micromachines

MDPI

Article

Smart Tactile Sensing Systems Based on Embedded CNN Implementations

Mohamad Alameh [1], Yahya Abbass [1], Ali Ibrahim [1,2,*] and Maurizio Valle [1]

[1] Department of Electrical, Electronic and Telecommunication Engineering and Naval Architecture (DITEN)-University of Genoa, via Opera Pia 11a, 16145 Genova, Italy; Mohamad.Alameh@edu.unige.it (M.A.); Yahya.Abbass@edu.unige.it (Y.A.); Maurizio.Valle@unige.it (M.V.)

[2] Department of Electrical and Electronics Engineering, Lebanese International University (LIU), Beirut 1105, Lebanon

* Correspondence: Ali.Ibrahim@edu.unige.it; Tel.:+39-3279364917

Received: 1 December 2019; Accepted: 15 January 2020; Published: 18 January 2020

Abstract: Embedding machine learning methods into the data decoding units may enable the extraction of complex information making the tactile sensing systems intelligent. This paper presents and compares the implementations of a convolutional neural network model for tactile data decoding on various hardware platforms. Experimental results show comparable classification accuracy of 90.88% for Model 3, overcoming similar state-of-the-art solutions in terms of time inference. The proposed implementation achieves a time inference of 1.2 ms while consuming around 900 µJ. Such an embedded implementation of intelligent tactile data decoding algorithms enables tactile sensing systems in different application domains such as robotics and prosthetic devices.

Keywords: tactile sensing systems; embedding intelligence; convolutional neural network

1. Introduction

Embedding intelligence near the sensor location may enable tactile sensing systems to be incorporated in many application domains such as prosthetics, robotics, and the Internet of Things. Tactile sensing systems are composed of three main parts, as shown in Figure 1. The distributed tactile sensors are in charge of converting the mechanical stimuli applied on their surface into electrical signals. Tactile sensors could be made from different materials, e.g., capacitive, piezoelectric, and piezoresistive materials [1]; they should be able to enable capabilities similar to what happens on the human skin such as normal and shear force detection, vibration detection, softness, texture, shapes, etc. The readout electronics interface with the sensor arrays by acquiring and digitizing the electrical signals to be then processed by the digital tactile data processing unit [2].

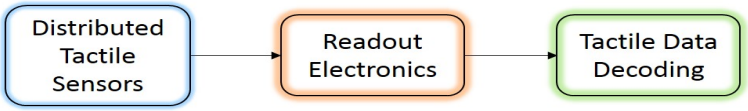

Figure 1. Block diagram of the tactile sensing system.

Decoding tactile information concerns different kinds of tasks, which could be categorized as: simple or complex processing depending on the algorithm's complexity. For simple processing, an example of the information retrieved is temperature, the intensity of the contact force, and contact location, direction, and distribution. Concerning complex processing, more intelligent tasks are expected such as patterns, textures, and roughness classification or touch modalities' discrimination. Employing the complex processing approach enables intelligence in tactile sensing systems. It is

achieved by applying sophisticated and complex data decoding algorithms able to extract the meaningful information from sensors. Machine learning (ML) has emerged as an efficient method in many fields and in everyday tasks in smartphones and electronic systems. ML is a powerful learning from examples paradigm used to address classification and regression problems. In particular, Convolutional (CNN) and Deep Neural Networks (DNN) have recently proven their effectiveness when applied to image recognition and tactile data decoding [3]. Many recent research works have focused on the development of ML algorithms for tactile sensing systems [4]. However, embedding machine learning algorithms on hardware platforms near the sensors location is challenging due to the complexity such algorithms impose in terms of time latency and energy consumption. Our main goal is to achieve a tactile sensing system able to perform smart tasks. This system is intended to be portable/wearable, for which the energy budget is limited. Moreover, for the target applications, i.e., robotics and prosthetics, being lightweight is a critical constraint limiting the hardware and battery size.

In this perspective, this paper presents the implementation of CNN algorithms on different hardware platforms. The main contribution of this paper may be summarized as follows:

- It proposes an optimized CNN model, adopted from Gandarias et al.'s research [5], based on reduced data, which demonstrates the ability to provide comparable results in terms of accuracy, i.e., 90.88%, with reduced hardware complexity.
- It presents efficient implementations of the CNN model on different hardware platforms for embedded tactile data processing. The proposed implementations achieve a time inference of 1.2 ms while consuming around 900 µJ. The work demonstrates its suitability for real-time embedded tactile sensing systems.
- It raises a discussion about integrating intelligence into tactile sensing systems and how it enables tactile sensing systems in different application domains.

The remainder of this paper is organized as follows: Section 2 reports the state-of-the-art, showing the recent embedded CNN implementations. In Section 3, we illustrate the experimental setup and methodology. In Section 4, the hardware implementation is explained. The results and discussion are presented in Section 5, followed by the conclusions in Section 6.

2. State-of-the-Art

Different works have addressed the tactile data classification problem, using different methods including, but not limited to, machine learning and deep learning [6–10]. While most of the work done was focused on the methodology itself, few works addressed the implementation on embedded platforms where the real application should reside. Gandarias et al. [11] used two approaches to classify eight objects: finger, hand, arm, pen, scissors, pliers, sticky tape, and Allen key, using a 28 × 50 tactile sensory array attached to a robotic arm, the first approach using the Speeded-Up Robust Features (SURF) descriptor, while the second a pre-trained AlexNet CNN for feature extraction, with a Support Vector Machine (SVM) classifier for both approaches. In Yuan et al.'s research [12], a CNN was also used for active tactile clothing perception, to classify clothes grasped by a robotic arm equipped with a tactile sensor that output a large RGB pressure map. Based on different textile properties: thickness, smoothness, textile type, washing method, softness, stretchiness, durability, woolen, and wind proof. Each property held two or more labels, e.g., the thickness can be a number from 0–4, and the employed model for textile classification was VGG-19 pretrained on ImageNet [13]. In Rouhafzay et al. [14], a combination of virtual tactile sensors and visual guidance was employed to distinguish eight classes of simulated objects; the tactile sensor size was 32 × 32, and the input size of the neural network was 32 × 32 × 32, which was a sequence of tactile sensor images. Abderrahmane et al. [15] introduced a zero-shot object recognition framework, to identify previously unknown objects based on haptic feedback. They used BioTac sensors, and two CNNs were employed: one for visual data (input size: 224 × 224 × 30) and the other for tactile data (32 × 30). They overcame the results of SVM in a previous

work [16]. In Alameh et al.'s research [3], transfer learning was used to classify touch modalities obtained through a small 4 × 4 piezoresistive sensory array, by transforming tensorial data into images and then using different CNN models trained on ImageNet [13]. In Gandarias et al.'s research [5], they used a light CNN based (only three convolutional layers inside) on AlexNet, to identify 22 objects using their pressure map, collected from a 28 × 50 tactile sensory array. Other works include those in [17–19].

While all these previous works were not implemented in an embedded environment, we can find few others targeting an embedded implementation for tactile sensing applications. The need for embedded implementation arises from the need to have low power, small form factor electronics to process the tactile information, especially in prosthetic applications [20]. Osta et al. [21] demonstrated an energy efficient system for binary touch modality classification, based on SVM and implemented on a custom hardware architecture. The energy per inference was 81 mJ, and the inference time was 3.3 s. Ibrahim et al. [22] presented a real-time implementation on FPGA for touch modality classification. Using SVM, they achieved a 350 ms inference time and a 945 mJ inference energy for three class classification, as well as 970 ms/6.01 J for a five class classification.

3. Experimental Setup and Methodology

3.1. Dataset

Targeting the classification of tactile data, the use of the dataset collected in [5] was considered. Tactile data were collected by a high resolution (1400 pressure taxels) tactile array, which was attached to the 6 DOF robotic arm AUBO Our-i5 [5]. A set of piezoresistive tactile sensors was distributed with a density of 27.6 taxels/cm^2, forming a matrix of 28 rows by 50 columns. The dataset was composed of pressure images that presented the compliance of 22 objects with the tactile sensors. These images were divided into 22 classes labeled as adhesive, Allen key, arm, ball, bottle, box, branch, cable, cable pipe, caliper, can, finger, hand, highlighter pen, key, pen, pliers, rock, rubber, scissors, sticky tape, and tube. Figure 2 shows an example of the tactile images of three objects used for the training of the CNN model. Each taxel in the tactile array presents a pixel in the pressure image; thus, each pressure image is 28 × 50 × 3 in size. Therefore, the color of the pixel presents the pressure applied at the corresponding taxel. The minimum pressure is presented by black color, and the maximum pressure is presented by red color. Pressure images were then transformed into grayscale images (image size = 28 × 50 × 1), forming the tactile dataset.

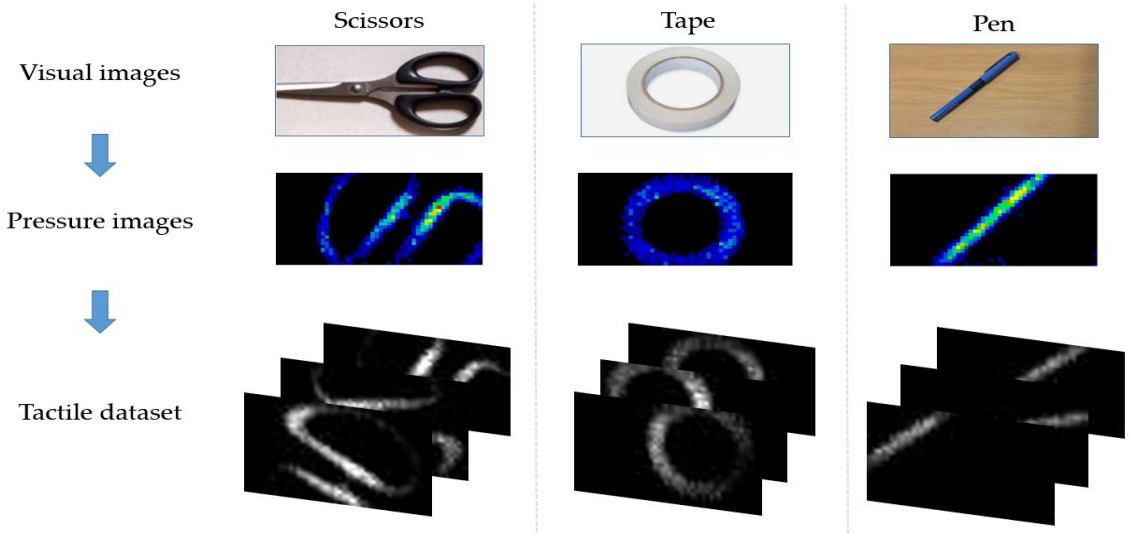

Figure 2. Examples of visual (**top**) vs. pressure (**middle**) vs. tactile images (**bottom**) of common objects.

3.2. Tested Model

Due to computational and memory limitations in the embedded application, a light CNN model was required to perform classification tasks with high accuracy and fewer parameters. In this work, we chose to use one of the models implemented in [5] as a base model to classify the objects in the aforementioned dataset. Among all the implemented networks, we chose to use the custom network TacNet4 because it was the best network that fit the embedded application (fewer parameters with high accuracy [5]). The model was based on AlexNet, which is usually used in computer vision for object classification [23]. The network was composed of 3 Convolutional layers (Conv1, Conv2, and Conv3) with filters sizes (5×5, 8), (3×3, 16), and (3×3, 32) respectively. Each convolutional layer was followed by a Batch Normalization (BaN), Activation (ReLU), and Maxpooling (Maxpool) layer, respectively, where all pooling layers used 2×2 maxpooling with a stride of two. A Fully Connected layer (FC = [fc4]) with 22 neurons followed by a softmax layer were used to classify the input tactile data and give the likelihood of belonging to each class (object). The input shape of the model was configured to the size of the collected tactile data. Figure 3 shows the detailed structure of the network used.

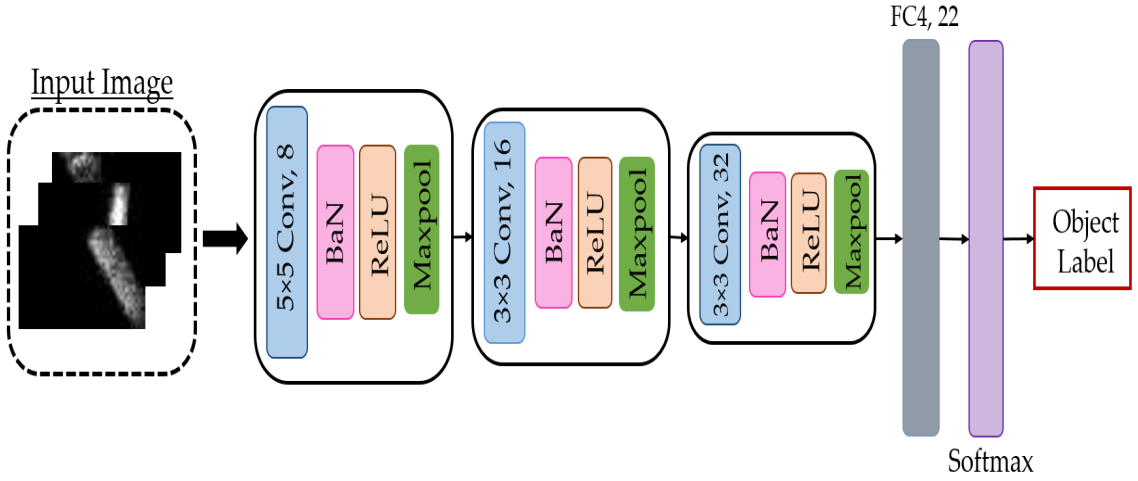

Figure 3. Architecture of the tested model. BaN, Batch Normalization.

The network was implemented in MATLAB R2019b using the Neural Network Toolbox. A total of 1100 tactile images were used to train the model. The learning process was implemented in MATLAB by dividing the tactile data into three sets: training, validation, and test sets.

When having an adequate dataset, the validation set is expected to be a good statistical representation of the entire dataset. If not, the results of the training procedure highly depend on how the dataset is divided.

To avoid this, In this work, we used the cross-validation method. The data were partitioned into five folds, and each fold was divided into training, validation, and test sets. The training set formed 80% of the dataset, and the validation and test sets formed 10% each. This process was then repeated five times until all the folds were used, without having common elements across all folds for the validation and test sets, as shown in Figure 4.

For each training process, the training set was composed of 880 images, 40 images for each label, whilst each of the validation and test sets was composed of 110 images. Training the model from scratch required a large dataset to achieve high accuracy. For that reason, data augmentation techniques, i.e., flipping, rotation, and translation in the X and Y axis, were applied to the dataset. Hence, the amount of tactile data available for training and validation was increased to 5280 and 660, respectively. The performance of the implemented model was evaluated based on the recognition rates

achieved in a classification experiment of the test set composed of 110 original images (objects) from 22 classes.

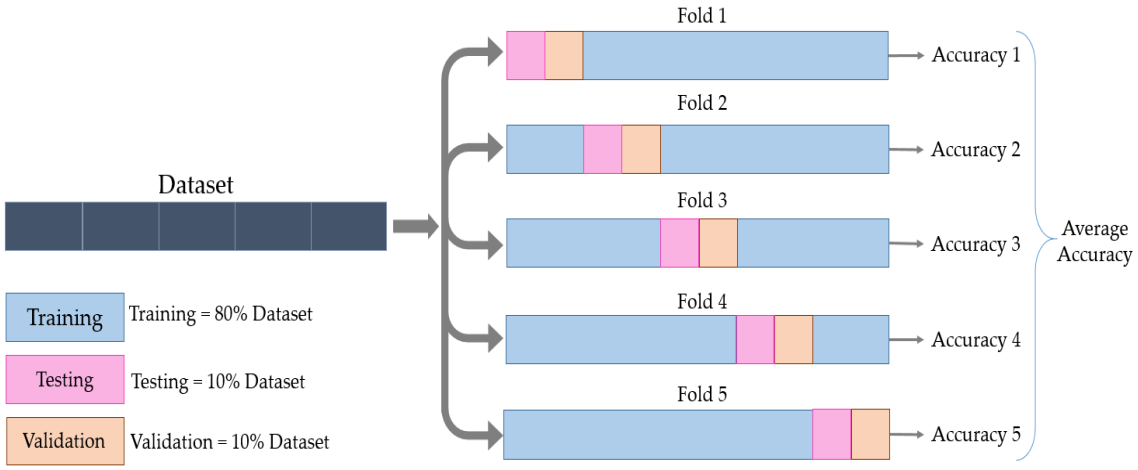

Figure 4. Visual representation of the training, test, and validation split using cross-validation.

For embedded applications, with computational, memory, and energy constraints, it is necessary to decrease the number of trainable parameters in the CNN model. In this work, we chose to decrease the number of parameters of the trained model by decreasing the input image size (i.e., lower resolution images); an example is shown in Figure 5. For that reason, several experiments were performed to choose the smaller size of the input data, keeping the same classification accuracy. The input shapes were chosen in a way that each shape resulted in a reduction of the number of parameters.

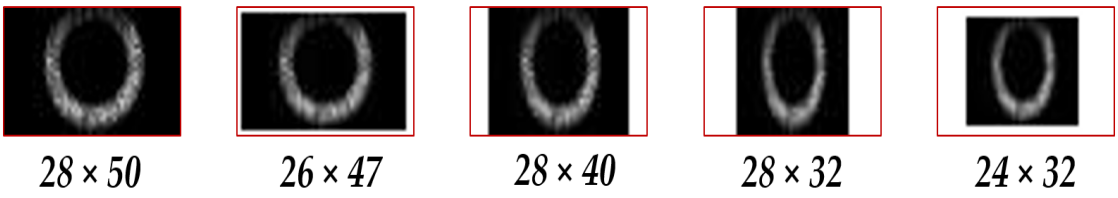

Figure 5. Example of an image resized for the sticky tape object; the red canvas is shown for illustration, which signifies the original image size (28 × 50).

Table 1 shows how the number of parameters of the layers depended on the input shape. The change in the input shape affected only the number of parameters of the fully connected layer. This was due to the fact that the number of parameters in the convolutional layer depended only on the size and number of the filters assigned for each layer (((width of the filter × height of the filter) + 1) × No. of filters), while in the FC layer, the number of parameters ((No. of neurons in the FC layer × No. of neurons in the previous layer) + 1) was affected by the size of the input image and the output layer. The performance of the model was studied with five different input shapes, as shown in Table 1. This resulted in five different models with different input shapes, each one trained from scratch 5 times (one time per fold), which output 25 trained NNs. Figure 6 shows the training and validation accuracy over epochs, for the first three models among the five models. The figure shows that the accuracy achieved by the three models was close to 100%. Each model was evaluated with MATLAB by running a classification task on the test set.

Table 1. Distribution of the number of parameters on the models' layers.

Layers	Model 1 (28 × 50)	Model 2 (26 × 47)	Model 3 (28 × 40)	Model 4 (28 × 32)	Model 5 (24 × 32)
Conv1	208	208	208	208	208
BaN1	16	16	16	16	16
Conv2	1168	1168	1168	1168	1168
BaN2	32	32	32	32	32
Conv3	4640	4640	4640	4640	4640
BaN3	64	64	64	64	64
FC	19,734	16,918	14,102	11,286	8470
Total	25,862	23,046	20,230	17,414	14,598

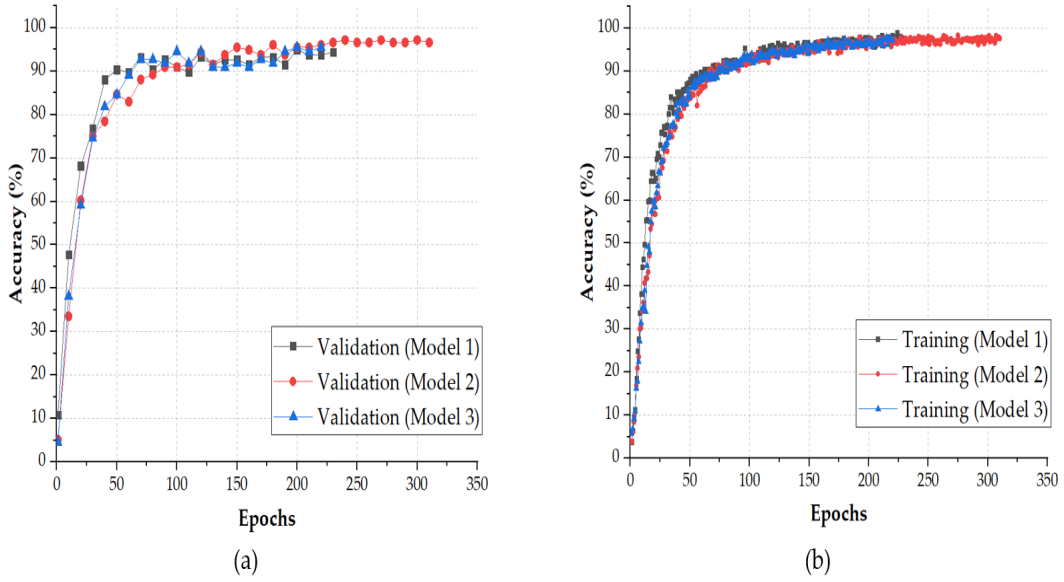

Figure 6. Learning accuracy for the 3 configurations of the TactNet4model: (**a**) training; (**b**) validation.

Figure 7 shows the change in the number of trainable parameters and the average classification accuracy, with respect to the change in the input shape, as well as the FLOPs. The classification accuracy presented the average test accuracy among the five folds. The figure shows that it was possible to decrease the input size from 28 × 50 × 1 to 26 × 47 × 1 or to 28 × 40 × 1 and achieve an increase in the classification accuracy from 90.70% to 91.98% and 90.88%, respectively. Decreasing the input size of the model resulted in a drop in the trainable parameters from 25,862 to 23,046 and 20,230 parameters, respectively, for the aforementioned models. This decrease in the number of parameters would also induce a decrease of the number of Floating Point Operations (FLOPs), as shown in Figure 7; the average ratio of the decrease in the number of parameters with respect to the decrease in the number of FLOPs was 1/44 i.e., with each decrease in number of parameters, there was a 44 times decrease of the FLOPs. The number of FLOPS in Figure 7 corresponds to the convolutional layers only, where most of the FLOPs were, and these FLOPs were calculated according to the following formula [24]: FLOPs = n × m × k, where n is the number of kernels, k is the size of the kernel (width × height × depth), and m the size of output feature map (width × height), while the depth in the kernel size corresponds to the depth of the input feature map.

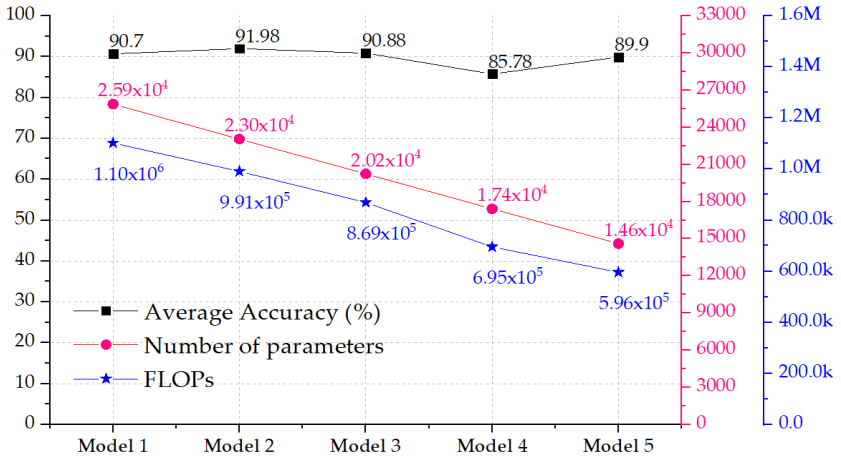

Figure 7. Comparison of the performance, number of trainable parameters, and FLOPS in the convolutional layers.

4. Embedded Hardware Implementations

The models obtained from MATLAB were converted to Open Neural Network Exchange (ONNX) format [25]. ONNX provides an open source format for AI models, both deep learning and traditional ML, which enables the inter-operability between different frameworks. Figure 8 shows how the CNN model was converted into different formats for different hardware platforms. Figure 7 shows the number of trainable parameters and the corresponding accuracy for each model. It is clearly shown that all models preserved comparable accuracy, but the best were the first three, i.e., Model 1, Model 2, and Model 3. However, since Model 2 and Model 3 demonstrated a reduced number of training parameters and accordingly a reduced number of operations (FLOPS), they were selected for the hardware implementation. This choice was based on the fact that reducing FLOPS reduced the inference time and power consumption.

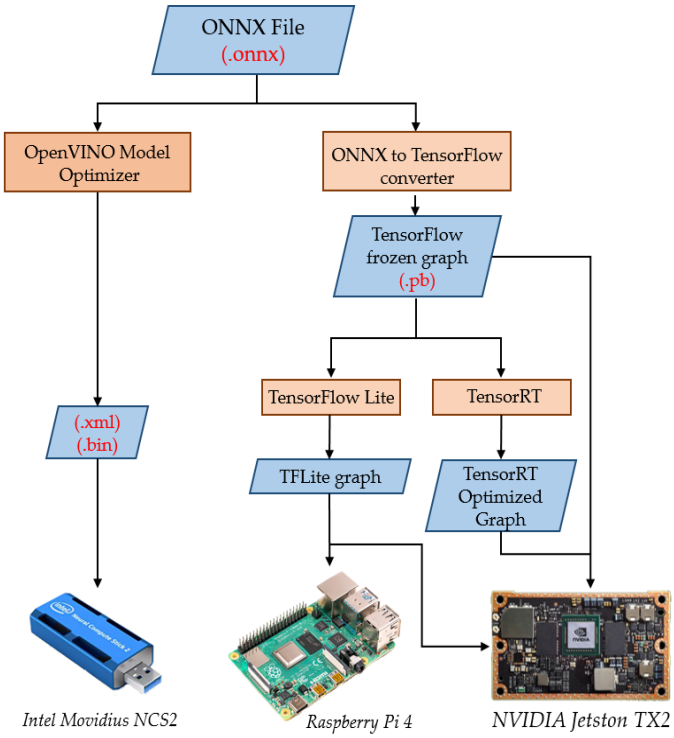

Figure 8. Implementation flow.

The reason behind the selection of hardware platforms was as follows:

1. The custom architecture targeting the embedded implementation of neural networks, e.g., Movidius NCS2.
2. The high usability of ARM processors in embedded architectures, e.g., Raspberry Pi 4.
3. The high performance architecture, designed for parallel processing in general, and also optimized for embedded applications: e.g., NVidia Jetson TX2.
4. The support for the execution of pretrained neural network models coming from different platforms without retraining.

4.1. Movidius Neural Compute Stick 2

Movidius Neural Compute Stick 2 (NCS2) is a hardware accelerator designed by Intel for on-chip neural network inference, especially CNNs, equipped with the Intel Movidius MyriadX Vision Processing Unit (VPU). It contains 16 SHAVE (Streaming Hybrid Architecture Vector Engine) cores [26] and a dedicated hardware neural network accelerator. It requires a host to flash the neural network, as well as to feed it with data and invoke the inference to get the results back via the USB 3.0 port. The host can be a Linux, Windows, or Mac based machine. To achieve these tasks, Intel provides OpenVINO: Open Visual Inference and Neural network Optimization Toolkit, a cross platform toolkit that enables deep learning inference and easy heterogeneous execution across multiple Intel® hardware (VPU, GPU, CPU, FPGA). The optimizations offered by OpenVINO are: batch normalization and scale shift, linear operation merge and linear operation fusion. The details were mentioned in [27].

4.2. Jetson TX2

NVidia's Jetson TX2 [28] is a power efficient embedded AI computing device, designed mainly for edge AI, and belongs to the Pascal™ family of GPUs, loaded with 8 GB of memory, 59.7 GB/s of memory bandwidth, and 8 GB of RAM. In this experiment, we used TensorFlow (TF) [29] for the inference, as well as NVidia TensorRT [30] under Ubuntu OS. TF is an open source end-to-end machine learning platform, while TensorRT is a platform for high performance deep learning inference dedicated to NVidia hardware. It includes a deep learning inference optimizer and a runtime that delivers low latency and high throughput for deep learning inference applications.

As an optimization for TensorFlow, TensorFlow Lite (TFLite) [31] is an open source deep learning framework for on device inference. The same TensorFlow model can be converted into the TFLite model. To perform an inference with a TFLite model, the TFLite interpreter is required, which uses a static graph ordering and a custom (less dynamic) memory allocator to ensure minimal load, initialization, and execution latency [31], also reducing the weights' precision, e.g., floating point vs. fixed point precision, without affecting the accuracy.

4.3. ARM

As for the implementation on the ARM architecture, we used Raspberry Pi 4, equipped with a Quad core Cortex-A72 (ARM v8) 64-bit System on Chip (SoC) @ 1.5 GHz and 4 GB RAM. For the inference on this hardware, we used the TFLite runtime library, under the Ubuntu OS.

For all the mentioned platforms, both power consumption and inference time were calculated. The inference time was calculated by averaging 110 inferences, which corresponded to the test set size. As for the power consumption, two methods were used:

1. the provided APIs in Jetson TX2, which provided readings about voltage, power, and input current to the GPU.
2. the external USB multimeter, connected in serial to the power source for both Raspberry Pi and the Movidius NCS2.

5. Results and Discussion

In this work, we achieved a better accuracy in tactile data classification using CNN compared to the original model obtained in [5], even by resizing the input, therefore decreasing the number of trainable parameters. The chosen models reduced the number of trainable parameters by a maximum of 21.77% of the original trainable parameters and also increased the accuracy by a maximum of 1.28%, noting that Model 5 ($24 \times 32 \times 1$) with 0.8% less accuracy than the original model had 42% fewer trainable parameters. Choosing the right model depended on the implementation, i.e., a trade-off between accuracy and hardware complexity should take place: if the best accuracy was targeted, then Model 2 should be selected; while the choice of Model 3 would be when less hardware complexity was needed, accepting a small accuracy degradation. Reducing the input size while still keeping the same or even better accuracy could be explained in three points:

1. The random initialization of the weights may lead in different runs to different accuracy results, e.g., 10 different runs for training Fold 4 of Model 2 with the same hyperparameters gave different results, as shown in Table 2, which shows an average of 94.36% and a standard deviation of 1.904%.
2. Random selection of batch data and data shuffling would affect also the update of the weights and make them different from one training to another.
3. The feature extraction process achieved by CNN was error resilient [32]. A CNN could still extract features even with some manipulation of the input image. This was one of the reasons for data augmentation [33] when training neural networks, which was to let the neural network learn the features even from augmented images (scaled, rotated, flipped, etc.), instead of learning only the samples in the original dataset. In our case, the features were still detectable even after image resizing, as shown in Figure 5.

Table 2. Accuracy results for 10 runs on Model 2, Fold 4.

Trials	Accuracy (%)
1	96.36
2	92.73
3	94.55
4	91.82
5	97.27
6	93.64
7	92.73
8	95.45
9	96.36
10	92.73
Average ± Stdev	94.36 ± 1.904%

According to Tables 3 and 4, the smallest power consumption and inference time were obtained using TensorRT under Jetson TX2, which was 153 mW dynamic power within 5.29 ms as the inference time, implying 0.809×10^{-3} Joules of dynamic energy (see Table 5). The most dynamic energy consumption was for the Intel Movidius NCS2, 1.9 ms × 800 mW = 1.52×10^{-3} Joules. Regarding the power consumption, since the neural network used was small compared to the hardware capacity, the power consumption was almost the same for the three models, noting that the accuracy on the USB power meter was on the 10 mW scale, so that a difference of less than 10 mW between two measurements could not be detected using this instrument.

Table 3. Comparison of the inference time between models.

Platform		Inference Time (ms)		
Hardware	**Software**	**Model 1**	**Model 2**	**Model 3**
Jetson TX2	TensorRT	5.5597	5.2905	5.919
	TF	6.2943	5.4691	5.946
	TFLite	1.3384	1.2181	1.2445
Core i7	MATLAB	3.245	2.6139	2.4715
Movidius NCS2	OpenVINO	1.9	1.9	1.86
Raspberry Pi4	TFLite	1.615	1.473	1.21

Table 4. Power consumption.

Platform		Current (mA)		Voltage (V)	Consumed Power (mW)		
Hardware	**Software**	**Static**	**Total**		**Static**	**Total**	**Dynamic**
Jetson	TensorRT	8	16	19.072	152	305	153
	TF	8	16	19.072	152	305	153
Movidius NCS2	OpenVINO	-	160	5	-	800	800
Raspberry Pi4	TFLite	560	700	5	2800	3500	700

Table 5. Energy consumption.

Platform		Energy Consumption (μJ)		
Hardware	**Software**	**Model 1**	**Model 2**	**Model 3**
Jetson TX2	TensorRT	850.6341	809.4465	905.607
	TF	963.0279	836.7723	909.738
Movidius NCS2	Open VINO	1520	1520	1488
Raspberry Pi4	TFLite	1130.5	1031.1	847

6. Conclusions

This paper presented the implementation of a smart tactile sensing system based on an embedded CNN approach. The proposed model optimized a state-of-the-art model proposed in [5] by reducing the input data size. The experimental results were comparable in terms of accuracy after reducing the size from (28 × 50) to (26 × 47) and (28 × 40). The hardware implementation on different hardware platforms, namely Movidius NCS2, NVidia's Jetson TX2, and Cortex-A72 (ARM v8), was provided. The proposed models showed better performance on hardware platforms when time inference was compared. Power consumption was also measured and compared among different platforms. Targeting portable tactile sensing systems, the proposed work demonstrated the feasibility of integrating machine learning methods on a hardware platform to enable intelligence for such a system. This may pave the way for smart tactile sensing systems to be applied in prosthetics and robotics.

Author Contributions: Conceptualization, A.I.; methodology, M.A. and A.I.; software, M.A. and Y.A; validation, A.I.; investigation, M.A. and Y.A.; data curation, M.A. and Y.A.; writing, original draft preparation, Y.A., M.A., and A.I.; writing, review and editing, A.I. and M.V.; visualization, Y.A.; supervision, A.I. and M.V.; funding acquisition, M.V. All authors read and agreed to the published version of the manuscript.

Funding: The authors acknowledge financial support from Compagnia di San Paolo, Grant Number 2017.0559, ID ROL19795.

Acknowledgments: The authors would like to thank the NVidia corporation for the donation of the Jetson TX2 development kit.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CNN	Convolutional Neural Network
DCNN	Deep Convolutional Neural Network
SVM	Support Vector Machine
ML	Machine Learning
FPGA	Field-Pogrammable Gate Array

References

1. Ibrahim, A.; Pinna, L.; Seminara, L.; Valle, M. Achievements and Open Issues Toward Embedding Tactile Sensing and Interpretation into Electronic Skin Systems. In *Material-Integrated Intelligent Systems-Technology and Applications*; John Wiley & Sons, Ltd.: WeinHeim, Germany, 1 December 2017; Chapter 23, pp. 571–594. doi:10.1002/9783527679249.ch23. [CrossRef]
2. Saleh, M.; Abbass, Y.; Ibrahim, A.; Valle, M. Experimental assessment of the interface electronic system for PVDF-based piezoelectric tactile sensors. *Sensors* **2019**, *19*, 4437. doi:10.3390/s19204437. [CrossRef] [PubMed]
3. Alameh, M.; Ibrahim, A.; Valle, M.; Moser, G. DCNN for Tactile Sensory Data Classification based on Transfer Learning. In Proceedings of the 2019 15th Conference on Ph.D Research in Microelectronics and Electronics (PRIME), Lausanne, Switzerland, 15–18 July 2019; pp. 237–240. doi:10.1109/prime.2019.8787748. [CrossRef]
4. Luo, S.; Bimbo, J.; Dahiya, R.; Liu, H. Robotic tactile perception of object properties: A review. *Mechatronics* **2017**, *48*, 54–67. doi:10.1016/j.mechatronics.2017.11.002. [CrossRef]
5. Gandarias, J.M.; Garcia-Cerezo, A.J.; Gomez-de Gabriel, J.M. CNN-Based Methods for Object Recognition With High-Resolution Tactile Sensors. *IEEE Sens. J.* **2019**, *19*, 6872–6882. doi:10.1109/jsen.2019.2912968. [CrossRef]
6. Cheng, G.; Dean-Leon, E.; Bergner, F.; Olvera, J.R.G.; Leboutet, Q.; Mittendorfer, P. A Comprehensive Realization of Robot Skin: Sensors, Sensing, Control, and Applications. *Proc. IEEE* **2019**, *107*, 2034–2051. doi:10.1109/JPROC.2019.2933348. [CrossRef]
7. Martinez-Hernandez, U.; Dodd, T.J.; Prescott, T.J. Feeling the Shape: Active Exploration Behaviors for Object Recognition With a Robotic Hand. *IEEE Trans. Syst. Man Cybern. Syst.* **2018**, *48*, 2339–2348. doi:10.1109/TSMC.2017.2732952. [CrossRef]
8. Zou, L.; Ge, C.; Wang, Z.; Cretu, E.; Li, X. Novel tactile sensor technology and smart tactile sensing systems: A review. *Sensors* **2017**, *17*, 2653. [CrossRef] [PubMed]
9. Li, R.; Adelson, E.H. Sensing and Recognizing Surface Textures Using a GelSight Sensor. In Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, 23–28 June 2013; IEEE: Portland, OR, USA, 2013; pp. 1241–1247. doi:10.1109/CVPR.2013.164. [CrossRef]
10. Schmitz, A.; Bansho, Y.; Noda, K.; Iwata, H.; Ogata, T.; Sugano, S. Tactile object recognition using deep learning and dropout. In Proceedings of the 2014 IEEE-RAS International Conference on Humanoid Robots, Madrid, Spain, 18–20 November 2014; pp. 1044–1050. doi:10.1109/HUMANOIDS.2014.7041493. [CrossRef]
11. Gandarias, J.M.; Gomez-de Gabriel, J.M.; Garcia-Cerezo, A. Human and object recognition with a high-resolution tactile sensor. In Proceedings of the 2017 IEEE SENSORS, Glasgow, UK, 29 October–1 November 2017; IEEE: Glasgow, 2017; pp. 1–3. [CrossRef]
12. Yuan, W.; Mo, Y.; Wang, S.; Adelson, E. Active Clothing Material Perception using Tactile Sensing and Deep Learning. *arXiv* **2017**, arXiv:1711.00574.
13. ImageNet. Available online: http://www.image-net.org (accessed on 20 November 2019).
14. Rouhafzay, G.; Cretu, A.M. An Application of Deep Learning to Tactile Data for Object Recognition under Visual Guidance. *Sensors* **2019**, *19*, 1534. doi:10.3390/s19071534. [CrossRef] [PubMed]
15. Abderrahmane, Z.; Ganesh, G.; Crosnier, A.; Cherubini, A. Visuo-Tactile Recognition of Daily-Life Objects Never Seen or Touched Before. In Proceedings of the 2018 IEEE 15th International Conference on Control, Automation, Robotics and Vision (ICARCV), Singapore, 18–21 November 2018; pp. 1765–1770.

16. Abderrahmane, Z.; Ganesh, G.; Crosnier, A.; Cherubini, A. Haptic Zero-Shot Learning: Recognition of objects never touched before. *Robot. Auton. Syst.* **2018**, *105*, 11–25. doi:10.1016/j.robot.2018.03.002. [CrossRef]
17. Li, J.; Dong, S.; Adelson, E. Slip detection with combined tactile and visual information. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 21–25 May 2018; pp. 7772–7777.
18. Wu, H.; Jiang, D.; Gao, H. Tactile motion recognition with convolutional neural networks. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 1572–1577.
19. Kwiatkowski, J.; Cockburn, D.; Duchaine, V. Grasp stability assessment through the fusion of proprioception and tactile signals using convolutional neural networks. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; IEEE: Vancouver, BC, Canada, 2017; pp. 286–292. doi:10.1109/IROS.2017.8202170. [CrossRef]
20. Fares, H.; Seminara, L.; Ibrahim, A.; Franceschi, M.; Pinna, L.; Valle, M.; Dosen, S.; Farina, D. Distributed Sensing and Stimulation Systems for Sense of Touch Restoration in Prosthetics. In Proceedings of the 2017 New Generation of CAS (NGCAS), Genova, Italy, 6–9 September 2017; pp. 177–180. [CrossRef]
21. Osta, M.; Ibrahim, A.; Magno, M.; Eggimann, M.; Pullini, A.; Gastaldo, P.; Valle, M. An Energy Efficient System for Touch Modality Classification in Electronic Skin Applications. In Proceedings of the 2019 IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 26–29 May 2019; pp. 1–4.
22. Ibrahim, A.; Gastaldo, P.; Chible, H.; Valle, M. Real-time digital signal processing based on FPGAs for electronic skin implementation. *Sensors* **2017**, *17*, 558. [CrossRef] [PubMed]
23. Jansen, K.; Zhang, H. Scheduling malleable tasks.May 23, 2018 In *Handbook of Approximation Algorithms and Metaheuristics*; Chapman and Hall/CRC: New York, NY, USA, 15 May 2007; pp. 45-1–45-16. doi:10.1201/9781420010749. [CrossRef]
24. Lu, Z.; Rallapalli, S.; Chan, K.; La Porta, T. Modeling the resource requirements of convolutional neural networks on mobile devices. In Proceedings of the MM 2017—Proceedings of the 2017 ACM Multimedia Conference, Mountain View, CA, USA, 23–27 October 2017; pp. 1663–1671, doi:10.1145/3123266.3123389. [CrossRef]
25. Open Neural Network Exchange. Available online: https://github.com/onnx/onnx/ (accessed on 20 November 2019).
26. Intel Movidius NCS2. Available online: https://software.intel.com/en-us/neural-compute-stick (accessed on 20 November 2019).
27. OpenVino Model Optimization Techniques. Available online: https://docs.openvinotoolkit.org/latest/_docs_MO_DG_prepare_model_Model_Optimization_Techniques.html (accessed on 20 November 2019).
28. NVIDIA Jetson Modules and Developer Kits for Embedde Systems Development. Available online: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems (accessed on 20 November 2019).
29. TensorFlow. Available online: https://www.tensorflow.org (accessed on 20 November 2019).
30. NVIDIA TensorRT. Available online: https://developer.nvidia.com/tensorrt (accessed on 20 November 2019).
31. TensorFlow Lite. Available online: https://www.tensorflow.org/lite (accessed on 20 November 2019).
32. Hanif, M.A.; Hafiz, R.; Shafique, M. Error resilience analysis for systematically employing approximate computing in convolutional neural networks. In Proceedings of the 2018 Design, Automation Test in Europe Conference Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 913–916. [CrossRef]
33. Perez, L.; Wang, J. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. *arXiv* **2017**, arXiv: 1712.04621.