Artificial Vision on Edge IoT Devices: A Practical Case for 3D Data Classification

C. Wisultschew, A. Otero, J. Portilla, and E. de la Torre
Universidad Politécnica de Madrid, Centro de Electrónica Industrial
José Gutiérrez Abascal 2, 28006, Madrid, Spain
{c.wpuigdellivol, joseandres.otero, jorge.portilla, eduardo.delatorre}@upm.es

***Abstract*—Nowadays, with the huge advance of sensor technology and the increase of the amount of data generated by them, techniques have to be developed to be able to process all this amount of information in real-time applications on edge devices, close to where data is being generated. If all that information has to be sent to the cloud to be processed, it has certain disadvantages in terms of latency, bandwidth, privacy and reliability, compared to locally processing it on the edge. In this paper, the implementation of deep learning algorithms in low power and limited resources devices in an Internet of Things scenario is studied. In order to work in real-time applications, the influence of different low power consumption deep learning hardware accelerators is studied. Finally, a practical case for smart farming is shown with comparative results in terms of power consumption and performance when running the same artificial vision algorithm on different devices.**

***Index Terms*—Smart Farming, Neural Networks, Deep Learning, Neural Compute Hardware Accelerator, Internet of Things.**

I. INTRODUCTION

Nowadays sensors are becoming increasingly complex, generating a great amount of information. This increase in the amount of data needs advanced techniques to be processed. In the current market, there is a wide variety of complex sensors that generate a large amount of data such as RGB-depth cameras, photonic sensors, LiDAR (Light Detection and Ranging) sensors or hyperspectral cameras, among many others. This large amount of data allows obtaining a large amount of information, which processed in a fast and efficient way enables these complex sensors to be used in IoT applications. Performing the processing of all these sensors in real time on autonomous platforms is a challenge in terms of processing capabilities together with power consumption. One of the aims of this work is to improve the prediction accuracy in artificial vision tasks using complex sensors to take advantage of the large amount of information they provide when compared to traditional sensors.

An artificial intelligence subgroup which works fine with large amounts of information is machine learning. It allows machines to learn from a database in order to be able to generate a prediction for each new input. Machine learning techniques are beginning to be applied on smart farming scenarios both to improve productivity and product quality. The information collected by sensors is processed with machine learning algorithms with the aim of providing crop state information, as well as predicting or detecting possible failures. This is the case of the work in [1], where the information from historical weather is processed using neural networks with the aim of predicting both weather and possible diseases or pests in grape crops, which are very susceptible to environmental changes.

Another trend in smart farming is to take advantage of the IoT to generate autonomous crop systems, connecting all the sensors to each other to process the information in real time, in order to generate decisions about how to activate actuators such as the irrigation circuit, the lighting or ventilation, among others. This is the case of this work [2], in which an autonomous hydroponic closed system is controlled without the need of human intervention.

The proposal of this work is to move the intensive processing of deep learning models from the cloud to the edge [3]. This entails certain advantages, which are:

- Latency: Network latency is currently a bottleneck. As the processing is carried out locally, the information that travels through the network is a small amount of all the raw data generated, allowing the network to be offloaded.
- Privacy: Performing processing in the edge allows not to send critical information through the network, achieving an increase in security.
- Scalability: Adding new devices to the network does not imply an increase in the network bandwidth or in the cloud processing resources.
- Reliability: By distributing computation, a problem on the network does not prevent the correct functioning of the edge devices, as they will perform all the processing inside the same devices without relying on the connection with the cloud. Also, a failure on the network has less impact due to the data can be re-routed through other ways to ensure that the destination is reached.

In this work, the problem of how to efficiently process a huge amount of information coming from complex sensors, such as those used for artificial 3D vision, is studied. However, an increase in computing requirements is usually associated with an increment in power consumption, which is a key factor in autonomous systems. For this reason, the use of novel hardware accelerators with reduced power consumption combined with high performance is considered in this work. As a result, authors propose a methodology to explore the implementation of artificial vision algorithms for 3D object

classification in two different hardware accelerators. On one side there is Intel Movidius Neural Compute Stick based on Myriad2 MA2450 SoC [4] and on the other hand, there is Google Coral edge TPU [5]. These devices have been launched to the market specifically for running deep learning neural networks [6] with reduced power consumption. Experimental results and conclusions are provided in terms of the accuracy in the predictions, the performance, and power consumption. As a practical case, a smart farm is studied, in which can be found autonomous drones, trucks or smart sensors connected between them on an IoT scenario. A large amount of information needs to be processed in a fast and efficient manner.

The rest of this paper is organized as follows. In section 2, the most widely used algorithms in the literature for 3D object classification are described. Section 3 explains the methodology for the implementation of neural networks on low power consumption hardware accelerators. Finally, in section 4, a comparison in terms of performance and power consumption is made for the implementation of a custom deep learning neural network running on different devices. Finally, conclusions are detailed in Section 5.

II. DEEP LEARNING FOR 3D DATA

Nowadays, it is common to find complex sensors such as LiDAR in autonomous systems. However, many systems that process 3D information in real time are not able to take advantage of all the amount of information that can be extracted due to their low computing resources. One of the best ways to extract a large amount of information from this type of sensors are convolutional neuron networks [7]. CNNs rely on convolutional layers to extract representative features of the data. The first convolution is able to detect primitive features such as lines or curves. The more convolutional layers added to networks, the more complex information can be extracted from feature maps. After the convolutional layers, there is typically a final stage called fully connected, that uses these feature maps to provide a prediction. This type of architecture really shines in image, text, and voice classification tasks. The following subsections explain some of the most used neural networks in 3D object classification tasks.

As point clouds have irregular shapes (they are sparse by nature, differently to 2D images, where all the pixels contain information), many solutions in the state-of-the-art preprocess this data before providing it to the network. This aims at increasing accuracy, but at the cost of also increasing the computational requirements. Accuracy represents the capability of our neural network to identify a class of an object from the 40 classes available in the ModelNet40 database. The greater the precision of the network, the more objects it will be able to predict correctly.

A. PointNet

The PointNet [8] is a solution which works in point clouds without previous preprocessing them. For this reason, PointNet achieves very high performance when working with sensors that generate raw point clouds as an output, such as LiDAR. PointNet can be used in object classification, segmentation or semantic analysis of scenes.

As reported in the state-of-the art, this model reaches an accuracy of 89.2 % trained with the Stanford ModelNet40 database [9], which is composed of 40 categories of CAD 3D objects divided into 9,843 for training and 2,468 for validation. A preprocessing is performed to convert CAD objects in the database to point clouds (which would be the output of a real sensor). 1024 points are sampled uniformly on the faces of the mesh according to the area of the face and normalized them on a unit sphere. To improve the accuracy in the classification, PointNet randomly rotates the object on the upper axis and jitters the position of each point with Gaussian noise with zero mean and 0.02 standard deviation.

B. VoxNet

The VoxNet [10] architecture works with point clouds input data format. VoxNet starts by segmenting the point cloud, then converts each of these segments from point cloud to binary voxel (volumetric pixel) format. This format is used as input from the network. This conversion makes possible the acceleration of the processing of the data, due to the fact that being binary, multiplications are simplified to be only by 0 or 1. To do this, first, it carries out the segmentation and then performs the prediction using 3D convolutional neural networks.

This model reaches an accuracy of 83% when trained with the ModelNet40 [9] database. This is a limited accuracy, but this network has the advantage of having the shortest prediction time of all the alternatives, if preprocessing time is not taken into account. This architecture is even faster compared with traditional architectures that work with RGB images for object classification.

C. Frustum PointNet

Frustrum PointNet [11] works with input data from an RGB-Depth camera. The architecture of this network is divided into three parts, starting by applying a convolutional neural network to generate a point cloud with the information provided by the RGB image and the depth map together. Next, the point cloud is provided as an input to the PointNet network, which first segments and then performs the classification. Frustrum PointNet works fine for simple cases with objects that are not very hidden and at a distance that provides a sufficient number of points, and it does not need many points to get a prediction. One of the problems with data coming from RGB-Depth cameras is that accuracy is considerably reduced in low light environments. Another disadvantage of working with point clouds is the fact that a minimum number of points is required to obtain a reliable prediction. Finally, if two objects are very close to each other, results produced by the network can be erroneous.

D. Multi-view Convolutional Neural Networks (MVCNN)

The MVCNN [12] architecture captures virtual 2D pictures from different views of the 3D CAD object, in order to

aggregate the information coming from each picture. This way, it allows obtaining more information than it would be obtained from a single picture. To extract the information, each view i coming from the 3D image is provided to a convolutional neural network i which is specialized in that view i as can be seen in Figure 1. The calculations of each of the CNNs are independent between them and have no time dependencies. In this stage, feature extraction is carried out for each view (using CNN X blocks in Figure 1, which are independent convolutional neural networks). Then all the information coming from each of the views is aggregated to finally give this information as input to the classification stage (fully connected layers named as FC block in Figure 1), in which the final prediction is made.

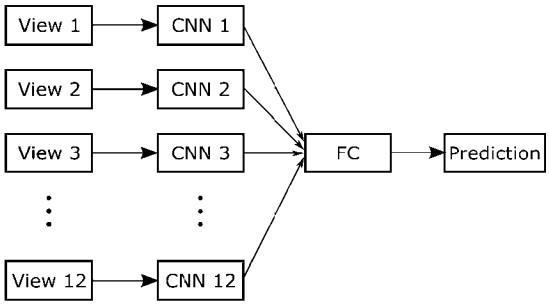

Fig. 1. MVCNN architecture.

Compared to other networks for 3D processing in the state-of-the-art, MVCNN has one of the highest accuracies (90.1 %) in object classification using the ModelNet40 database. The accuracy also depends on the location of the views and the number of views that are used, obtaining a higher accuracy when the number of views is increased. However, when more views are used the increase in computing resources must be taken into account.

E. RotationNet

RotationNet [13] has the same architecture as the MVCNN. The difference lies in the fact that, during inference, only the views that maximize the accuracy in the classification of each object are used. To understand how RotationNet works, let's suppose that 3 views on an MVCNN architecture are available. As shown in Figure 2, all the possible combinations of view configurations are tried with each of the networks in the first stage. This way, the configuration that provides the highest accuracy in the classification can be selected to be finally used. RotationNet is currently the network architecture with greater accuracy in the ModelNet40 Benchmark Leaderboard reaching a 97.37 % accuracy.

F. MobileNet

MobileNet [14] is a image classification deep neural network architecture, which is optimized to be used in edge devices. One of the aims of MobileNet is to allow the implementation of image recognition applications in mobile phones and other handheld devices, which do not have high

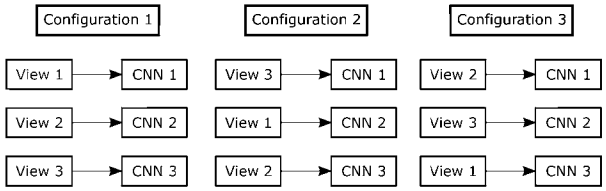

Fig. 2. RotationNet architecture.

computing resources available. The MobileNet algorithm is open source and has several versions with different performances depending on the image size and the accuracy obtained in the classification to adapt to the needs of each device.

Among the architectures studied previously based on point clouds, voxels, and multi-view, RotationNet is the one that achieves the highest accuracy in the 3D object classification task employing the ModelNet40 database. The main disadvantage of RotationNet lies in the inference time for classifying one object, since it is a multiple of the number of views. On the other hand, architectures such as PointNet, MVCNN, or VoxNet are optimized to obtain very low inference times with the disadvantage of not having such a good accuracy when compared with RotationNet. A pre-application analysis must be carried out taking into account the types of data generated by the sensors, together with the limit inference time, the minimum admissible accuracy and the power consumption. In a smart farming scenario, it would be necessary to use networks that provide a very quick response in some critical applications as close as possible to the sensors, such as detecting an object in the path of an autonomous drone or truck. On the other hand, in some tasks it would be more important to obtain good accuracy in the predictions, such as identifying pests or fires.

III. Deep Learning Hardware Accelerators

Nowadays, a large number of low power consumption coprocessors capable of accelerating the processing of deep learning neural network algorithms are emerging. When working with 3D data on edge devices, it is necessary to use this kind of accelerators due to the intense computing processing required. In this section, two of the most widely used edge coprocessors are analyzed, with the aim of showing the advantages and disadvantages of each one.

A. Intel Movidius Neural Compute Stick

Intel® Movidius™ Neural Compute Stick (NCS) is a neural computing accelerator which is available in the market for 70 $ and that has a growing community of users behind it. It was chosen for its low cost and for its capability to implement custom networks to speed them up. Several of these devices can also work in parallel to obtain greater acceleration. Further, they can be implemented in distributed deep learning environments.

Movidius NCS provides low-power, high-performance vision processing solutions. Intel Movidius Neural Compute

Stick includes the SoC Myriad 2 [4] family of vision processing units (VPUs). The Myriad 2 SoC provides solutions across various target applications including embedded deep neural networks, pose estimation, 3D depth-sensing, visual inertial odometry and gesture/eye tracking.

The Myriad 2 MA2450 SoC contains 12 parallelizable vectors cores, each of them working as a Very Long Instruction Word (VLIW) architecture, which allows performing SIMD operations. This processor architecture is designed to perform a large number of operations in parallel, which accelerates, among others, the convolutional stage, composed by multiplications and additions. Furthermore, the SoC contains a Streaming Image Processing Pipeline (SIPP), which has more than 20 programmable hardware accelerators specific for image processing. Some of these are the Harris Corner detector, luminance and chrominance denoising, gamma correction, sharpening filter among others. The communication between the host and the SoC is managed through 2 RISC processors. The Myriad 2 MA2450 SoC has 4 Gbit of LPDDR III RAM.

Movidius provides an SDK to convert networks previously designed in TensorFlow [15] or Caffe [16] (which are frameworks to create, modify, train and inference neural networks) to a Movidius NCS compatible format called graph file. This SDK has the disadvantage of not admitting several operations which can be performed in TensorFlow or Caffe, so implementing complex networks is not an easy task. In the case of networks that use 3D data, it must be notice that Movidius NCS does not support operations working with 3D data.

Once the network in a format supported by Movidius NCS is generated, the API provided by the manufacturer is used to perform the inference. This is done with C or Python languages, which allow to load the graph file in Movidius NCS. In version 2 of the API, specific virtual FIFOs are provided to load the images, with the aim of increasing the performance of this process.

When using Movidius NCS it is necessary to ascertain that the size of the network in memory does not exceed the limit of Movidius NCS which is 320 MB, due to the fact that if it is larger, it brings to inference times which can be up to 50 times greater. This is due to the fact that if this happens, the inference has to be repeated multiple times for the different pieces of the image. This has to be taken into account, since loading and offloading a network takes more time than making an inference, since it is necessary to load in the Movidius memory both the network architecture and the neuron weights.

B. Google Coral edge TPU

In 2016 Google presented the first generation of Tensor Processing Units (TPUs) [17], which are dedicated processors for training complex machine learning algorithms. TPU processors are optimized to accelerate data processing by parallelizing operations as a GPU do. However, the ALUs in a TPU are connected to each other as an array in such a way that data follows a systolic flow. The disadvantage of TPUs is its high power consumption. Google has just presented in 2019 a Google edge TPU machine learning accelerator coprocessor, a low power consumption TPU designed for edge devices. This device works well in neural network applications such as image processing, speech recognition or text processing. The cost of this device is 75 $, it has a USB 3.1 type C connection (5Gb / s transfer speed) and is powered at 5V with a current of at least 500 mA.

IV. IMPLEMENTATION

The motivation for using information in 3D instead of 2D is that a greater amount of information that can be obtained this way, including depth, which is an essential parameter in autonomous systems to accurately determine the distance of the obstacles.

Due to the limited number of supported operations of Tensorflow by the compilers of both Movidius NCS and Coral edge TPU, the MVCNN architecture has been selected to be used as the baseline in this work, and modified using part of the MobileNet v1 architecture, which has Tensorflow operations admitted by both compilers. This way, the unavailability of unsupported operations in Movidius NCS and Coral edge TPU is overcome.

When modifying MVCNN with part of the MobileNet architecture the TensorFlow accuracy is not reduced by working with the ModelNet40 database. This also occurs with RotationNet due to the fact that it has the same architecture as MVCNN. The remaining networks shown in this work were customized employing TensorFlow in order to have operations supported by Movidius NCS or Coral edge TPU. However, some architectures such as PointNet and Frustrum PointNet did not get adequate accuracy. Furthermore VoxNet architecture has some operations such as 3D convolutions that are not replaceable. There are networks that can not be implemented in Movidius NCS or Coral edge TPU, due to there are irreplaceable operations in certain types of architectures. Before implementing a network in Mvoidius NCS or Coral edge TPU it is recommended to make sure if the operations that are in the network are supported by the compilers due to the fact that the information provided by the error messages of the compilers is limited.

A. MVCNN on Movidius NCS

The deep neural network MVCNN is designed and trained using TensorFlow. In order to adapt it to a format supported by Movidius NCS it is necessary to use the Movidius SDK, which does not accept all TensorFlow operations. Unaccepted operations must be removed from the network to be implemented in Movidius NCS. This process is long and many errors arise when using this tool. As a practical case, Boolean variables are not accepted by Movidius SDK, so if the original Tensorflow network contains some Boolean variable it will have to be removed. After customize the MVCNN network using part of MobileNet architecture, it is necessary to check that its accuracy is not affected. In the case of the TensorFlow MVCNN custom network used in this work the accuracies remain closely similar to the original network.

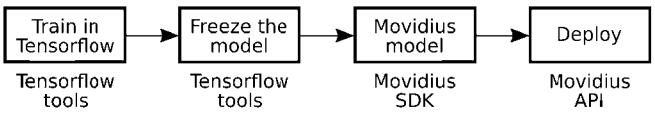

Fig. 3. Movidius Neural Compute Stick Workflow.

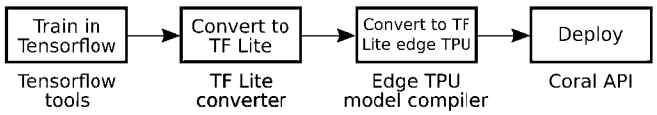

Fig. 4. Coral edge TPU Workflow.

As shown in the Figure 3, the workflow to deploy a deep neural network in Movidius NCS begins with the design and training of the network in Tensorflow. When the model is trained, it has to be frozen, which is a process to save all the parameters that define the architecture and weights of the network in a single file in a way that facilitates its later use. Next, a file that defines the architectures and the weights of the network in Movidius NCS format is generated using the Movidius SDK. Finally, the Movidius API is used to deploy the network in the stick.

When converting the network from Tensorflow to Movidius NCS format, certain modifications are made, such as converting all the variables to `float16` data-type. This might cause a reduction in the accuracy of the predictions, whether the network was designed with larger data types. For this reason, it is necessary to check the validity of the network. Movidius SDK has a tool for this validation, mvNCCheck. With this tool, comparisons are obtained in terms of accuracy between the original network in Tensorflow and the custom Movidius NCS network. It can also be used to quickly check if the network is compatible with Movidius NCS. Using mvNCCheck in our custom network based on MVCNN, it may be verified that the format conversion to `float16` allows an increase in performance, due to the fact that the original network was designed with `float32` data type. However, using mvNCCheck can also be verified that there is no significant reduction in prediction accuracy.

B. MVCNN on Coral edge TPU

The implementation of deep neural networks in Coral edge TPU has the same problem as in the case of Movidius NCS, since the Coral compiler does not support all the operations supported by Tensorflow. As shown in Figure 4 the workflow to achieve the implementation of a neural network in Coral edge TPU starts by training a model in Tensorflow. When it is trained, the model is converted to Tensorflow Lite format using the Tensorflow Lite converter tool. This conversion is performed to optimize the Tensorflow architecture in order to be implemented in devices with low computational resources. Many errors appear in this step due to operations that are not supported by Tensorflow Lite. Once the file is generated, the online web compiler called edge TPU model compiler is used to generate a format supported by Coral edge TPU. Finally, the network is deployed using Coral API.

C. Experimental Results

The MVCNN network has been divided into thirteen parts as can be seen in Figure 1. Twelve of them belong to each of the convolutional networks of each view (CNN X blocks in Figure 1), independent of each other, since each one is specialized in a view. These networks are responsible for taking the input image and generating feature maps at the output. The convolutional layer is the one that requires the most operations compared with fully connected layer. As a result, is the one that takes the most processing time (shown in Table I). Therefore, all the information coming from the feature maps is grouped and given to the last network, the fully connected stage (FC block in Figure 1), which processes all the information, to finally provide a prediction. It must be taken into account that a convolutional layer is carried out for each view, which means that the time will have to be multiplied by 12 if the processing is made in one device. However, the time can be reduced by distributing the processing of each convolutional stage as they do not depend on each other. Table I shows the average processing times in the different parts of the network, each part were evaluated using 10000 samples.

TABLE I
INFERENCE TIMES ON OUR CUSTOM MVCNN USING MODELNET40 DATASET.

Device	Averge time on CNN X layer [ms]	Averge time on FC layer [ms]	Averge total time to make a prediction [ms]
Coral Edge TPU	0.408	0.067	4.963
Movidius NCS	13.94	2.62	169.9

The custom MVCNN network used in this work has been implemented in several devices measuring inference time and power consumption. First of all, it has been implemented on an Intel i7-860 and i7-8700 processors using TensorFlow model, as the role of a traditional desktop computer employing a general purpose processors. On the other hand, it has been implemented on a Raspberry Pi 3 B using TensorFlow model, with the role of a traditional edge device as it has low power consumption and high computing resources based on general purpose processors. Finally, it has been implemented in Movidius NCS and Google Coral edge TPU with the role of low power AI accelerator ASICs. The employed models are custom Movidius Neural Network and TensorFlow Lite specific for edge TPU respectively.

In terms of data types, it must be taken into account that in the original Tensorflow network design, the data are in `float32` format. When using the Movidius NCS the data is converted to `float16` and the Coral edge TPU is designed to use `uint8` data types. In the CPUs (I7-860, I7-8700, ARMv8-A) the data are in the original network format, which is `float32`. Afterwards, it is shown that these changes have repercussions in terms of inference time and accuracy in the prediction.

As it is shown in Table II, when using Coral edge TPU, the shortest inference times are obtained due to, first, to the systolic array architecture and, second, to the data-type used. Working with `uint8` data type performance increases at the cost of providing lower accuracy, due to the quantization of the data that were originally in `float32`. Coral edge TPU has the lowest energy consumption of all the devices used in this work, reaching an average of 0.446 W when making inference on our custom MVCNNN network. As it is shown in Table II, there is another factor corresponding to the number of inferences consuming one W in one second, it can be verified that Coral edge TPU is by far the best.

TABLE II
COMPARATIVE STUDY ON DIFFERENT DEVICES RUNNING OUR CUSTOM MVCNN USING MODELNET40 DATASET.

Devices	Averge time on classify an object [ms]	Averge power consumption [W]	Inferences consuming 1 W per s
i7-860 processor	270.46	95	0.039
i7-8700 processor	80.28	32.8	0.379
Raspberry Pi 3B	1092.23	2.85	0.324
Movidius NCS	169.9	0.98	6.005
Coral edge TPU	4.963	0.446	451.8

Movidius NCS is the second fastest device. There is so much difference between Movidius NCS and Coral edge TPU performance due to the data-type used since Movidius NCS uses `float16`, dissimilar Coral edge TPU that uses 8-bits unsigned integer with the advantage that the accuracy obtained by Movidius NCS is greater than those of Coral edge TPU. The Raspberry Pi 3 has a 1.2GHz 64-bit quad-core ARMv8-A processor and has a power consumption of only 2.85 W, with the disadvantage of being the slowest of all alternatives studied. With regard to CPUs, the core operates at 2.8 GHz in i7-860 and at 3.2 GHz in i7-8700. The biggest drawback of these platforms is their high power consumption, being more than 100 times higher than Coral edge TPU. In the same way, regarding the time to classify an object, it is necessary to know that the CPUs use `float32` as data-type, which reduces the performance and increases the accuracy of the prediction.

V. CONCLUSIONS

Different architectures and hardware accelerators have been evaluated int this work to find an optimal solution for the implementation of deep neural networks working on 3D data in edge devices.

To accelerate processing on edge devices there are two different alternatives available. On the one hand, Movidius NCS, which accelerates deep neural networks maintaining the original precision of the network implemented. On the other hand, there is Coral edge TPU, which achieves 30 times faster acceleration than Movidius NCS running on our custom MVCNN network, with the disadvantage of reducing the accuracy considerably in the network studied. This reduction depends on the neural network architecture to be used. Both solutions can be used together, as could happen in a autonomous drone operating in a smart farming scenario, which needs to detect very quickly if it has any object in its path using Coral edge TPU to avoid the collision with it as quickly as possible. Then, it could analyze more in detail the object that was detected using Movidius NCS to provide a more accurate prediction.

ACKNOWLEDGMENT

This work has been partially funded by Spanish R&D PLATINO project (Ref. TEC2017-86722-C4-2-R).

REFERENCES

[1] S. Sannakki, V. S. Rajpurohit, F. Sumira, and H. Venkatesh, "A neural network approach for disease forecasting in grapes using weather parameters," in *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, July 2013, pp. 1–5.

[2] M. A. Zamora-Izquierdo, J. Santa, J. A. Martínez, V. Martínez, and A. F. Skarmeta, "Smart farming iot platform based on edge and cloud computing," *Biosystems Engineering*, vol. 177, pp. 4 – 17, 2019, intelligent Systems for Environmental Applications. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1537511018301211

[3] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Sep. 2015. [Online]. Available: http://doi.acm.org/10.1145/2831347.2831354

[4] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, and V. Toma, "Always-on vision processing unit for mobile applications," *IEEE Micro*, vol. 35, no. 2, pp. 56–66, Mar 2015.

[5] S. Cass, "Taking ai to the edge: Google's tpu now comes in a maker-friendly package," *IEEE Spectrum*, vol. 56, no. 5, pp. 16–17, May 2019.

[6] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608014002135

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[8] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[9] Zhirong Wu, S. Song, A. Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 1912–1920.

[10] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 922–928.

[11] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[12] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[13] A. Kanezaki, Y. Matsushita, and Y. Nishida, "Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[14] 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," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[15] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, and X. Zhang, "Tensorflow: A system for large-scale machine learning," 05 2016.

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: ACM, 2014, pp. 675–678. [Online]. Available: http://doi.acm.org/10.1145/2647868.2654889

[17] N. Jouppi, C. Young, N. Patil, and D. Patterson, "Motivation for and evaluation of the first tensor processing unit," *IEEE Micro*, vol. 38, no. 3, pp. 10–19, May 2018.