architecture: Kernels for Bayesian optimization in conditional parameter spaces, *arXiv Prepr. arXiv1409.4011*, 2014.
- [38]. E. Bakshy, *et al.*, AE: A domain-agnostic platform for adaptive experimentation, *SemanticScholar*, 2018.
- [39]. M. Balandat, *et al.*, BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization, in *Advances in Neural Information Processing Systems*, Vol. 33, 2020.
- [40]. L. van der Maaten, G. Hinton, Visualizing data using t-SNE, *J. Mach. Learn. Res.*, Vol. 9, Issue 86, 2008, pp. 2579–2605.
- [41]. V. Prokhorov, E. Shareghi, Y. Li, M. T. Pilehvar, N. Collier, On the importance of the Kullback-Leibler divergence term in variational autoencoders for text generation, *arXiv Prepr. arXiv1909.13668*, 2019.



Published by International Frequency Sensor Association (IFSA) Publishing, S. L., 2021 (<http://www.sensorsportal.com>).

Universal Sensors and Transducers Interface (USTI)

for any sensors and transducers with frequency, period, duty-cycle, time interval, PWM, phase-shift, pulse number output



- * Input frequency range: 0.05 Hz ... 9 MHz (144 MHz)
- * Selectable and constant relative error: 1 ... 0.0005 % for all frequency range
- * Scalable resolution
- * Non-redundant conversion time
- * RS232, SPI, I2C interfaces
- * Rotational speed, *rpm*
- * Cx, 50 pF to 100 μF
- * Rx, 10 Ω to 10 MΩ
- * Pt100, Pt1000, Pt5000, Cu, Ni
- * Resistive Bridges
- * PDIP, TQFP, MLF packages

Just make it easy !