4.3

AI-Based Quality Control System at the Pressing Stages of the Champagne Production

Lucas Mohimont[1], Mathias Roesler[1], Angelo Steffenel[1], Nathalie Gaveau[2], Marine Rondeau[1], François Alin[1], Clément Pierlot[2], Rachel Ouvinha de Oliveira[2], Marcello Coppola[3] and Philipe Doré[4]

[1]University of Reims Champagne-Ardenne, France
[2]Champagne Vranken-Pommery, France
[3]ST Microelectronics, France
[4]CEA-LIST, France

Abstract

Deep learning (DL) is a hot trend for object detection and segmentation, thanks to Deep Neural Networks (DNNs). Image recognition is a powerful tool for precision viticulture, having strong potential in yield estimation and automatic quality estimation of the grapes. However, developing the models is one part of the problem; deploying them in the field, at the edge of the network, is another problem that comes with its own constraints. This paper studies the use of embedded devices to run DNN algorithms for real-time grape segmentation at the wine press. The results show that it is possible to use edge devices while respecting a real-time context with little detection quality losses.

Keywords: grape detection, precision viticulture, deep learning, edge computing, computer vision, object detection, Tensorflow-Lite.

4.3.1 Introduction and Background

Computer vision has helped automate tasks that once required intensive manual labour. For example, it has been used to automatically count fruits and vegetables such as peppers [14] or oranges [11]. Applying this to viticulture is a more challenging problem because each fruit, i.e., the grape, is made of several berries with colours that can vary depending on the variety (white or red) or even resemble the colour of the foliage before the grapes ripen. Nonetheless, detecting grapes automatically is a necessary step for solving other, more complex problems such as yield estimation. Dunn et al. [5] were the first to propose a method for detecting grapes in images. Since then, many methods have been developed to achieve better detection rates and be used on large scales.

Indeed, several approaches can be used to identify the location of grapes on an image. The most intuitive way is by looking at the colour of each pixel, as proposed by Dunn et al. [5]. Unfortunately, this method is sensitive to the lighting condition and cannot be used for different grape varieties: red or white.

Another approach to detecting grapes consists in trying to detect the individual berries as first proposed by [7], which uses the reflection properties of light on each berry. It will produce a specular reflection pattern that follows a Gaussian distribution that can be used to isolate the individual berries that compose the grape. Although it has been implemented in the field and evaluated on a large scale, this approach requires additional equipment (flash or lamp) and works best at night. Therefore, it is not a practical method for use in the field.

Machine learning methods have been proposed to create a binary estimation on each pixel or pixel block of an image. In this case, the selected pixel or block is classified as either a grape or not a grape. Some methods require a feature extraction process before the classification [4][10] and others, based on deep learning, combine the feature extraction and the classification within the same model [3][2]. In this first case, many different features can be used, for example, the average of the RGB channels in the pixel block [10] or the colour histogram [9]. Several classifiers are possible as well, such as the Multi-Layer Perceptron (MLP) [2], Support Vector Machines [4] and AdaBoost [10]. Each method will be a combination of these two different algorithms - feature extractor and classifier. One of the main problems with this approach is that the quality of the classification results depends on the choice of the feature extractor, which in turn depends on

the researcher's choice. One of the main problems with this approach is that the quality of the classification results depends on the choice of the feature extractor, which in turn depends on the researcher's choice.

Convolutional Neural Networks can overcome this problem by combining the feature extraction process and the classification of the extracted features in the same algorithm. Different architectures have been examined with the objective of detecting grapes using transfer learning which yields good results [3]. Some other models have also been explored, such as the Mask R-CNN by Santos et al. [13], Faster R-CNN, R-FCN, and SSD [8]. These methods detect the location of grapes in the image with bounding boxes. Other approaches use deep learning for semantic segmentation to detect individual berries [6][15] or grapevine flowers [12], which we use in this work.

The deployment of deep learning for industrial applications is a challenge. Current deep learning models are trained on powerful GPUs. The challenge is to convert these models for real-time inference on the field. One way to deploy the algorithms is to use specific hardware such as embedded devices, essentially small computers that can operate in remote places like vineyards or wine presses. These small devices have limitations, most notably in computing power and available memory. These constraints must be acknowledged to choose the most suitable tools and algorithms for onboard applications. These constraints include the inference time that must be low enough for practical use. Luckily, a wide range of readily available boards with various capabilities can be used for deploying grape detection algorithms. The option of creating a board for a specific application is always possible.

This paper focuses on detecting unwanted elements (green or ripen grapes, leaves, stones, tools) before delivering the grapes to the press. This paper will be looking at the deployment of a deep neural network for semantic segmentation on a readily available embedded device, enabling AI inference at the edge of the network. Different versions of the model will be tested, and the performances will be analysed based on these three criteria: inference time, performance loss when compared to the original model, and model size. In addition, results must be obtained in less than 15 seconds not to impact the winery production chain.

4.3.2 Methodology

Our team acquired the images for training and testing at the wine press, using a GoPro camera mounted over the weighting device, as illustrated

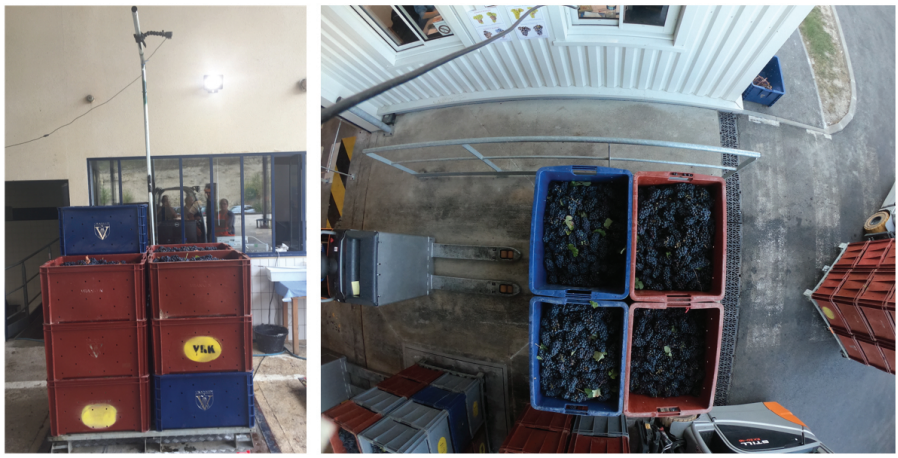

Figure 4.3.1 Image acquisition at the wine pressing sites.

in Figure 4.3.1. Each high-resolution image (4000 by 3000 pixels) covers four crates at the wine press containing grapes and some of the surrounding environment. However, the model can only accept image blocks of 224 by 224 pixels as input. Therefore, the training and validation set images were split into smaller blocks that correspond to 12713 image blocks for the training set and 3250 image blocks for the validation set. The model was implemented in Python using TensorFlow's Keras API and saved in the *hdf5* format. All the weights and biases are stored as 32-bit floating-point numbers (float32 datatype) for this model. Figure 4.3.2 illustrates the image analysis workflow, where a segmentation mask is devised and used to select only the classes we are interested in (in this case, it applies a binary mask to select only grapes).

Embedded devices are not necessarily powerful enough to run an AI model written in Python. For efficiency reasons, the applications that are run on these kinds of devices are usually programmed in a compiled language such as C or C++ instead of an interpreted language like Python. However, to avoid having to rewrite entire models in one of these languages and yet still deploy them on smaller, embedded devices, Tensorflow has created a conversion process to optimise an *hdf5* model developed using their API. The process converts the model into an optimised FlatBuffer by, for example, fusing layers when possible. This conversion aims at reducing the overall model size while trying to maintain the performance of the original model. Different options are available when converting a model from the *hdf5* format

(a) Input image (b) Segmentation mask (c) Segmented image

Figure 4.3.2 Example of the image processing steps.

to the *tflite* format. A commonly used one is the quantisation of the weights and biases to optimise the model. From the original encoder-decoder model, two variants were generated using this converter. Both variants were created with the TOCO converter provided by TensorFlow and saved in the *tflite* format. The first model was converted in the most straightforward manner using the API provided by TensorFlow's version 2.2. No datatype conversion was performed on the weights, biases, or activations for this model, and they maintain their original datatype of float32.

While also being converted using TensorFlow's version 2.2, the second variant took advantage of the post-training integer quantisation process to convert the datatype of the constant tensors (i.e., weights and biases) and the variable tensors (i.e., activations) from float32 to int8. This quantisation process reduces the model's size and memory usage while increasing inference speed, allowing it to run on smaller devices. However, it will inevitably decrease the global performances of the model due to rounding errors that will occur during the conversion. To convert the variable tensors (the output of the intermediate layers), a representative dataset must be provided to estimate the range of the floating-point tensors by running a few inference cycles. A specific dataset does not need to be created for this process; therefore, the representative dataset was generated using the images in the validation set from the original model. Several of the images were cut into blocks of the same size as the model's input. A total of 179 images of 224x224 pixels were included in the representative dataset. Before being used for the quantisation of the model, the images were normalised. This conversion operation is necessary for being able to use the model on a TPU. The accelerator can only run layers that have been converted beforehand. If the entire model is not quantised, then the operations that have not been affected by the process will be run on the CPU.

In this case, all layers were successfully converted, except the first and the final one. As the images are normalised before input, the value of each pixel is a floating-point number between 0 and 1. If the input layer only accepted integers, then the pixel values would be rounded off, and the input image would only contain values of 0, creating a black image and rendering the inference pointless. For the final layer, even though the model's output is a binary mask with integer values for pixels — 1 represents a pixel belonging to a grape and 0 a pixel that does not – the performances significantly deteriorated if the output of the final layer was of type int8. Therefore, the first and final layers were not converted using post-training quantisation.

The device used to run the AI models is an STM32MP157C-DK2 board produced by STMicroelectronics. It has two processors, a dual-core Cortex-A7 32 bits and a Cortex-M4 32 bits. The latest version of the X-Linux-AI package, created explicitly by STMicroelectronics to run AI models on their devices, was installed on the board. This package comes with TensorFlow Lite 2.4.1 and the necessary support libraries for using Coral Edge TPU accelerators. Since it cannot have any version of TensorFlow installed on it, the STM32MP157C-DK2 can only run *tflite* models. Because this board has no dedicated GPU for artificial intelligence, inferences can only be performed using its CPU, which can be pretty slow. Therefore, we equipped the board with a Google Coral USB accelerator (Figure 4.3.3). This tensor processing unit (TPU) is an ASIC processor specifically designed to accelerate the inference of artificial intelligence models, provided as TensorFlow Lite models.

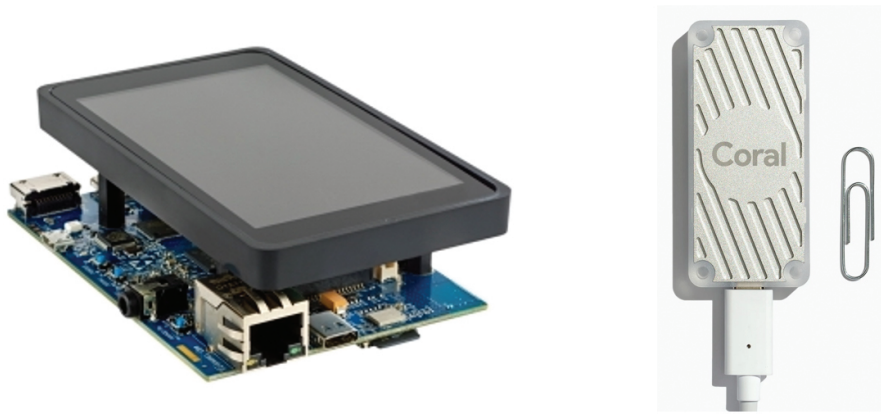

Figure 4.3.3 STM board and TPU accelerator used in this work.

To evaluate the performances, three criteria were used: the inference time, the model's overall size, and the intersection over union (IOU) score. The inference time is used to compare the hardware and software options concerning the real-time constraint that has been set, 15 seconds in this case. The size of the model is given to show how efficient the compression is during the conversion process. Finally, the IOU score was calculated relative to the results obtained with the DGX server. Using this metric, we aim to assess how the inferences from different model versions differ from the original one. Hence, an IOU score of 1 means that the variants' performances are the same as those of the baseline model.

4.3.3 Results and Discussion

The obtained results are presented in Table 4.3.1. Also, Figure 4.3.4 shows an example of the output for each model variant on the STM32MP157C-DK2 board against the baseline output obtained on the DGX server. The inference time and the IOU score presented here were obtained by averaging the individual inference time and IOU score of all the images in the test set. Three tests were run with different models. The STM32MP157C-DK2 board can only run the *tflite* versions of the model (quantised and not quantised) because the board does not support TensorFlow but only the TensorFlow Lite runtime environment. Therefore, the quantised model was run twice, once without using the TPU accelerator and the second time with the accelerator.

The results show that the chosen device is not powerful enough to run the *tflite* model using only the CPU, whether quantised or not and fit the requirements. The quantisation process does allow for a slight decrease in the inference time, of a factor of 1.1 only. This improvement is far insufficient to satisfy the real-time requirements that were set. The only viable solution is to use the TPU accelerator, as the inference time is reduced by a factor of 13 when comparing it with the same model without using the accelerator. Using the accelerator has a drawback as it forces the model to be quantised, inducing a performance degradation as shown by the relative IOU score of 0.93. Considering the significant reduction in the inference time, the slight deterioration of the performances is justified, especially since it is the only scenario that fills the real-time requirements. However, it is interesting to note that since the IOU score is 1 for the non-quantised model, the conversion from the *hdf5* format to the *tflite* does not impact its performances and the only effect is to reduce its overall size. The compression factor between each model is approximately three. More precisely, it achieves a factor of

Table 4.3.1 Results obtained with the STM32MP157C-DK2 board.

	IOU score	Inference (sec.)	Size (in MB)
Baseline	N/A	0.4	93
tflite not quantised	1	137	31
tflite quantised without TPU	0.93	117	8.7
tflite quantised with TPU	0.93	11	8.7

(a) Baseline (b) *tflite* model not quantised (c) quantised without TPU (d) quantised with TPU

Figure 4.3.4 Output of the models.

3 between the original and the *tflite* version and a factor of 3.6 between the non-quantised and the quantised version. Even if the quantisation process impacts the performances, it still allows for complex and heavy models to run on devices with limited resources.

4.3.4 Conclusion

Deep neural networks require large amounts of resources to operate. This is not a problem when deployed on various servers with powerful GPUs; however, this impedes deploying trained models on embedded devices with limited capabilities. To tackle this problem and avoid having to rewrite models in programming languages better suited for smaller devices such a C or C++, different converters exist to reduce the size and the necessary resources for the models to run. These converters allow models initially developed using high-end APIs such as TensorFlow to be easily deployed on boards such as the STM32MP157C-DK2.

In this paper, a deep neural network with an encoder-decoder architecture for semantic segmentation was converted to the *tflite* format, allowing it to run on two small devices. The evaluation proposed in this paper compares three criteria: the inference time, the IOU score relative to the non-converted

model, and the model size. The obtained results are very encouraging. They show that deploying the converted model in a real-time context is possible while limiting the performance losses due to its conversion. Furthermore, the time constraints at the wine press are relatively light, allowing the exploration of model architectures that are not necessarily conceived for real-time applications, such as the original encoder-decoder architecture used in this case study. Nonetheless, this paper gives some insights into the trade-off between performances and inference time when deploying models to smaller devices.

Still, other alternative converters have not been studied here. For instance, the N2D2 platform [1] developed by the CEA-List can convert models from an ONNX format to various targets, including the STM32MP157C-DK2 board. Using N2D2 would bypass the use of Python and TensorFlow Lite by creating a specifically designed project in C. Using this converter may provide better inference time while maintaining the same performances and will be explored in the future.

Acknowledgements

This work is conducted under the framework of the ECSEL AI4DI "Artificial Intelligence for Digitising Industry" project. The project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 826060. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Austria, Czech Republic, Italy, Latvia, Belgium, Lithuania, France, Greece, Finland, Norway.

We also would like to thank the ROMEO Computing Center[1] of the University of Reims Champagne-Ardenne, where the original model was developed and trained.

References

[1] CEA-LIST/N2D2, https://github.com/CEA-LIST/N2D2, original-date: 2017-01-06, Apr. 2021.

[2] N. Behroozi-Khazaei, M.R. Maleki, 'A robust algorithm based on color features for grape cluster segmentation', Computers and Electronics in Agriculture 142, pp. 41-49, 2017. DOI: 10.1016/j.compag.2017.08.025

[1] https://romeo.univ-reims.fr

[3] H. Cecotti, A. Rivera, M. Farhadloo, M.A. Pedroza, 'Grape detection with convolutional neural networks'. Expert Systems with Applications 159, 2020. DOI: 10.1016/j.eswa.2020.113588

[4] R. Chamelat, E. Rosso, A. Choksuriwong, C. Rosenberger, H. Laurent, P. Bro, 'Grape detection by image processing'. IECON 2006 - 32nd Annual Conference on IEEE Industrial Electronics. pp. 3697-3702, 2006. DOI: 10.1109/IECON.2006.347704

[5] G.M. Dunn, S.R. Martin, 'Yield prediction from digital image analysis: A technique with potential for vineyard assessments prior to harvest'. Australian Journal of Grape and Wine Research 10(33), 196-198, 2004. DOI: 10.1111/j.1755-0238.2004.tb00022.x

[6] J. Grimm, K. Herzog, F. Rist, A. Kicherer, R. Töpfer, V. Steinhage, 'An adaptable approach to automated visual detection of plant organs with applications in grapevine breeding'. Biosystems Engineering 183, 170-183, 2019. DOI: 10.1016/j.biosystemseng.2019.04.018

[7] M. Grossetete, Y. Berthoumieu, J.P. Da Costa, C. Germain, O. Lavialle, G. Grenier, 'Early estimation of vineyard yield: Site specific counting of berries by using a smartphone'. In: International Conference on Agriculture Engineering (AgEng), 2012, https://hal.archives-ouvertes.fr/hal-00950298

[8] K. Heinrich, A. Roth, L. Breithaupt, B. Möller, J. Maresch, 'Yield prognosis for the agrarian management of vineyards using deep learning for object counting'. Wirtschaftsinformatik 2019 Proceedings, p. 15, 2019. https://aisel.aisnet.o rg/wi2019/track05/papers/3

[9] S. Liu, S. Marden, M. Whitty, 'Towards automated yield estimation in viticulture'. Proceedings of the Australasian Conference on Robotics and Automation, Sydney, Australia p. 9, 2013.

[10] L. Luo, Y. Tang, X. Zou, C. Wang, P. Zhang, W. Feng, 'Robust grape cluster detection in a vineyard by combining the AdaBoost framework and multiple color components'. Sensors 16(1212), 2016. DOI: 10.3390/s16122098

[11] W. Maldonado, J.C Barbosa, 'Automatic green fruit counting in orange trees using digital images'. Computers and Electronics in Agriculture 127, 572-581, 2016. DOI: 10.1016/j.compag.2016.07.023

[12] R. Rudolph, K. Herzog, R. Töpfer, V. Steinhage, 'Efficient identification, localisation and quantification of grapevine inflorescences in unprepared field images using fully convolutional networks'. arXiv:1807.03770 [cs] pp. 95-104, 2018.

[13] T.T. Santos, L.L de Souza, A.A. dos Santos, S. Avila, 'Grape detection, segmentation, and tracking using deep neural networks and three-dimensional association'. Computers and Electronics in Agriculture 170, 105247, 2020. DOI: 10.1016/j.compag.2020.105247

[14] Y. Song, C. Glasbey, G. Horgan, G. Polder, J. Dieleman, G. van der Heijden, 'Automatic fruit recognition and counting from multiple images'. Biosystems Engineering 118, 203–215, 2014. DOI: 10.1016/j.biosystemseng.2013.12.008

[15] L. Zabawa, A. Kicherer, L. Klingbeil, R. Töpfer, H. Kuhlmann, R. Roscher, 'Counting of grapevine berries in images via semantic segmentation using convolutional neural networks'. ISPRS Journal of Photogrammetry and Remote Sensing 164, 73–83, 2020. DOI: 10.1016/j.isprsjprs.2020.04.002