Accurate and energy efficient ad-hoc neural network for wafer map classification

Ana Pinzari, Thomas Baumela, Liliana Lilibeth Andrade Porras, Maxime Martin, Marcello Coppola, Frédéric Pétrot

Journal of Intelligent Manufacturing
https://doi.org/10.1007/s10845-024-02390-7

Accurate and energy efficient ad-hoc neural network for wafer map classification

Ana Pinzari[1] · Thomas Baumela[4] · Liliana Andrade[1] · Maxime Martin[2] · Marcello Coppola[3] · Frédéric Pétrot[1]

Received: 2 April 2023 / Accepted: 30 March 2024

Abstract
Yield is key to profitability in semiconductor manufacturing and controlling the fabrication process is therefore a key duty for engineers in silicon foundries. Analyzing the distribution of the defective dies on a wafer is a necessary step to identify process shifts, and a major step in this analysis takes the form of a classification of these distributions on wafer bitmaps called *wafer maps*. Current approaches use large to huge state-of-the-art neural networks to perform this classification. We claim that given the task at hand, the use of much smaller, purpose defined neural networks is possible without much accuracy loss, while requiring two orders of magnitude less power than the current solutions. Our work uses actual foundry data from STMicroelectronics 28 nm fabrication facilities that it aims at classifying in 58 categories. We performed experiments using different low power boards for which we report accuracy, power consumption and power efficiency. As a result, we show that to classify 224×224 wafer maps at foundry-throughput with an accuracy above 97% using a bit more than 1 W, is feasible.

Keywords Semiconductor manufacturing · Advance process control · Wafer inspection · Convolutional neural networks · Low-power classification

Introduction

Yield, defined as the ratio of working dies per wafer over the total number of dies per wafer, is a fundamental metric in semiconductor process manufacturing, as it determines the profitability of a production line. Although the quest for high yield has been the focus of process engineers since the infancy of monolithic circuit integration (Murphy, 1964), today's process complexity and feature size makes it more relevant than ever (Park and Simka, 2021).

There are many reasons for which a die might be non functional and determining the precise cause or chain of causes can help yield engineers readjust process steps, be they chemical or mechanical. Early identification of the causes is very important to ensure proper monitoring of the process, improve yield, and overall warranty the sustainability of the fabrication.

Wafer maps are produced by a test equipment during semiconductor manufacturing. During front-end, metrology and defectivity inspections can be used to validate a set of wafers belonging to a batch under fabrication. On the one hand, metrology allows the acquisition of precise integrated circuits measurements, which are analyzed for validating design and

✉ Ana Pinzari
ana.pinzari@univ-grenoble-alpes.fr

✉ Frédéric Pétrot
frederic.petrot@univ-grenoble-alpes.fr

Thomas Baumela
thomas.baumela@elsys-design.com

Liliana Andrade
liliana.andrade@univ-grenoble-alpes.fr

Maxime Martin
maxime.martin@st.com

Marcello Coppola
marcello.coppola@st.com

1 Univ. Grenoble Alpes, CNRS, Grenoble INP, Institute of Engineering Univ. Grenoble Alpes, TIMA, Grenoble 38000, France

2 STMicroelectronics, 850, rue Jean Monnet, Crolles 38926, France

3 STMicroelectronics, 12, rue Jules Horowitz, Grenoble 38019, France

4 Elsys Design, Immeuble Le Leeds, 253 Bd de Leeds, Lille 59777, France

Published online: 01 May 2024

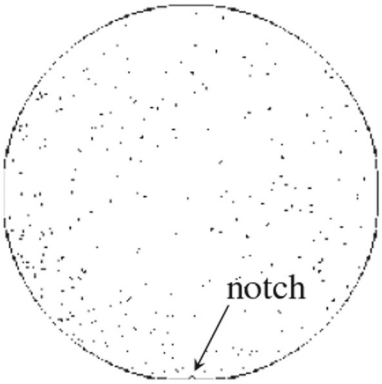

Fig. 1 Typical wafer map with the notch at the bottom. Black dots locate defects

manufacturing rules. On the other hand, defectivity inspection checks for particles or some defects present on each wafer under evaluation. Specialized inspection equipments, based on advanced optical or electron beam inspection systems (Patel et al., 2020), find the position coordinates of each defect present in the wafer. Then, a scanning system generates a two dimensional bitmap image in which the position of the defects is indicated by the black points. The resulting image, see Fig. 1, is called a *binary wafer map* (Hansen et al., 1997).

The orientation of the wafer is known thanks to a tiny notch used for alignment within the equipment. Defective wafer maps reveal patterns formed by the distribution of defects which can be representative of specific machine settings such as alignment, focus, or wafer handling errors such as scratches or localized trouble spots. These wafer maps are of utmost interest for the process engineer, as the different distribution of defects form patterns that denote process dysfunctions root causes (Nag et al., 2022). Coupling defectivity inspection and review scanning electron microscopy image analysis (López de la Rosa et al., 2023) allows for accurate problem identification of these defects.

There has been a lot of work associated to the correct classification of an open wafer map data set recently (Wu et al., 2015). Although very useful for research purposes, our experience has shown that this set of wafer maps has a limited representation of actual processes. It includes far too few categories, a total of nine common classes, and the wafer sizes are taken from different lots, with dimensions varying too much to be representative of a given production line, which is normally built around a specific wafer size. This also led researchers to define various methodologies for reworking and largely preprocessing the data set, and subsequently to propose high-precision neural network architectures, at the expense of the number of parameters. In hardware implementations, a large number of parameters translates into high computational costs and memory resources.

As the test equipment is used 24/7 during semiconductor manufacturing, we are looking for a small neural network, a more application-specific solution than the ones published in the literature, to reduce the cost and energy consumption of the inference process as much as possible. Thankfully, our wafer maps are simple one channel images, this means that we have room for a lot of optimizations, in particular using quantization methods to reduce the necessary resources to perform inference. Additionally, the throughput of wafer production is low enough to give us an extra margin in optimization, trading inference speed for even fewer resources. Overall, our goal is to be able to run the inference task on a low-cost industrial board with a few watts of power budget.

This paper is organized as follows. Section 2 reviews the related works in wafer map classification. Given the huge number of recent contributions to that topic, we focus on the more original works, and the ones that also target low resource usage. Section 3 introduces our data set, that includes 58 categories for a well established 28 nm process on 300 mm wafers. The approach we propose to devise an application specific network is presented Sect. 4, in which we also detail the interest of quantization. We report our experiments Sect. 5, with an emphasis on accuracy and resource usage, followed by a more detailed analysis of the performance achieved per second and per watt in Sect. 6. We wrap-up our work in Sect. 7.

Related works

Although statistical approaches have been used for long to classify wafer maps in high-volume production (Duvivier, 1999), it is only recently that the use of deep neural networks has been proposed to that aim. Specifically, Convolutional Neural Networks (CNN) and supervised methods have been widely used in manufacturing and factory applications during the ten last years (Fahle et al., 2020). CNN architectures have proven to be effective for specific applications such as image recognition or object detection problems in the semiconductor industry. Especially because they have shown an increase of both preprocessing efficiency and accuracy during wafer map pattern recognition and classification (Theodosiou et al., 2023). The emergence of such architectures is motivated by two factors: First the breakthrough brought by AlexNet (Krizhevsky et al., 2012) making deep neural networks clearly superior for image recognition, and second the availability of the WM-811K data set, a public and open wafer map collection donated to the community by Taiwan Semiconductor Manufacturing Company (TSMC).

Wu et al. (2015) introduced the WM-811K data set including 811,457 wafer maps collected from 46k lots in real-world fabrication. Although it is a public data set and it has been subject of many research works, some limitations deserve

mention. First, it is highly unbalanced, i.e. some defects appear very often and others rarely. Second, because labeling by experts is a time and costly consuming process, a large amount of unlabeled data is available, only approximately 20% of data patterns have been annotated. Third, defects are classified only in nine categories, which is far from representing the current sorting process of some production lines. As a result, this data set has been used as reference in many researches leading to the proposal of new learning methodologies, new image preprocessing and data augmentation techniques, as well as the introduction of CNN architectures focused on faulty wafer map classification. Recent and detailed reviews of these works are presented by Kim and Behdinan (2022) and Theodosiou et al. (2023). A common feature of the proposed works is their focus on demonstrating the high level of accuracy achieved by their approaches, without addressing the impact of high computational costs on the scalability and deployment of the deep learning solutions at the edge, i.e. in real industrial environments under time and resources constraints. As far as we know, none of the previous work targets an accurate and energy-constrained hardware implementation for wafer map classification, which is the focus of this paper.

The class-imbalanced issue has long been addressed. Saqlain et al. (2020) implement a data augmentation method to increase the size of minority defect classes. They use random rotations, horizontal flipping, width and height shifts, shearing range, channel shifting, and zooming. For model regularization, authors apply batch normalization and spatial dropout methods. Using a fully-balanced data set and a dedicated CNN architecture, they reach an average classification accuracy of 96.2%. Alawieh et al. (2020) propose a data augmentation scheme built around a convolutional auto-encoder to manage under-represented classes and then integrate selective learning during the classification process. They introduce a rejection option where the model chooses to abstain from predicting a class label when misclassification risk is high, then they achieve 99% of accuracy. Kahng and Kim (2021) demonstrated that the choice of data augmentation is a critical factor governing model performance. Authors expose that augmentation must be chosen carefully regarding the downstream application.

The label uncertainty issue has also been the focus of several researchers. Park et al. (2021) propose a methodology, based on discriminative feature learning and class label reconstruction (using Gaussian means clustering), for creating a new class for defect samples that cannot be categorized into known classes and detecting unknown defects. The average accuracy classifying these unknown defects was improved on 7.8%. Shim et al. (2020) use active learning for decreasing labeling costs. They propose a classification system based on four steps: A LeNet-5-like CNN architecture is trained using the initial labeled data set, an uncertainty prediction in the unlabeled wafer maps is calculated, and using top-K selection methods, a new set of wafers is extracted to be manually inspected by experts and merged with the original data set. Despite the proposed classification system involves a light-parameters CNN architecture, authors evoke that better trade-off between classification accuracy and labeling costs needs to be explored.

Limitations regarding the number of classes of public data set are addressed by Tsai and Lee (2020). They propose a CNN encoder-decoder for data augmentation and a classification method based on depthwise separable convolutions. This work, in addition to the WM-811K data set, uses a 21-defect data set with 16388 real-world maps collected from a Taiwanese company. After data augmentation the classification method reaches an accuracy of 97.01% and 95.09% in the two last data sets respectively.

In our research, we use real wafer data set from a French foundry company, composed of raw data coming from test equipment. Classes are defined by experts and fully-balanced by means of a simulator tool. This data set, including more defect classes than other data sets (58 classes), allows us to train and to fine-tune a light-parameter CNN architecture for an efficient-energy wafer map classification. Our architecture is also tested on the WM-811K data set.

As far as CNN architectures are concerned, the main focus remains on providing high-accuracy models. Nakazawa and Kulkarni (2018) propose a CNN model for image classification and retrieval which, based on synthetic wafer maps, achieves an accuracy of 98.2%. Kyeong and Kim (2018) introduce a CNN architecture composed of a number of individual classification models equivalent to the number of defect classes or patterns that can be detected. This architecture tackles the detection of mixed-type defects, by combining the results of individual classification models, but it entails high computational and storage costs in terms of scalability. Saqlain et al. (2020), introduced above for their data augmentation method, propose a CNN with 2.7 millions of parameters, which has been optimized compared to the reference VGG-16 model (with 134.2 millions of parameters). Wang et al. (2020) and Tsai and Lee (2020) also propose reduced CNN architectures with 2.3 and 1.6 millions of parameters respectively. Although the number of parameters has considerably reduced, these architectures are still too large to be implemented in low-power hardware devices. Andrade et al. (2021) presents experimentations with simplified AlexNet, MobiletNetV1, and VGG, so as to limit the number of parameters they require. This latter architecture leads to the best accuracy while requiring the least parameters. To reduce the number of parameters, other approaches tackle quantization on deep learning models. Nag et al. (2022) performs INT8 quantization on their encoder-decoder architecture, which can perform both classification and segmentation with 98.2% of accuracy. Zhang et al. (2022)

propose a framework based on a binarized neural network. It reaches a classification accuracy of 94.83% and reduces memory requirements by 29.70x. Despite being focused on model size and runtime reductions, none of them refer to power reduction techniques or power/accuracy trade-offs for neural network inference. Only Tsai and Lee (2020) evokes power efficiency in a short paragraph, saying simply that the network fits in an NVidia Jetson Nano board ($\approx$ 10 W) and performs 5 frames per second inference on 64x64 images.

In our research, energy efficiency is the first criterion taken into consideration. Our model was chosen ad-hoc and after experimenting with various architectures based mostly on convolution layers. The analysis of aforementioned work and predefined neural networks led us to converge on a small, precise model that remains largely efficient in terms of power consumption. The model has been specifically developed for our data set, and then tested on the WM-811K data. We focused on CNNs, as these particular architectures remain fascinating for their ability to learn and progressively reduce data. A high accuracy, that is our second criterion, can be achieved by going deeper through the learning kernels.

Data set

Data collection is the starting point of any work making use of neural network. The quality and quantity of the data play a key role in reaching high accuracy during inference (Cortes et al., 1995; Dodge and Karam, 2016). In general, simple problems require a range of thousands of samples, and for more complex problems they can reach several million. To address the problem of data imbalance, the data set must be representative and have a good preponderance of observations (samples) for each type of category (class).

Data collection

The raw wafer data come from STMicroelectronics plant located in Crolles, France, which targets the 28 nm node on 300 mm wafers. An extensive analysis of the faults that can occur during manufacture led to the definition of 58 defect classes. Collecting raw wafers representing all classes is a difficult process, particularly for classes related to rare events during production. To deal with data imbalance, ST uses a drawer tool that can generate synthetic images only to balance the number of samples per class. This process is supervised by process engineers who select the generated images that are the most realistic to ensure that no false cases are included in the training data. Each class is represented by about 2,000–3,000 black and white images resulting in a complete data set of 121,550 400$\times$400 pixel images. Figure 2 shows an example of six of these classes.

Resizing images

Scaling has a major impact on the size of the network and should have no impact on the quality of our data. To determine the optimum scaling rate, we carried out a few experiments. By training a neural network with images of different sizes, we compare the performance of the model. The convergence rate indicates the learning capacity of the network, which in turn depends on the quality of the data. Figure 3 shows the validation accuracy for different images scales. For each scenario, the model is trained on 50-pixel resized images, starting with a 300$\times$300 px data set. We can observe that best performance is clearly achieved for images with higher resolutions, such as 300$\times$300 px and 250$\times$250 px. The smaller the images, the lower the quality and, consequently, the longer it takes to learn the same model. We resized our images to 224$\times$224 px for comparison purposes, as quite a few existing CNN models (VGG16, MobileNetV1, GoogLeNet, etc) use this data size. Figure 3 shows that training the neural network with 224$\times$224 px images leads to high performance and therefore fast learning, converging at almost the same speed as for higher image resolutions. With this size, we ensure that there is no loss of information and that each image subsequently retains the characteristics corresponding to its own failure category. For the resizing method, we applied the nearest neighbor interpolation algorithm McLain (1976).

Building an application specific neural network classifier

Our goal is to build a neural network architecture that is specifically designed and optimized for our classification task. This not necessarily allows us to increase the accuracy but rather gives us a better control on the memory size and computations required at inference time. The first step is to find a suitable architecture and optimize it to get an acceptable accuracy while minimizing storage and computing needs. The second step is to optimize it further by using quantization to reduce weights to 8-bit integers.

Finding a suitable architecture

Our constraints

As we target small hardware devices, we have to find the right balance between resources and accuracy. We perform the search for the appropriate network architecture using floating-point representation, keeping in mind that the parameters size will be reduced by quantization. Our approach is mostly empirical, as the current network architecture search (NAS) approaches are mainly intended for very complex problems and huge networks (Ren et al., 2021).

Motivation and base architecture

Convolutional neural networks (CNNs) have been conceived for high dimensional data. These architectures are notable by their ability to learn spatial features from input

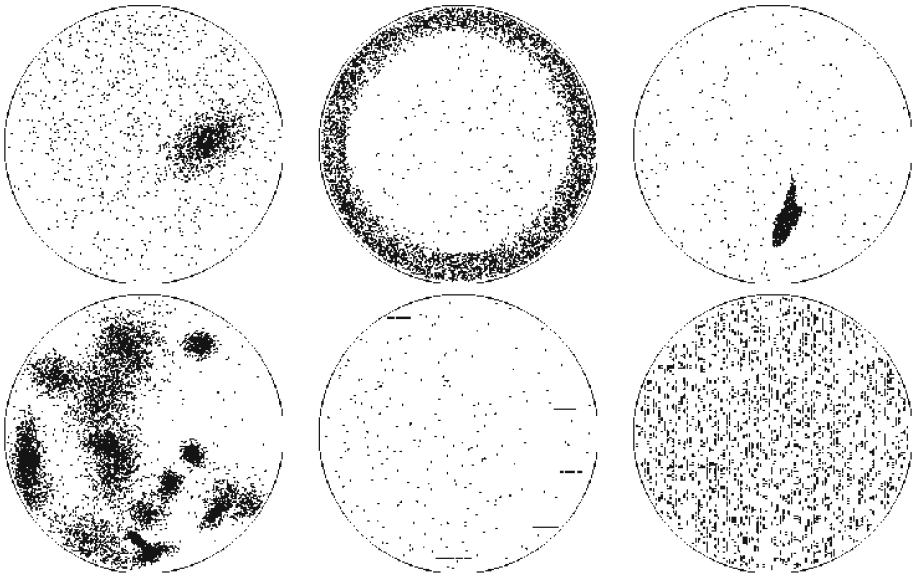

Fig. 2 Examples of wafer map defect patterns. The first line shows the classes "CLUSTER-BIG", "DONUT-EOW-DENSE", "FINGERPRINT", and on the second line "FULL-WAFER", "HORIZONTAL-MULTI" and "MATRIX" classes are shown

images, such as shapes and textures, while gradually reducing their size. As per traditional approach and matter of comparison, we applied some predefined CNNs models such as AlexNet, VGG16, ResNet, GoogLeNet (Szegedy et al., 2015) on our data set. We obtained a good predictive accuracy, the only issue is that the size of these models is simply not appropriate for inference on small electronic devices. When trying to optimize these networks by reducing their size, or by using much smaller networks such as Lenet-5, the accuracy drops at 95.5%. This led us to look for other approaches. Our attention was drawn to the strength of the AlexNet and GoogLeNet architectures. The challenge was to leverage the advantages of these architectures, and to propose a new, less complex neural network for our multi-class classification problem.

AlexNet is among the first networks to adopt an architecture with consecutive convolution layers. A convolution layer transforms the input image into a sequence of feature maps. The deeper the layers, the more feature maps obtained and therefore the more relevant the information obtained. Relevant features from images are basically learned by kernels. The number of kernels should be sufficiently representative for a useful learning. The first convolution layer of the AlexNet architecture starts with large 11×11 kernels, to spot spatial features of the image from a global perspective. These convolutions are interesting but expensive, so we're decided to follow this approach with slightly smaller 7×7 filters and with a stride (displacement between two consecutive convolutions) of 2, instead of 4 used in AlexNet. Taking into consideration that the number of parameters of one convolutional map is determined by the size of the kernel (k_x, k_y), for one channel image $(n = 1)$ and m convolutional maps (kernels) there are $m \times (k_x \times k_y \times n) + m$ (last m being for biases) parameters. For example, the first convolution layer

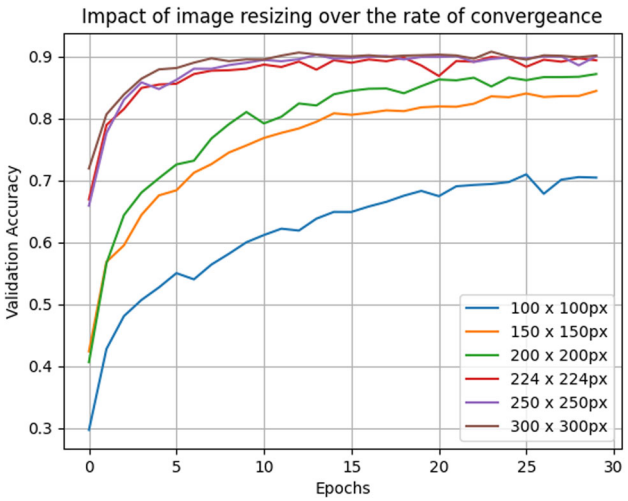

Fig. 3 Comparison of classification performances selecting data sets at different scales. The red curve represents the retained 224×224 resolution

of AlexNet starts with 96 kernels (or neurons) making a total of 17,712 parameters $(96 \times (11 \times 11) + 96)$, in comparison to 1,600 parameters of our proposed first layer (32 kernels with a size of 7×7, and 32 biases).

On the other hand, GoogLeNet presents an interesting trick for analyzing feature maps in depth: different kernels are applied at the same time to the same input data. It introduces the notion of inception module within which there are convolution and maxpooling layers (see Fig. 4). Instead of choosing the size of the filter to apply (1×1, 3×3 or 5×5) an inception module gives the possibility to use all of them together. To offset the number of parameters, the pointwise convolutions are introduced. The 1×1 kernels placed before each 3×3 and 5×5 convolutions are meant to reduce the num-

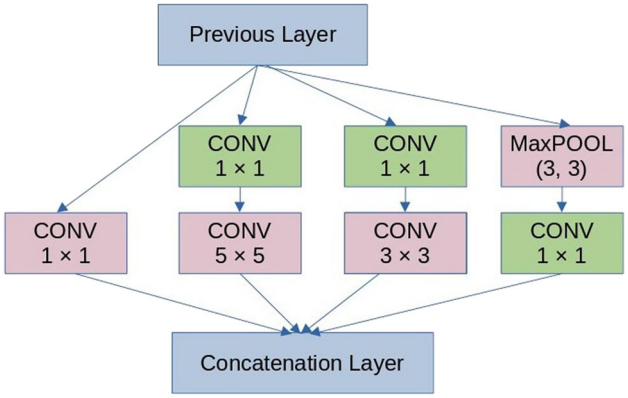

Fig. 4 Inception module diagram

ber of parameters and thus the computational cost. By using multiple kernels, the network learns better by freely applying filters in spatial correlations, which ultimately results in better accuracy. The final volume of all feature maps is due to the end concatenation operation, so the size of feature maps from each lane should remain the same (we do not use any padding).

Method: what modifications we do and why

Searching for a suitable architecture usually follows an empirical way. In our research, we did some intermediate experiments before arriving at the final model. From the beginning, we tried to adapt the AlexNet architecture and reduce the number of parameters (see Table 1). To do this, we completely avoided fully connected layers (FC) and proceeded with a model based only on convolution layers (Test 1). To lighten the topology of our network, we reduced the amount of filters for each layer (Test 2), the introduction of one inception block has improved the predictive accuracy of our model. The path to our final architecture and the above described experiments are summarized in Table 1. For each architecture we show the number of trainable parameters, FLOPs, the accuracy mean and standard deviation for 5 iterations.

Our model architecture

Comparing with AlexNet, which has almost 60 million parameters, and GoogLeNet with 6 million, our ad-hoc neural network has less than half a millon, or 478,150 parameters to be precise. A graphical view of the final architecture is shown Fig. 5.

There are 17 network layers (convolution, subsampling and fully connected layers) of which 10 are learnable layers. The input shape is a $224\times 224\times 1$ image which is subsampled across the entire architecture into a $7\times 7\times 116$ dimensional vector.

Table 2 summarizes the description of our architecture. The model starts with a 7×7 convolution layer, followed by a max-pooling layer and an inception block. Next, the pooling layer maximizes the response of each feature map and reduces the space exploration of feature maps. The 1×1 convolutions reduce the number of parameters, and the last two consecutive 3×3 convolution layers take the role of increasing the search space exploration of feature maps, before being flattened and passed to logistic regression.

For simplicity sake, we use `Relu` (The Rectified Linear Unit function) as activation function, setting at zero negative values and keeping positive values as they are, for all layers but the last one. Since we are facing a multi-class classification problem, the last layer is a dense layer and its activation function is the `Softmax` regression. Softmax produces a probability score for each class and predicts the class with the highest estimated score. For fast convergence the `categorical_crosentroppy` is the loss function used as a feedback signal, the adjustment of weights parameters is carried out by the `Adam` optimizer. These parameters refer to the compilation step.

For the learning algorithm an important aspect is the tuning of hyperparameters. By varying the *batch size* and *learning rate*, the accuracy is remarkably improved. The regularization of hyperparameters is directly related to the control of the optimisation function during the training. For our model, we started with a *batch size* of 512 and a *learning rate* of 10^{-3}, which was successively reduced to 256, 128 till 32 by decaying the learning rate to 10^{-6}. The choice of hyperparameters remains empirical, to the best of our knowledge, there is no exact approach automating the learning process. The hyperparameters are selected according to the specificity of the network architecture and the type of data.

Model evaluation For a better evaluation of our model, we show the results on both our data set and the WM-811K industrial one. Our aim is simply to obtain a better architecture comparison for two valuable data sets, taking into account that the WM-811K wafer maps were taken as-is. The only pre-processing was the resizing of wafers to a resolution of 64×64 pixels, the most common size in the data set, which in fact represents over 90% of all images.

STMicroelectronics Data Set

Figure 6 shows the performance of the model on a balanced subset of 116,000 images. We have chosen 2,000 images per class, representing 95% of the original data set. This balanced subset was then divided into training data (80%) and test data (20%). The subset of 5% of unused original images was kept as validation data. We may observe that starting by the 10^{th} epoch onwards, the loss function on evaluation (test) data stops decreasing. In the field of machine learning, this case is called *overfitting*. Overfitting or overgeneralization is the situation in which the model learns well on the training data, but does not generalize well on the test data. To avoid it, we early stop the training of the model and continue with the tuning of hyperparameters, in our case, by setting the mini-batch size and learning rate to smaller

Table 1 Performances obtained for the ST Microelectronics data set, and for different neural network architectures

Layer	AlexNet	*Test 1*	*Test 2*	Our model
Conv2D	96, (11× 11), s4	96, (11× 11), s4	32, (7× 7), s2	32, (7× 7), s2
MaxPool2D	(3× 3), s2	(3× 3), s2	(2× 2)	(2× 2)
Conv2D	256, (5× 5)	256, (5× 5)	-	**1 Inception**
MaxPool2D	(3× 3), s2	(3× 3), s2	(3× 3), s2	(3× 3), s2
Conv2D	384, (3× 3)	384, (3× 3)	12, (1× 1)	12, (1× 1)
Conv2D	384, (3× 3)	384, (3× 3)	116, (3× 3), s2	116, (3× 3), s2
Conv2D	256, (3× 3)	256, (3× 3)	116, (3× 3), s2	116, (3× 3), s2
MaxPool2D	(3× 3), s2	(3× 3), s2	-	-
FC/Dropout	4096 / 0.5	-	-	-
FC/Dropout	4096 / 0.5	-	-	-
FC	58	58	58	58
# params	58,495,738	4,258,554	465,590	478,150
# FLOPs	2,126,591,932	2,018,125,756	56,258,108	125,518,940
Accuracy mean (%)	97.16	96.94	95.09	99.89
Std. Dev. (±)	0.07	0.07	0.11	0.01

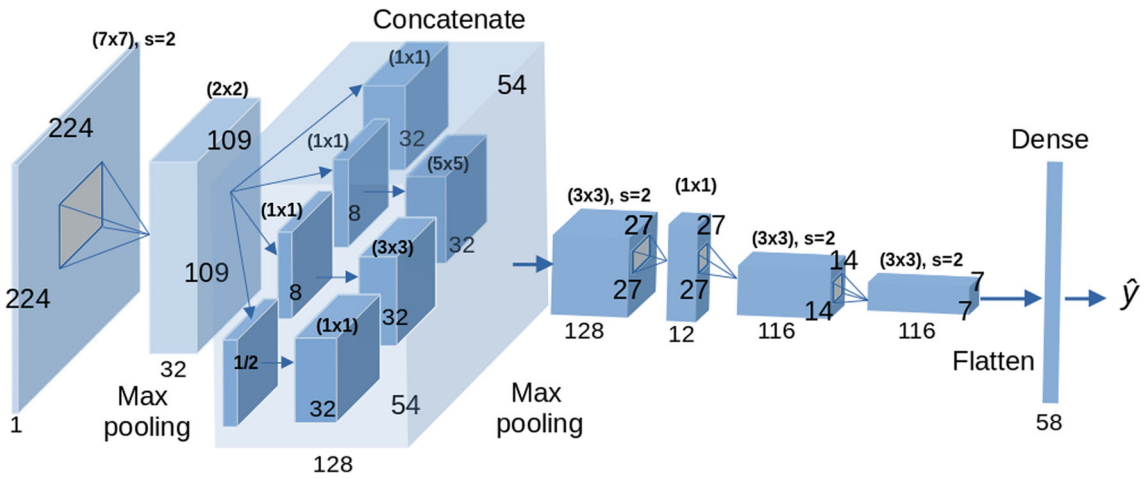

Fig. 5 Proposed neural network architecture

Table 2 Description of the model architecture

Layer	*k (n × n) means k kernels of size n × n*
Conv2D	32 (7× 7), stride of 2, no padding
MaxPool2D	(2× 2), stride of 2
Inception Block	32 (1× 1), 8 (1× 1), 8 (1× 1), MaxPool (3× 3)
	32 (3× 3), 32 (5× 5), 32 (1× 1)
MaxPool2D	(3× 3), stride of 2
Conv2D	12 (1× 1)
Conv2D	116 (3× 3), stride of 2, padding with zeros
Conv2D	116 (3× 3), stride of 2, padding with zeros
FC/Softmax	58
Number of parameters	478,150

Table 3 Top-1 Accuracy on the STMicroelectronics data set (testing and validation data)

	Testing Data	Validation Data
Nb of images	116,000	5,550
Top-1 accuracy (%)	99.922	99.935
Loss	0.0049	0.0051

values. In both graphs, we observe that from the 20th epoch onwards, the model learns well again and the evaluation on the test data increases in accuracy. Moreover, the descent of the loss functions decreases continuously until the end of the learning phase. We reach a testing accuracy of our model of 99.92%.

Table 3 shows the performance of our model on testing and validation subsets.

WM-811K Data Set

The WM-811K data set has a total of 811,457 wafer maps, with 172,950 labeled wafers: 25,519 are raw wafer maps with a real pattern belonging to 8 categories *Center* (4294 images), *Donut* (555 images), *Edge-Loc* (5189 images), *Edge-Ring* (9680 images), *Loc* (3593 images), *Random* (866 images), *Scratch* (1193 images) and *Near-full* class (149 images); and 147,431 mixed-type failure wafers, labeled as class 'None'. Each one of these wafer maps was collected from a real-world fabrication line.

In our experiments, all wafer maps were resized to 64× 64 pixels and the data set was set to 32,891 samples (all samples of the 8 classes and 5% of samples from class 'None'). For this experiment, we consider the following method: for the first convolution layer it makes sense to take a stride of 1 (instead of 2, as applied for our wafer maps). As we are using much smaller input wafers (64× 64), there is no need to reduce relevant information, especially when it comes to the first layer. The rest of the architecture remains intact and is applied as described in Table 2. With this method and for this data set, the testing accuracy is 96.63%. Another experiment was carried out on the collection of 25,519 wafers belonging to all 8 classes. Each wafer map has a single defect, and each wafer corresponds to a single class, which explains why the testing accuracy of our model is higher, at around 99.53%.

In both cases, we obtain a small model with 163,276 parameters for the 8 classes, and 165,133 parameters for a 9-class classification.

Figure 7 shows the confusion matrix with predicted scores for (a) 9 classes (32,891 samples), and (b) 8 classes (25,519 samples). We can see that the model's performance is not so much affected by the number of samples per class (the 'Near-full' class is an under-represented class, with only 149 samples), but rather by the spatial textures of the defects, which either have a similar or a less defined criterion. Classes with an exact defect shape such as 'Donut', 'Center', 'Edge-Ring', 'Near-Full' are the classes for which the model classified best. In this experiment, we can also observe the difference in model prediction when considering single-label classes (one-to-one correspondence), and data including samples of undefined failure type (such as the type of class 'None'). This class can be considered for multi-label classification task, as it contains wafer maps with similar defects to other classes. For example, 141 images from the 'None' class were misclassified in the 'Scratch' class, 89 images in the 'Loc' class and 96 in the 'Edge-loc' class. On the other hand, the introduction of the 'None' label in the training set allows the model to learn better about outliers. This may be of considerable practical interest when, during wafer production, new defect patterns may emerge and it would be useful to lead the model to generalize better on these test data.

Now that we have a model with high predictive accuracy, we will proceed with optimization techniques regarding computational and memory requirements necessary to enable the execution of our model on small edge devices.

Quantization principles

Quantization consists of reducing the number of bits necessary to represent a value. Its use in neural networks is not new (Dundar and Rose, 1995; Hoskins et al., 1995), but using it on deep convolutional neural network raises new challenges. There are now many different quantization approaches, ranging from quantizing only the parameters, quantizing both parameters (often only weights, not biases) and activations, quantizing on 16, 8, or even 2 or 1 bit, etc. The approaches using the smallest bit sizes are meaningful for hardware implementations only (Andri et al., 2016; Umuroglu et al., 2017; Zhao et al., 2017; Prost-Boucle et al., 2018) to name a few. For the sake of this work, which targets of-the-shelf micro-controller based boards, we will restrict ourselves to an 8-bit quantization of the weights that is well suited to byte based computation in software, or with existing hardware accelerators (either ad-hoc or performing matrix–vector or matrix-matrix multiplications). As a result, the most demanding part of the neuron output computation ($v_j = \sum_{i=0}^{n-1} x_i w_{ij}$) uses only 8-bit integer multiplications. This is key because the area and power complexity of a multiplier is in $O(b^2)$ where b is the number of bits of the inputs. Each multiplication produces a $2b$-bit result, that is accumulated with the adder to produce a $(2b + \log_2 n)$-bit result, n being the number of inputs of the neuron. Using a 32-bit addition is a safe guess here, as there are very few chances that the accumulation takes place with more than 2^{16} inputs. It is also safe to have a bias b_j on 32-bit, as this is a single addition performed after all integer multiplications ($o_j = v_j + b_j$).

As Tensorflow has been the first widely available framework to provide fine-tuned 8-bit integer arithmetic imple-

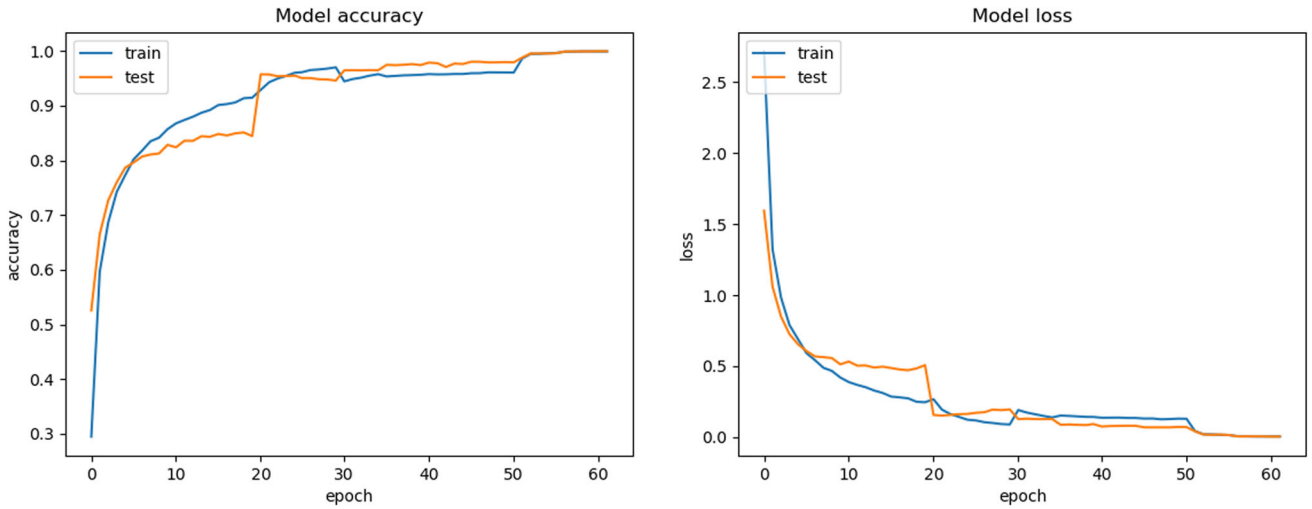

Fig. 6 Model scores: the evolution of testing accuracy and loss during the learning phase (STMicroelectronics data set)

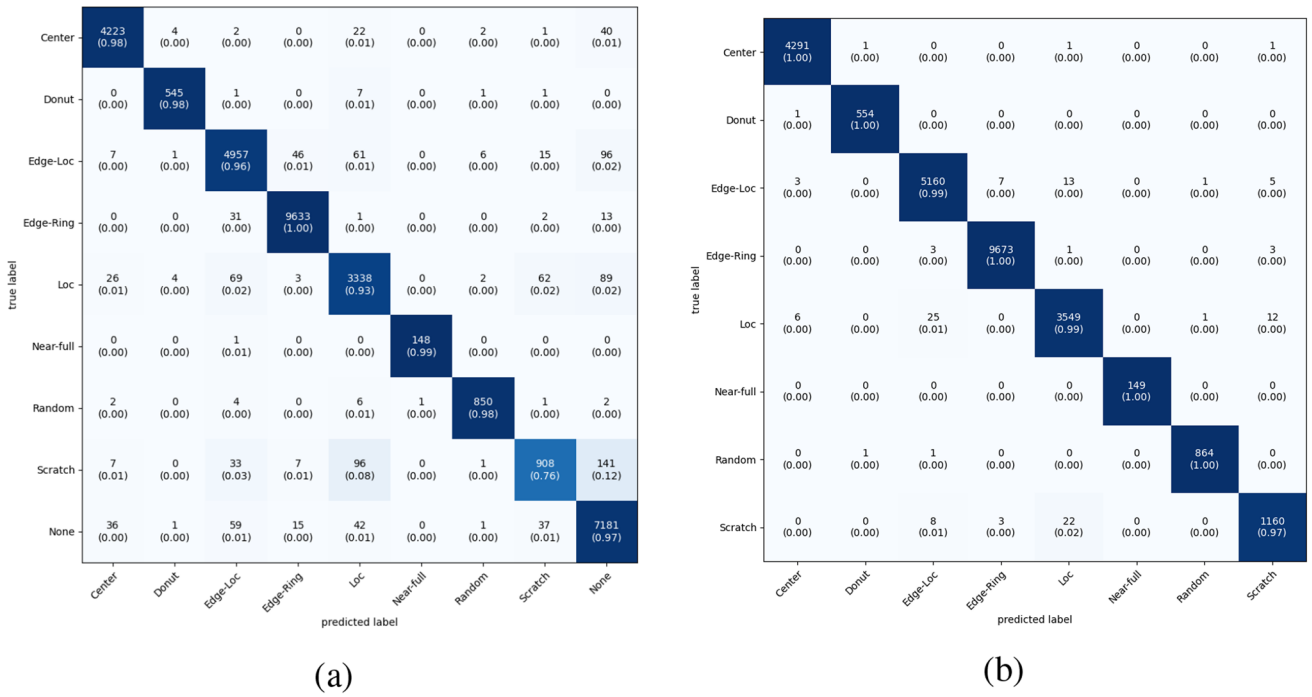

Fig. 7 Confusion matrix for WM-811K data set based on (a) 9 classes and (b) 8 classes

mentations for micro-controllers (using e.g. SIMD instructions) and Google TPU (Jouppi et al., 2021), we opted for using it given our high power-efficiency goal. We briefly summarize here the quantization approach that is advocated by and implemented in this framework, which is thoroughly detailed in (Jacob et al., 2018). For a given convolution layer, the quantization process produces in addition an offset (called zero-point, zp), and for each output channel of the layer a scale under the form of an integer multiplicand M and a shift s. The scale factor and offset must be applied before the activation function, leading (roughly, as the idea is to divide by 2^s which is not a raw shift for negative values) to $y_j = ((o_j \times M) \gg s) + zp$. These operations, done only once per kernel, typically fit on 32-bit, and the final result is saturated to -128 or 127.

From a practical perspective, there are two main ways for quantizing a network: Post-training quantization (PTQ) and quantization-aware training (QAT). PTQ consists of finding offsets and scale values to approximate the weights of an already trained network. Post-training works quite well on large networks, especially when lowering weight size to 8 bits or above. To further reduce bit size without incurring high

accuracy losses, it is usually necessary to use QAT. This consists of training the network by taking into account the low precision behavior during the process.

Google's TensorFlow-Lite (TF-Lite) open source framework provides an API to convert and interpret quantized networks. Given our target that is micro-controllers possibly backed by an accelerator, for which lower than 8-bit precision is useless, we use the post-training quantization method. It produces weights and biases quantized to a fixed-point precision of 8-bit using the approach mentioned above, and required by integer-only accelerators. PTQ takes a fully trained model and doesn't require additional modifications for conversion into a quantized model. Nevertheless, an important point for the conversion process is to provide a representative data set, i.e. a small subset of the original data set which covers the entire value space. This gives the quantization process the range of inputs values and it can then find the most appropriate 8-bit representation (multiplicand M and shift s) for each weight and activation value. To achieve the best possible performance, i.e. ensure that all computations are done using SIMD instructions or outsourced to the TPU, it is recommended to strictly stick to the 8-bit data type. For this purpose, we perform a full integer optimization with the TF-Lite converter, i.e. the inputs and the outputs use 8 bits.

The accuracy with the quantization process activated is given in Table 4.

Experiments

The experiments are divided into three sections. The first one focuses on accuracy, validating the accuracy stability and analyzing the confusion matrix to identify problematic classes. The second one shows an analysis of the performances both on float and integer implementations over a representative panel of low-cost development boards. Finally, the third section details the power efficiency analysis of our board panel, comparing possible use-case minimizing the power footprint.

Accuracy results and analysis

We first evaluate the accuracy of our CNN architecture in a classical floating-point representation. To that aim, we use the statistical *K-fold cross validation* method. The method consists of dividing the data set into k batches of equal size and use at each iteration *k-1* batches as training data and only one batch as test data. The division into k batches (folds) means that we have k evaluations. Figure 8 shows an example of applying this method by setting $k = 3$. For each evaluation the training set (blue color) is iteratively represented by *k-1* batches and one batch is retained for the testing (green color).

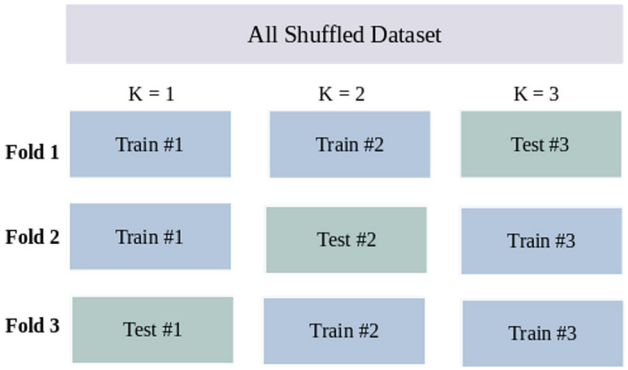

Fig. 8 Example of K-Fold cross validation method

Table 5 shows the performance obtained by the model for 5 iterations. For each iteration, the network was trained on training data and evaluated on validation data. An average score is given at the end of each iteration to provide a better comparison of accuracy when evaluating the model.

Confusion matrix

K-fold cross-validation is an efficient and reliable way to evaluate our approach by varying the distribution of the data at each iteration and obtaining different scores. Nevertheless, the confusion matrix is the ideal tool to visualize the performance of our classifier for each class separately. As the confusion matrix for ST data set has a large size, we represent the results of the classifier as a graph.

Figure 9 shows the distribution of only incorrectly classified images among the 58 classes. Per 2,000 images representing each class, we show in orange the number of images that are classified differently from the actual class (False negatives, FN), in green the images that were classified as the actual class while they actually belong to other classes (False positives, FP). In the graph, the FN and FP data are given for each class (from 1 to 58) and at the same scale.

To understand where the classifier fails, let's examine the most wrongly predicted classes: 19, 20, 32, 35, 54 and 56. Classes 19 "EOW-EXTREME" and 20 "EOW-EXTREME-LIGHT" are very close in terms of defects; they are almost the same class with a more or less important degree of repetition of failures. In addition, a number of 19 "EOW-EXTREME-LIGHT" images were classified as class 32 "RANDOM". This is the class (see Fig. 9, class 32) with a rather trivial failure type, *i.e.,* a random distribution of failure patterns, reason for which the classifier confused with classes representing similar defects. In the graph, 33 images (FP shown in green colour) were found as "RANDOM" class by the classification algorithm. Classes 54 and 56 represent vertical failures with different location spots: 54 "VERTICAL-3 H"

Table 4 Inference accuracy of the quantized model before (QAT) and after (PTQ) training

	Quantization-aware Training	Post-training Quantization
Accuracy (%)	97.63	97.35

Table 5 K-Fold cross validation method for validation data set

	Iteration 1		Iteration 2		Iteration 3		Iteration 4		Iteration 5	
%	Loss	Acc	Loss	Acc	Loss	Acc	Loss	Acc	Loss	Acc
Fold 1	0.0065	99.88	0.0056	99.87	0.0045	99.91	0.0066	99.88	0.0048	99.93
Fold 2	0.0054	99.87	0.0057	99.89	0.0063	99.89	0.007	99.88	0.0053	99.91
Fold 3	0.0058	99.89	0.0057	99.89	0.0058	99.93	0.0039	99.93	0.0065	99.87
Avg	0.0059	99.88	0.0058	99.88	0.0057	99.89	0.0057	99.89	0.0057	99.89

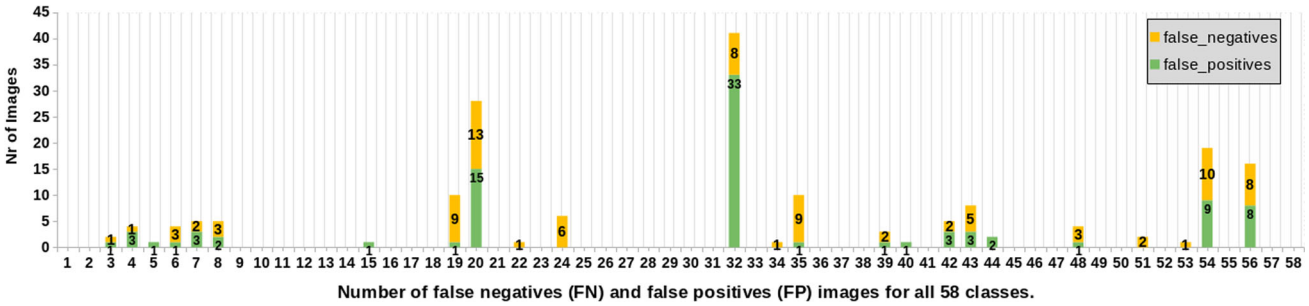

Fig. 9 Confusion graph for 58 classes. The false negatives images are shown in orange color and the false positives in green

(right side), and 56 "VERTICAL-9 H"(left side). To our surprise, the model has actually classified these images very precisely. The rotation method used to increase the data set led to an accidental misslabelization of these samples, classifying them in the opposite class

A detailed analysis is presented in Table 6, where we show the performance metrics of our model, such as precision, recall and F1-Score for each class of the STMicroelectronics data set. Classes with the highest number of classification errors are marked in Bold.

Time and power evaluation of the model

With an accurate model in our hands, we must validate that the throughput yield on low-cost hardware is high enough to follow the pace of a wafer production line. Typically, we aim at 1 inference per second, a high enough throughput to follow modern line paces within a comfortable margin.

These experiments are conducted using software implementation of our quantized neural network model 4.2 as well as the unquantized version. They are each using the available kernel implementation provided with their development kit without neither modification nor optimization from our side. Further optimization is surely possible, though we show that solely optimizing the neural network model is enough to deliver the required performances using general purpose hardware.

Experiments are conducted on the following hardware targets:

- X86 Desktop CPU 48 cores / 96 threads (float and int)
- Google Coral CPU quad Cortex-A53 (int and float)
- Google Coral TPU V1 coprocessor 4 TOPS (int)
- Jetson CPU Quad Cortex-a53 (int and float)
- Jetson Maxwell GPU, 128 CUDA cores (float)
- STM32MP1 CPU Cortex-A7 (int and float)

Figure 10 describes the workflow to create a Tensorflow Lite model for inference on the above mentioned edge devices. Our conversion focuses on creating a quantized model that can be realized either using floating-point values which target CPU and GPU, or an 8-bit fixed point model, for CPU and TPU acceleration. For optimal use of Coral's TPU, the tflite model must be compiled at the end with the edgetpu compiler to check the compatibility of the quantized operations and then map them onto the TPU.

Once we have the models, we analyse the real-time performance of our model for different systems. The experiments target the number of inferences our model can perform per second, by measuring the latency for different scenarios: unquantized tensorflow model (binary32[1]), tflite model

[1] The *binary{256/128/64/32/16}* types correspond to the floating point representations defined in the IEEE 754-2008 standard on the number of bit indicated in their name.

Table 6 Performance metrics for STMicroelectronics data set

Class	Precision (%)	Recall (%)	F-Measure (%)	*Class*	Precision (%)	Recall (%)	F-Measure (%)
1	100	100	100	*30*	100	100	100
2	100	100	100	*31*	100	100	100
3	99.95	99.95	99.95	*32*	99.60	98.37	**98.98**
4	99.95	99.85	99.90	*33*	100	100	100
5	100	99.95	99.98	*34*	99.95	100	99.97
6	99.85	99.95	99.90	*35*	99.55	99.95	99.75
7	99.90	99.85	99.88	*36*	100	100	100
8	99.85	99.90	99.87	*37*	100	100	100
9	100	100	100	*38*	100	100	100
10	100	100	100	*39*	99.90	99.95	99.92
11	100	100	100	*40*	100	99.95	99.98
12	100	100	100	*41*	100	100	100
13	100	100	100	*42*	99.90	99.85	99.88
14	100	100	100	*43*	99.75	99.85	99.80
15	100	99.95	99.98	*44*	100	99.90	99.95
16	100	100	100	*45*	100	100	100
17	100	100	100	*46*	100	100	100
18	100	100	100	*47*	100	100	100
19	99.55	99.95	99.75	*48*	99.85	99.95	99.90
20	99.35	99.25	**99.30**	*49*	100	100	100
21	100	100	100	*50*	100	100	100
22	99.95	100	99.97	*51*	99.90	100	99.95
23	100	100	100	*52*	100	100	100
24	99.70	100	99.85	*53*	99.95	100	99.97
25	100	100	100	*54*	99.50	99.55	**99.52**
26	100	100	100	*55*	100	100	100
27	100	100	100	*56*	99.60	99.60	**99.60**
28	100	100	100	*57*	100	100	100
29	100	100	100	*58*	100	100	100

Top-1 Accuracy: 99.92%

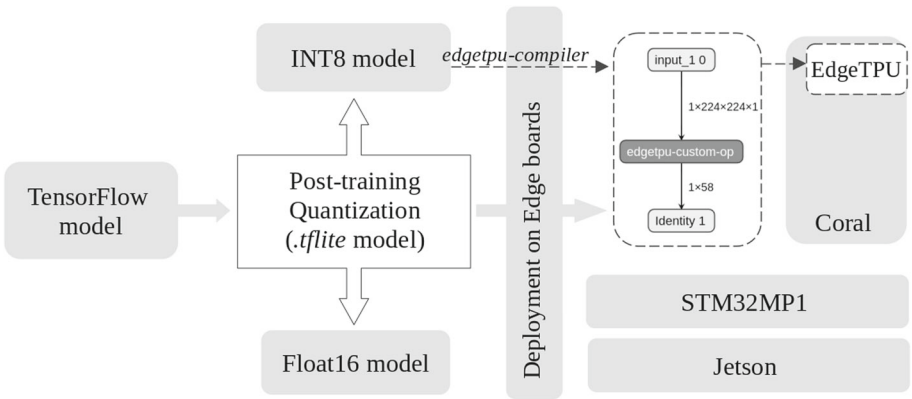

Fig. 10 The workflow for creating a tflite model (int8 and binary16) for inference on edge boards: Google Coral including the compiled model for the EdgeTPU, STM32MP1 and Jetson

Table 7 Inference performance and latency measurements for randomly selected images. Experiments were done on x86 standalone server, Google Coral, STM32P1 and NVIDIA Jetson boards

Performance (inferences/s)					
	CPU			Accel.	
	binary32	binary16	int8	TPU	GPU
x86	52.5	322.5	312.5	-	-
Coral	-	20	31.8	902	-
MP1	-	4.5	5.5	-	-
Jetson	26	38	56	-	47
Latency (ms)					
	CPU			Accel.	
	binary32	binary16	int8	TPU	GPU
x86	19	3.1	3.2	-	-
Coral	-	49.4	31.4	1.11	-
MP1	-	223	181	-	-
Jetson	38.5	26.4	17.8	-	21.2

(binary16 and int8) and edgetpu model (int8). Inference is performed one image at a time, i.e. the batch size is set to 1.

As shown by the conducted experiments reported Table 7, all our targets can meet the requirements in terms of production line throughput. An x86 CPU desktop machine uses binary32 floats by default to infer a wafer map. With quantization, there is a gain in memory resources and therefore a higher inference speed, with no visible loss in precision. The MP1 board performs faster for integer arithmetic, being dedicated to real-time low-power tasks, its floating-point unit implementation favored low power and low area over peak performance. For the Coral SoC, the best performance is achieved by the TPU ML accelerator, the performance is more than 30x higher (902 i/s) than on its CPU. The Jetson CPU shows good inference performance for models at half precision. The binary16 operations are faster than the binary32 ones, so these quantized models should be preferred on this device. Regarding the GPU part, a batch size of 1 is not at all enough to benefit from its computational power.

The following experiments focus on power consumption. They are performed on a batch size of 100 images and within the range of 1 to 32 batches processed at a time.

Using coral board

Figure 11 shows the performance achieved by the TPU and the CPU of the Coral board. We can observe that for large batch sizes, the TPU hardware accelerator achieves performance up to 1600 inferences/s for a power consumption of 4.2 W. Running the tflite model on the CPU (ARM vector instructions), and without edgetpu optimization, we obtain a performance of 33 inferences/s (ips) for the int8 model leading to a power consumption of 4.3 W, and a lower consumption of 3.8 W for the binary32 model, with 21 ips. In the power curves, we can observe a repetitive power overshoot of a bit less than 1 W per batch. This is due to the cooling fan that starts when using larger batches. Note that for inferences at a batch size of 1, the fan was never activated.

Using STM32MP1 board

The STM32 MP1 board targets low power industrial applications. The throughput of the floating-point model improves when we increase the batch size, is still lower than the integer one. For the integer model, there is not much improvement in performance, see Fig. 12. We can also report that it was not possible to exceed a batch size of 32 with floats due to memory limitations. But we were able to go up to batches of 128 for 8-bit integers due to their much smaller memory footprint.

Using NVidia Jetson

Figure 13 shows CPU float experiments with two inference kernels. One is the tensorflow base interpreter, the other is the tensorflow lite implementation. Both have similar throughput (a little lower for tflite) but there's a non negligible change in power consumption going from 5W to 3.5W. The latter being close to integers which are even more interesting with a little more throughput for a little less power consumption. Note on the little "break" in GPU curve: It occurs at a batch size of 128 which is the number of cuda core to feed with images. This is why we lose some throughput at 129 before slowly catching up the maximum throughput.

The maximum system latencies measured led us to conclude that per batch inference is preferable over inferring wafer maps as they come. In Sect. 6, we confirm these observations taking into account the performance obtained both per second and per Watt.

Power efficiency analysis

All boards achieve an accuracy of 97% confirming the numbers we measured on the host using the Tensorflow framework. About inference time, the Coral reaches, thanks to its TPU, a rate of $\approx$ 900 inferences per seconds, while the Jetson with its GPU plateaus at 250. The MP1, without hardware support for inference, attains 5.5 inferences per seconds. These results are good regarding our target problem, and are actually very satisfying considering the size of the data (224$\times$ 224 pixels per image) and the number of classes (58). Real-time for wafer manufacturing means that we must be in line with the throughput of the defect inspection equip-

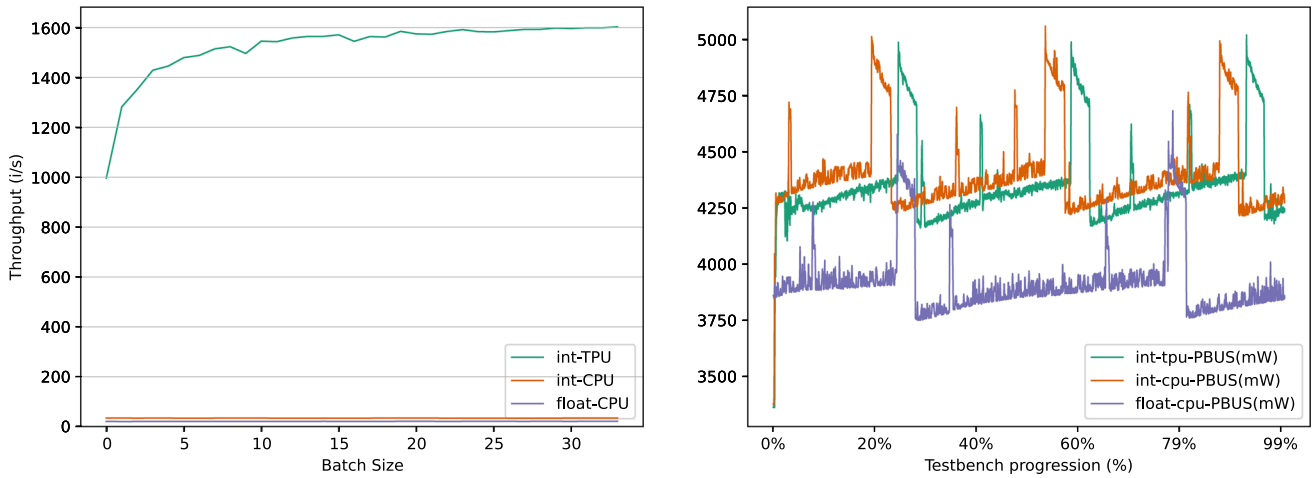

Fig. 11 Coral performance and power measurements

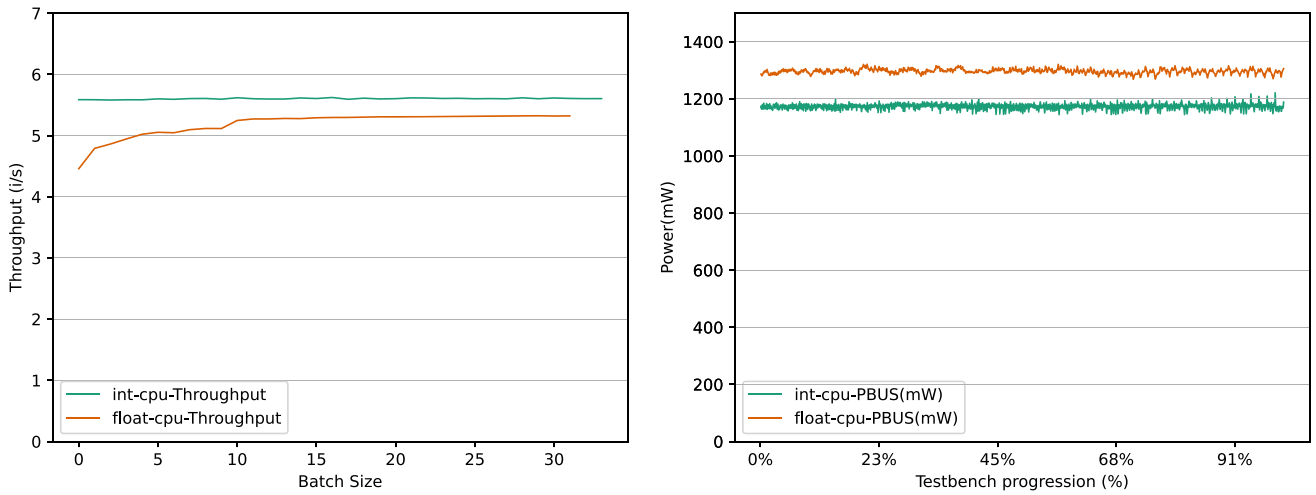

Fig. 12 MP1 performance and power measurements

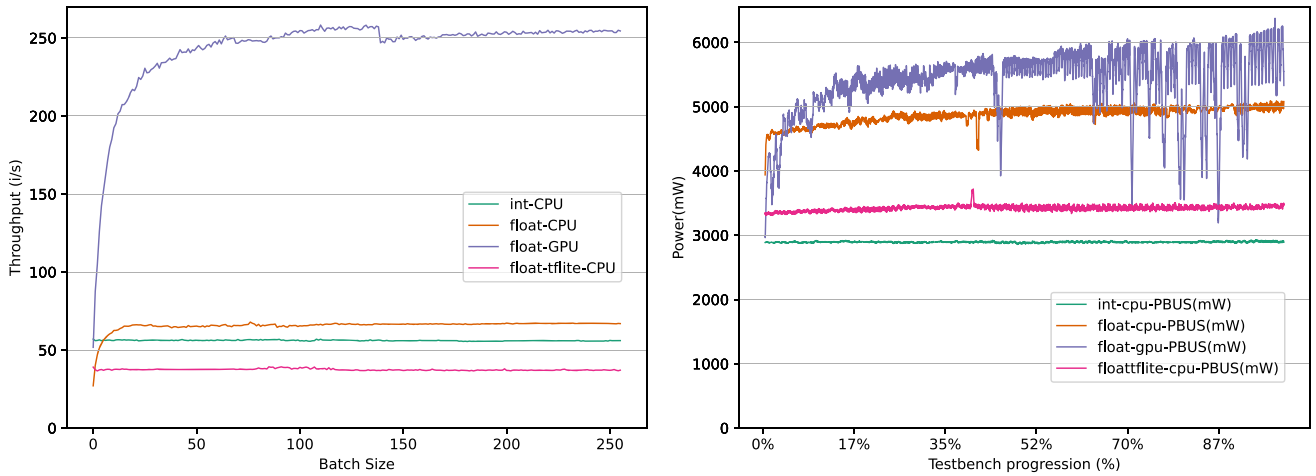

Fig. 13 Jetson performance and power measurements

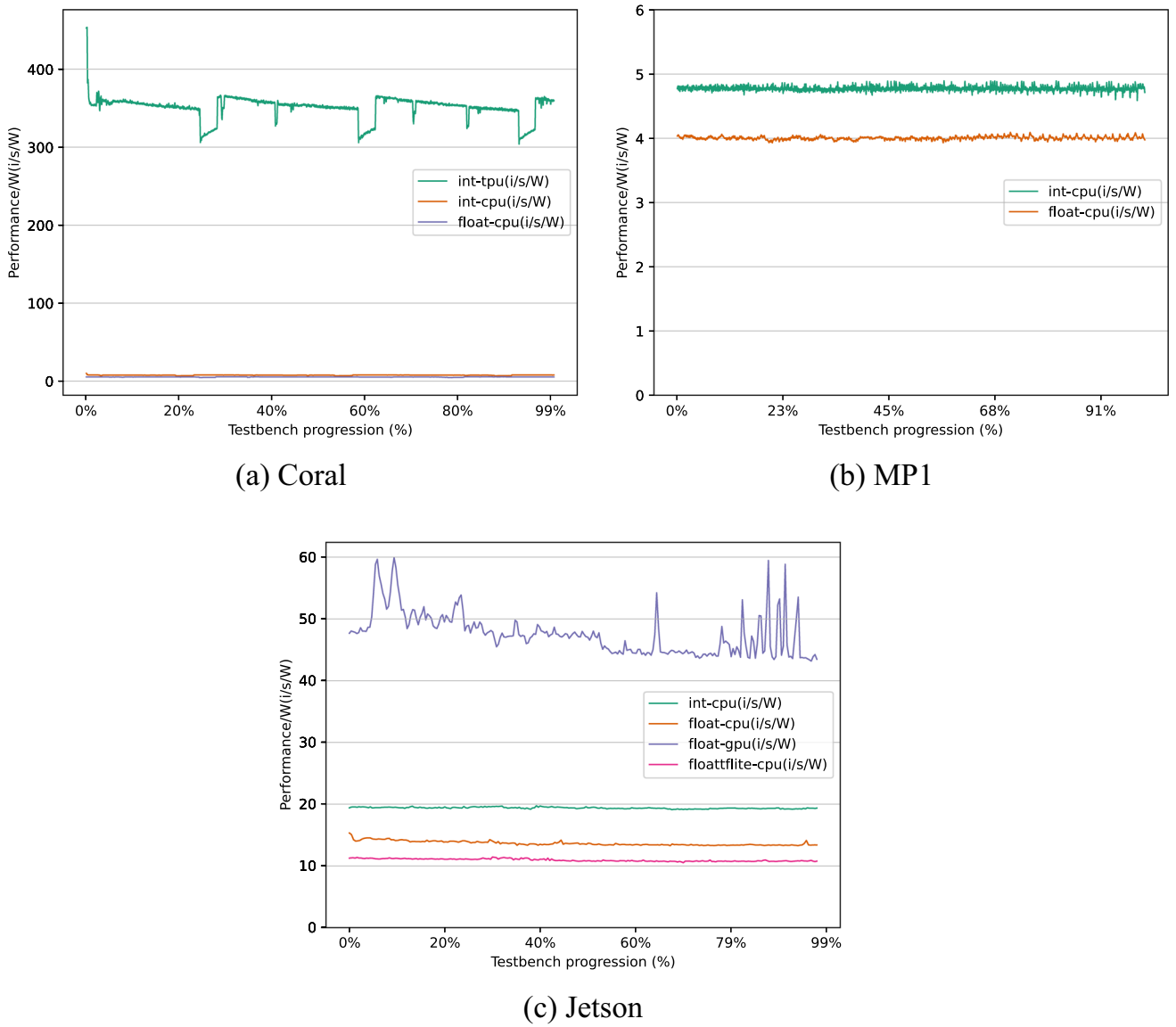

Fig. 14 Power/throughput measures

ment, which needs between 90 s and 5 min to inspect a wafer. Making a minimum of around 5 inferences per seconds for the MP1 is thus well enough and gives a margin of progression both on production line speed and data complexity. For instance, it gives room for higher resolution wafer maps or more classes.

The power measures we have made show that the Coral board has an idling power of 3.3 W, the one of the Jetson Nano is around 1.5 W, while the MP1 stays at 1 W. This of course is due to the internal hardware: the Coral embeds a higher grade processor and a hardware tensor processing unit. The Coral board has a cooling fan which might add to its overhead (the MP1 being a fanless board), although the fan never actually ran during our experiments. Although it has a GPU, the Jetson performs quite well on this metric. We measure the whole board components, including DRAM, peripheral accesses and the Linux kernel running on their cores, not specifically the coprocessor making the inference (Sze et al., 2020). We thus are in a realistic use case, giving figures of merit as they could be measured on an actual device attached to a machine in the production line.

Figure 14 plots the actual measures we did use a power meter on the power supplies of the boards. We run inference at the maximum batch size for a given board on our data set to obtain power efficiency results, in inference per second per watt (i/s/W).

Table 8 summarizes the numbers we get for each board. Results show without surprise that the Coral with its TPU is over 6× faster that the Jetson and around 300× faster than the MP1. When it comes to inferences per second per watt, the

Table 8 Performance and power efficiency summary for all boards

Board	PU Type	Inference Performance (i/s)	Power (W)	Performance per Watt (i/s/W)
Coral	TPU	1600	5.0	320
	CPU-float	21	3.9	5.4
	CPU-int	33	4.3	7.7
STM32MP1	CPU-float	5.3	1.3	4
	CPU-int	5.6	1.2	4.7
Jetson	GPU	255	5.8	44
	CPU-float	66	4.8	13.8
	CPU-tflite	34	3.3	10.5
	CPU-int	56	2.9	19

Coral performs better with 320, against 44 for the Jetson and 4 for the MP1. This gives a much better power-efficiency for the Coral board, which is easily explained by the dedicated ASIC for neural network acceleration. The GPU is also well suited to these kinds of workloads, while pure CPU computation is overall less efficient. We can note that there is roughly one order of magnitude in power efficiency between an ASIC and a GPU, and again one order of magnitude between a GPU and a CPU, which is, for dedicated workloads, common wisdom.

However, the real matter is energy, i.e. the power consumed over time. Indeed, the total consumed energy would be actually higher with more power-hungry and fast hardware because of the nature of how batches of wafer maps are generated by test equipments. Faster devices would actually stay inactive most of the time, wasting their idle power waiting the next data batch. Thus, deliberately getting less throughput still delivers the required throughput for the industrial semiconductor use case we consider, while being among the lowest power consumption solution we can get for this type of classification problem. A good future improvement of such a solution would be to better tune hardware performances to either downclock, standby or even shutdown the inference platform at the right moment to save even more power. Finally, the market cost of small boards such as the one used in these experiments are well under GPU solutions, making them even more attractive considering both the cost of the initial purchase, the exploitation, and the maintenance and replacement cost.

Conclusion and future work

Wafer map classification is an important step in semiconductor process control. While the throughput of the test equipment is low compared to, e.g. video rates, it runs around the clock. Therefore, having a low footprint accurate power-efficient solution usable directly on the industrial machines is of interest. To that end, we present in this paper a purpose defined neural network architecture that features a low parameter count that we further quantize to limit the computation and memory resources necessary to perform inference. We implement this network on micro-controller boards with and without hardware inference accelerator, and show that it can perform inference in real-time, at a 4 inference per second per watt on a small microcontroller and at over 300 i/s/W using an embedded TPU accelerator.

We proposed an approach centered on neural network model optimization, demonstrating that it is a good approach toward low-power deep learning solutions. This approach can be applied to other industrial use-cases sharing the same data set features. In particular, data sets with low interferences such as our black and white wafer maps generated by a consistent test equipment are well suited. For instance, industries such as railway or photovoltaic manufacturing have test equipment generating very similar data with small changes between them. In the end, AlexNet and GoogLeNet are very effective also with much more complex data such as 24-bit real life photographs, but they are overkill solutions when applied to very specific industrial applications. As promising as this seems, one of the most important aspects is the training data set. A clean training data set is the absolute necessity before applying any sort of deep learning approach and must be the very first priority. This means that it must be large enough, well labeled and well balanced. Only that prerequisite enables efficient model optimization eventually allowing to downscale inference platforms.

With an appropriate model, further optimization can be made by focusing on the actual inference implementation. First, kernel implementation can be optimized with power usage in mind. For instance, some instructions are by nature more power consuming than others, such as memory load and stores. Using works focusing on instruction-level power consumption optimization could thus be used to trade more power efficiency against performance, e.g. by redoing computation rather than storing and then loading an intermediate results. Secondly, hardware implementation solutions allow

to even further optimize inference power efficiency. In that situation model quantization can be further pushed toward ternary or binary models, as demonstrated in other application domains (De Vita et al., 2020). This allows very power efficient computations and minimizes memory needs, saving even more power while accuracy is only slightly degraded.

Author Contributions All authors contributed to the conception and design of the work presented in this paper. Original data collection, data augmentation, classification and data analysis were performed mainly by Maxime Martin and Ana Pinzari. All authors contributed to later refinements on these topics. Implementation was mainly conducted by Ana Pinzari and Thomas Baumela. The first draft of the manuscript was written by Ana Pinzari, Thomas Baumela and Frédéric Pétrot. All authors commented on previous versions of the manuscript. All authors read and approved the current manuscript.

Funding This work has been conducted under the framework of the EdgeAI "Edge AI Technologies for Optimised Performance Embedded Processing" project. This project has received funding from KDT JU under Grant Agreement No. 101097300. The KDT JU receives support from the European Union's Horizon Europe research and innovation program and Austria, Belgium, France, Greece, Italy, Latvia, Luxembourg, Netherlands, Norway. F. Pétrot would like to acknowledge the financial support of the French Agence Nationale de la Recherche (ANR) through the MIAI@Grenoble Alpes ANR-19-P3IA-0003 grant.

Data availability Given the high sensitivity of yield information for silicon foundries and the fact that the vast majority of the wafer maps are actual output of testing equipments, the data set that supports the findings of this study cannot be made openly available. Data are located in secured access data storage in STMicroelectronics facilities, Crolles, France.

Declarations

Conflict of interest Maxime Martin and Marcello Coppola are employees of STMicroelectronics. Frédéric Pétrot received research grants from EU and France. Ana Pinzari and Thomas Baulema work has been funded by EU/France research grants. Liliana Andrade declares no potential conflict of interest.

References

Alawieh, M. B., Boning, D. & Pan, D. Z. (2020). Wafer map defect patterns classification using deep selective learning. In *57th ACM/EDAC/IEEE design automation conference* (pp. 1–6). IEEE Press. https://doi.org/10.1109/DAC18072.2020.9218580

Andrade, L., Baumela, T., Pétrot, F., Briand, D., Bichler, O., & Coppola, M. (2021). Efficient deep learning approach for fault detection in the semiconductor industry. In O. Vermesan, R. John, C. De Luca, & M. Coppola (Eds.), *Artificial intelligence for digitising industry applications* (pp. 131–146). River. https://doi.org/10.1201/9781003337232

Andri, R., Cavigelli, L., Rossi, D. & Benini, L. (2016). YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights. In *IEEE computer society annual symposium on VLSI* (pp. 236–241). Pittsburgh, PA, USA. https://doi.org/10.1109/ISVLSI.2016.111

Cortes, C., Jackel, L. & Chiang, W.-P. (1995). Limits on learning machine accuracy imposed by data quality. In *First international conference on knowledge discovery and data mining*. (pp. 57–62). Montreal, Canada, AAAI Press. https://doi.org/10.5555/3001335.3001345

De Vita, A., Pau, D., Di Benedetto, L., Rubino, A., Pétrot, F. & Licciardo, G. D. (2020). Low power tiny binary neural network with improved accuracy in human recognition systems. In *23rd Euromicro conference on digital system design* (pp. 309–315). Kranj, Slovenia. https://doi.org/10.1109/DSD51259.2020.00057

Dodge, S. F. & Karam, L. (2016). Understanding how image quality affects deep neural networks. In *Eigth international conference on quality of multimedia experience* (pp. 1–6). Lisbon, Portugal. https://doi.org/10.1109/QoMEX.2016.7498955

Dundar, G., & Rose, K. (1995). The effects of quantization on multilayer neural networks. *IEEE Transactions on Neural Networks, 6*(6), 1446–1451. https://doi.org/10.1109/72.471364

Duvivier, F. (1999). Automatic detection of spatial signature on wafermaps in a high volume production. In *International symposium on defect and fault tolerance in VLSI systems (ETF'99)* (pp. 61–66). Albuquerque, NM, USAIEEE. https://doi.org/10.1109/DFTVS.1999.802870

Fahle, S., Prinz, C., & Kuhlenkötter, B. (2020). Systematic review on machine learning (ML) methods for manufacturing processes - Identifying artificial intelligence (AI) methods for field application. *Procedia CIRP, 93*, 413–418. https://doi.org/10.1016/j.procir.2020.04.109

Hansen, M. H., Nair, V. N., & Friedman, D. J. (1997). Monitoring wafer map data from integrated circuit fabrication processes for spatially clustered defects. *Technometrics, 39*(3), 241–253. https://doi.org/10.1080/00401706.1997.10485116

Hoskins, B., Haskard, M. & Curkowicz, G. (1995). A VLSI implementation of multi-layer neural network with ternary activation functions and limited integer weights. In *20th international conference on microelectronics* (Vol. 2, pp. 843–846). https://doi.org/10.1109/ICMEL.1995.500978

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A. & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE conference on computer vision and pattern recognition* (pp. 2704–2713). https://doi.org/10.1109/CVPR.2018.00286

Jouppi, N. P., Hyun Yoon, D., Ashcraft, M., Gottscho, M., Jablin, T. B., Kurian, G. & Patterson, D. (2021). Ten Lessons from Three Generations Shaped Google's TPUv4i: Industrial Product. In *ACM/IEEE 48th annual international symposium on computer architecture* (pp. 1–14). Valencia, Spain. https://doi.org/10.1109/ISCA52012.2021.00010

Kahng, H., & Kim, S. B. (2021). Self-supervised representation learning for wafer bin map defect pattern classification. *IEEE Transactions on Semiconductor Manufacturing, 34*(1), 74–86. https://doi.org/10.1109/TSM.2020.3038165

Kim, T., & Behdinan, K. (2022). Advances in machine learning and deep learning applications towards wafer map defect recognition

and classification: A review. *Journal of Intelligent Manufacturing*. https://doi.org/10.1007/s10845-022-01994-1

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In: F. Pereira, C. Burges, L. Bottou & K. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 25). https://doi.org/10.1145/3065386

Kyeong, K., & Kim, H. (2018). Classification of mixed-type defect patterns in wafer bin maps using convolutional neural networks. *IEEE Transactions on Semiconductor Manufacturing, 31*(3), 395–402. https://doi.org/10.1109/TSM.2018.2841416

López de la Rosa, F., Gómez-Sirvent, J., Morales, R., Sánchez-Reolid, R., & Fernández-Caballero, A. (2023). Defect detection and classification on semiconductor wafers using two-stage geometric transformation-based data augmentation and squeezenet lightweight convolutional neural network. *Computers and Industrial Engineering, 183*, 109549. https://doi.org/10.1016/j.cie.2023.109549

McLain, D. H. (1976). Two dimensional interpolation from random data. *The Computer Journal, 19*(2), 178–181. https://doi.org/10.1093/comjnl/19.2.178

Murphy, B. T. (1964). Cost-size optima of monolithic integrated circuits. *Proceedings of the IEEE, 52*(12), 1537–1545. https://doi.org/10.1109/PROC.1964.344

Nag, S., Makwana, D., Mittal, S., & Mohan, C. K. (2022). WaferSegClassNet - A light-weight network for classification and segmentation of semiconductor wafer defects. *Computers in Industry, 142*(C), 28–41. https://doi.org/10.1016/j.compind.2022.103720

Nakazawa, T., & Kulkarni, D. V. (2018). Wafer map defect pattern classification and image retrieval using convolutional neural network. *IEEE Transactions on Semiconductor Manufacturing, 31*(2), 309–314. https://doi.org/10.1109/TSM.2018.2795466

Park, K., & Simka, H. (2021). Advanced interconnect challenges beyond 5nm and possible solutions. In *International interconnect technology conference* (pp. 1–3). Kyoto, Japan: IEEE. https://doi.org/10.1109/IITC51362.2021.9537552

Park, S., Jang, J., & Kim, C. O. (2021). Discriminative feature learning and cluster-based defect label reconstruction for reducing uncertainty in wafer bin map labels. *Journal of Intelligent Manufacturing, 32*(1), 251–263. https://doi.org/10.1007/s10845-020-01571-4

Patel, D., Bonam, R., & Oberai, A. (2020). Deep learning-based detection, classification, and localization of defects in semiconductor processes. *Journal of Micro/Nanolithography, MEMS, and MOEMS, 19*(2), 024801. https://doi.org/10.1117/1.JMM.19.2.024801

Prost-Boucle, A., Bourge, A., & Pétrot, F. (2018). High-efficiency convolutional ternary neural networks with custom adder trees and weight compression. *ACM Transactions on Reconfigurable Technology and Systems, 11*(3), 1–24. https://doi.org/10.1145/3270764

Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Chen, X., & Wang, X. (2021). A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys, 54*(4), 1–34. https://doi.org/10.1145/3447582

Saqlain, M., Abbas, Q., & Lee, J. Y. (2020). A deep convolutional neural network for wafer defect identification on an imbalanced dataset in semiconductor manufacturing processes. *IEEE Transactions on Semiconductor Manufacturing, 33*(3), 436–444. https://doi.org/10.1109/TSM.2020.2994357

Shim, J., Kang, S., & Cho, S. (2020). Active learning of convolutional neural network for cost-effective wafer map pattern classification. *IEEE Transactions on Semiconductor Manufacturing, 33*(2), 258–266. https://doi.org/10.1109/TSM.2020.2974867

Sze, V., Chen, Y.-H., Yang, T.-J., & Emer, J. S. (2020). How to evaluate deep neural network processors: TOPS/W (alone) considered harmful. *IEEE Solid-State Circuits Magazine, 12*(3), 28–41. https://doi.org/10.1109/MSSC.2020.3002140

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1–9). https://doi.org/10.1109/CVPR.2015.7298594

Theodosiou, T., Rapti, A., Papageorgiou, K., Tziolas, T., Papageorgiou, E., Dimitriou, N., & Tzovaras, D. (2023). A review study on ML-based methods for defect-pattern recognition in wafer maps. *Procedia Computer Science, 217*, 570–583. https://doi.org/10.1016/j.procs.2022.12.253

Tsai, T.-H., & Lee, Y.-C. (2020). A light-weight neural network for wafer map classification based on data augmentation. *IEEE Transactions on Semiconductor Manufacturing, 33*(4), 663–672. https://doi.org/10.1109/TSM.2020.3013004

Tsai, T.-H., & Lee, Y.-C. (2020). Wafer map defect classification with depthwise separable convolutions. In *IEEE international conference on consumer electronics* (pp. 1–3). Las Vegas, NV, USA. https://doi.org/10.1109/ICCE46568.2020.9043041

Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P., Jahre, M. & Vissers, K. (2017). FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays* (pp. 65–74). https://doi.org/10.1145/3020078.3021744

Wang, J., Xu, C., Yang, Z., Zhang, J., & Li, X. (2020). Deformable convolutional networks for efficient mixed-type wafer defect pattern recognition. *IEEE Transactions on Semiconductor Manufacturing, 33*(4), 587–596. https://doi.org/10.1109/TSM.2020.3020985

Wu, M.-J., Jang, J.-S., & Chen, J.-L. (2015). Wafer map failure pattern recognition and similarity ranking for large-scale data sets. *IEEE Transactions on Semiconductor Manufacturing, 28*(1), 1–12. https://doi.org/10.1109/TSM.2014.2364237

Zhang, Q., Zhang, Y., Li, J., & Li, Y. (2022). WDP-BNN: Efficient wafer defect pattern classification via binarized neural network. *Integration, 85*, 76–86. https://doi.org/10.1016/j.vlsi.2022.04.003

Zhao, R., Song, W., Zhang, W., Xing, T., Lin, J.-H., Srivastava, M., Gupta, R., & Zhang, Z. (2017). Accelerating binarized convolutional neural networks with software-programmable FPGAs. *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays* (pp. 15–24). ACM. https://doi.org/10.1145/3020078.3021741