Vision-based multi-size object positioning

***Abstract*—Accurate object positioning is critical in many industrial manufacturing applications. The execution time and precision of the object positioning task have a significant impact on the overall performance and throughput, especially in cost-sensitive industries such as semiconductor manufacturing. In addition, the object positioning algorithm must adapt to changes in object size, features, and environmental conditions in real-time. While traditional sensors struggle to cope with dynamic conditions, vision-based perception is more adaptable and robust. Vision-based perception can capture and analyze visual information by using cameras and image processing algorithms, providing a robust way to locate objects in dynamic environments. However, classical perception algorithms based on vision cannot handle objects with different characteristics, and modern object detectors that rely on deep neural networks struggle to adapt to image sizes, resulting in unnecessary computations. To address these challenges, this paper proposes an approach for designing a branched multi-input deep neural network (DNN) that considers variations in input image sizes to adapt the input branches. In essence, the proposed DNN reduces the computation time for images with lower dimensions. To validate the proposed approach, an IC dataset is created that represents the variations in object sizes as seen in semiconductor manufacturing machines. Depending on the choice of input branches, the average inference time is reduced by over 30% with a slight gain in detection accuracy.**

***Index Terms*—Object positioning, Vision-based sensing, Deep neural networks, Semiconductor manufacturing**

I. Introduction

The critical task of object positioning in automated manufacturing and assembly systems is performed in two main stages. First, the system detects the position of the object. Second, it computes the required motion to align the object with the predetermined *reference* position. Typically sensors like potentiometric sensors, Hall effect sensors, and linear encoders are used for detecting the position of an object. However, these sensors are not always accurate and are susceptible to errors. This happens mainly due to the thermal [1], and vibration effects [2] generated in the system. As a result, the system is not able to detect the object precisely, and the object gets misaligned with the reference position resulting in bad product quality. To avoid the misalignment issue, the machine is halted, or the bad product is discarded. Both of these outcomes adversely affect the production throughput of these machines.

With the advancements in deep learning and computer vision, an alternative approach to tracking object positions is emerging. This involves capturing an image around the region of interest, resulting in images of different dimensions for different product sizes, e.g. in the case of the IC (Integrated Circuit) dataset shown in Fig. 1, the product dimension ranges from 2 mm to 60 mm. Based on pre-determined references on the ground, the object's position is estimated using Deep Neural Networks. This estimation can be used directly to compute the required motion or fused with the position estimation from other sensors to make the sensing reliable. The precision of the vision-based estimation of position heavily depends on the resolution of the camera and the errors introduced in processing the image to localize the object, primarily because of transformations performed for resizing the input image. This error also gets added to the estimated motion for aligning the object. Therefore, the use of state-of-the-art deep neural networks for object localization in such applications is often not feasible as they are trained on fixed input dimensions and thus, the images have to be either scaled down, which results in lost precision or padded up, resulting in unnecessary computations hence reducing the throughput.

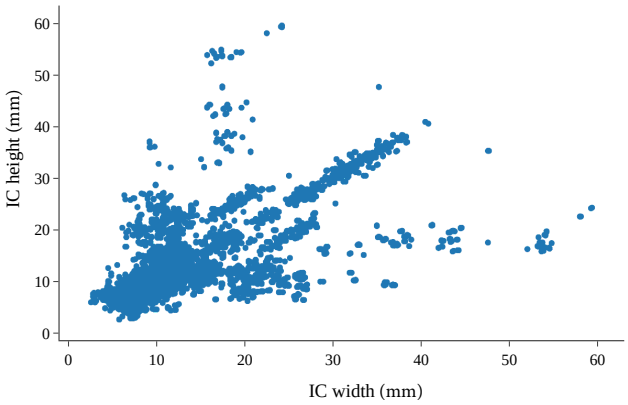

Fig. 1. Variation of product dimensions in IC dataset

Contributions: In this paper, we propose an approach to design branched multi-input deep neural networks for multi-size object localization. The number of input branches and dimensions are determined based on the expected variations in the input image dimensions. Additionally, a small common backbone network is used for the predictions. The images with higher dimensions pass through more convolutional layers and vice-versa. The effectiveness of the proposed model is demonstrated by considering the object positioning task in semiconductor manufacturing, although it is equally relevant and applicable in many other performance-demanding time-constrained domains. The model is evaluated with the help of the IC dataset, generated from the PCB-DSLR dataset [3]. For the IC dataset, three cases are discussed, Case 1 (baseline) with a single input branch, and Case 2 and Case 3 with two and three input branches, respectively. Our results show that Case 2 has a 25% gain in inference time over the baseline, whereas Case 3 gains over 30%. The detection quality of all three cases remains similar with a slight improvement in Case 2 and Case 3 over the baseline.

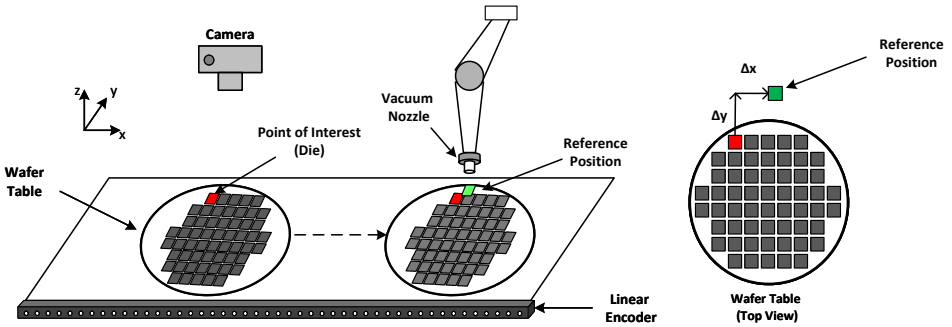

Fig. 2. (A) Simplified setup of the wafer stage in a semiconductor packaging platform with a camera mounted on top and linear encoder for reading the wafer table position. (B) The top view of the wafer table and targeted reference position where the die needs to be positioned.

II. Semiconductor Manufacturing

For the scope of this paper, object positioning in semiconductor assembly machines [4] is considered. Semiconductor manufacturing and packaging involve two primary phases: front-end (semiconductor wafer production) and back-end (assembly and packaging). The former involves several process steps like lithography and sputter deposition, which convert silicon into a processed semiconductor wafer, followed by pre-assembly steps like grinding, taping, and sawing. The wafer is ground to the required thickness and then taped to an adhesive foil in a flexible frame carrier (FFC). The wafer sawing cuts the silicon wafer into individual components called dies or chips. Subsequently, in the back end, the dies are attached to a substrate or package, followed by wire bonding and molding. Post-assembly steps involve trim and form, final test, and packing of the semiconductor product. An assembly machine is tasked with picking each semiconductor die or substrate or the packaged IC at different stages of production. A typical machine like this has to handle products of different sizes. For semiconductor dies, it ranges from $200\,\mu\mathrm{m}$ to $1\,\mathrm{mm}$, and in the case of the IC dataset, it ranges from 2 mm to 60 mm. Furthermore, these machines are known to have a throughput of 70,000 units per hour (UPH), i.e. producing a packaged product every $(60 \times 60 \times 1000)/70,000 \approx 50$ ms, which should be reduced further in order to improve the throughput.

A simplified setup of the wafer stage of a semiconductor manufacturing platform is shown in Fig. 2, where a camera is mounted on top to improve the positioning accuracy. Linear encoders are used to sense the position of the wafer table from which the position of the die is then estimated. This introduces many errors in the position estimations due to disturbances such as mechanical misalignment, material warping, friction, and vibration in the system. By utilizing the visual feedback from the camera, the controller can compensate for these errors and correctly align the dies to the pre-determined references.
In order to practically incorporate the vision feedback with the above-mentioned system, the following challenges need to be addressed:

C1 **Execution Time:** For the production throughput of 70,000 UPH the machine cycle is 50 ms and the inference time for the object localizer should be smaller than 20 ms. To further improve the production throughput the inference time should be as small as possible. Note that, as the DNN has to be used in a control loop, feedback needs to be obtained in each image frame and batch processing is not feasible.

C2 **Multi-size Products:** As the semiconductor manufacturing machine packages a wide range of products with varying sizes, the vision feedback mechanism should be able to efficiently detect ICs with sizes ranging from 2 mm to 60 mm.

C3 **Object Detection Precision:** The precision requirement for the object localizer varies at different stages of the assembly process. It also depends on the resolution of the camera, to localize the semiconductor dies higher resolution is needed as compared to the ICs. A prediction with an error less than 10 pixels is acceptable at all stages.

III. Related Work

Object detection and localization using computer vision algorithms have been studied for many years, with methods such as Hough transform [5] and template matching [6] being popular in classical computer vision literature. Hough transform is an iterative approach that extracts high-level information from an image and uses predetermined thresholds for detecting objects, making it less reliable in practical applications. Template matching uses a template image to find the object in the image, but it is not robust and not size invariant, requiring the template image to be resized to match the object size in the image. Recently, DNNs have been utilized to perform object localization. When passing the image through the DNN, a feature map is generated and bounding box regression is used to locate objects from the feature map. These methods can be categorized into single-shot and multiple-shot detectors. Single-shot detectors such as SSD [7] and YOLO [8] are computationally efficient as they perform object detection by processing the entire image once. These detectors divide the input image into a grid of cells, and each cell predicts the location, size, and confidence score of the object present in that cell. However, a major limitation of these detectors is their inability to handle images of varying sizes. Since the grid cells are fixed, the detectors struggle to detect objects that are smaller

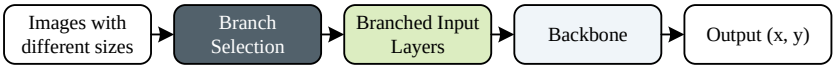

Fig. 3. Stages in proposed branched multi-input DNN

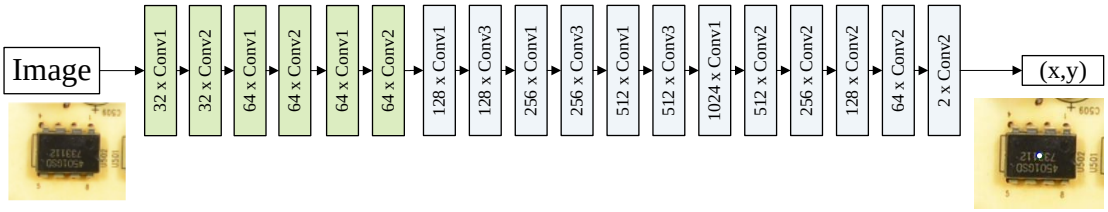

Fig. 4. Case 1: Baseline model with a single input branch with input dimension (640, 640)

or larger than the grid cells. Multiple-shot detectors, such as R-CNN [9] and its variants, use region proposal networks (RPNs) to propose regions of interest in the image. These regions are then fed to a classifier to determine whether they contain an object or not. While this approach achieves higher detection accuracy, it is computationally expensive and requires scanning the entire image multiple times with different scale windows. A model for object tracking is proposed in [10] it uses a branched input structure for dealing with images containing an object of different sizes, the number of branches and their dimensions are chosen without considering the variations of the image sizes which will still result in unnecessary computations when small images have to be padded up heavily. Furthermore, the final predictions are estimated by dividing feature maps into grids and using manually set thresholds to highlight key areas in the feature map.

IV. Branched multi-input Deep Neural Network

The proposed DNN architecture consists of a branched input structure followed by a common backbone neural network as shown in Fig. 3. The *input branches* are used to scale down the feature map while retaining some high-level information. The *backbone* is the common subsequent network connected to all the input branches and predicts the coordinates of the center of the object (x, y). The *Branch Selection* block checks the dimensions of the input image and passes it to one of the input branches. The *Branched Input Layers* provides multiple entry points for the images to the DNN, hence varying the amount of computation required to extract high-level features at the early stage of the model. The different convolution blocks used to design the input branches and the backbone are listed in Table I. The Conv2 layers, which are used in the input branches, could be seen as an alternative to the typical pooling layers, which are used to scale down the feature maps, but with a key difference i.e. Conv2 layers also retain some information from the feature map during the training process. Conv1 and Conv3 layers are identical (3,3) convolutional layers with stride (1,1) where Conv3 also has an additional (2,2) max pooling layer attached to it.

Designing the input branches: Having multiple input branches allows for faster inference for smaller images and slower inference time for large images. To optimize the model for optimal average inference time, the dimensions of the input branches are chosen in such a way that the number of images is distributed equally among all branches. This is achieved by generating equal height histograms for image dimensions in the dataset by following the steps mentioned in Algorithm 1. The $binEdges$ function takes the distributions (d) of image height (h) and width (w) and returns the *nbin+1* number of *bin_edges* for both h, and w. The dimensions for input branches are decided by taking the maximum of individual *bin_edges* values among h, and w for each branch.

TABLE I
Conv layers used in the model

Name	Filter	Stride	Max Pooling
Conv1	(3,3)	(1,1)	-
Conv2	(2,2)	(2,2)	-
Conv3	(3,3)	(1,1)	(2,2)

Algorithm 1 Compute branch dimensions

1: **Input:**$d \in [h, w]$, $nbin$
2: **Output:**$branches$
3: **function** BINEDGES(d, $nbin$)
4: $npt \leftarrow$ length(d)
5: $bin\_edge\_pos \leftarrow$ linspace$(0, npt, nbin + 1)$
6: sort(d)
7: **for** $i \in 1$ to $nbin$ **do**
8: $bin\_edges[i] \leftarrow$ interpolate$(d, bin_edge_pos[i])$
9: **end for**
10: **return** bin_edges
11: **end function**
12: $branches = max$(BINEDGES($h, nbin$),BINEDGES($w, nbin$))

In Algorithm 1, *linesapce* (line 5) creates an array with nbin+1 equally spaced samples from 0 to npt. *sort* (line 6) is used to sort the input data d. The *for* loop (line 7-9) uses *interpolate* (line 7) function to perform linear interpolation on the bin edge positions and find the corresponding values in the sorted array. This step essentially maps the positions of the bin edges to the actual data values.

Branch selection checks the dimension of the input image and passes the image to the branch with the closest bin_edges

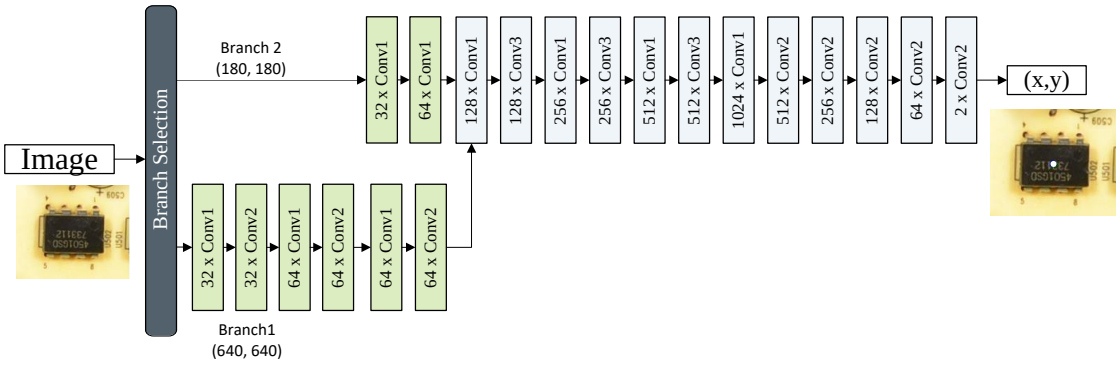

Fig. 5. Case 2: Model with 2 input branches, with input dimensions (640,640) and (180, 180)

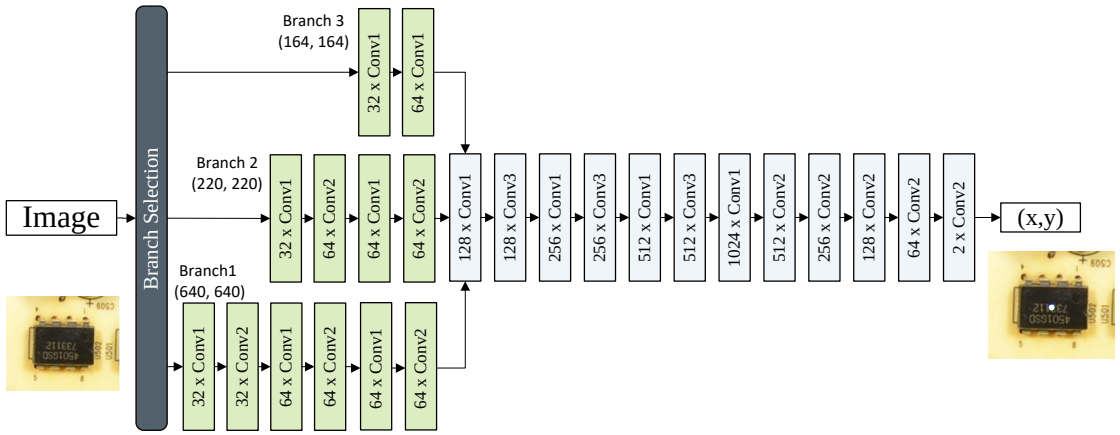

Fig. 6. Case 3: Model with 3 input branches, with input dimensions (640,640), (220,220) and (164,164)

value. bin_edges essentially gives the bounds to split the training data, but the test data may still contain bigger images thus an additional 5% of pixels are added to the bin_edges to accommodate slightly bigger images. Examples of models with two and three input branches are shown in Fig. 5 and Fig. 6 respectively. In both cases, all the input branches merge at the backbone.

V. IC Dataset

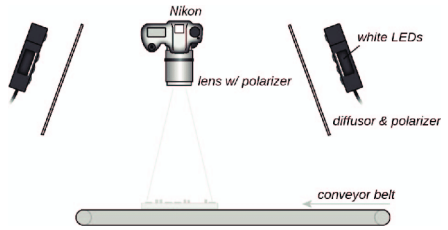

Fig. 7. Data collection setup for PCB-DSLR Dataset [3]

Though an industrial dataset for die alignment is available, it is proprietary and not available for public use. Therefore, a publicly available dataset is chosen for evaluating the proposed method. The location predicted by the object localizer has to be used for aligning the IC, the dataset should have a conversion factor to convert the distances from pixels to physical distance units. The PCB-DSLR dataset [3] satisfies the above-mentioned

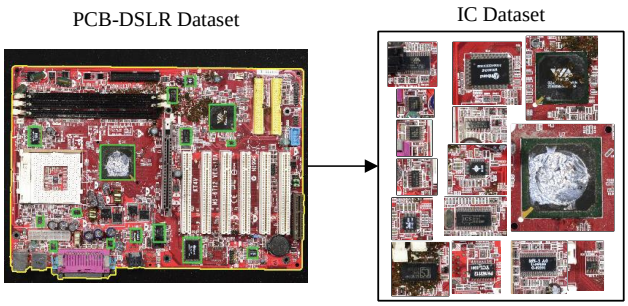

Fig. 8. An image from PCB-DSLR dataset and generated images for IC dataset

condition. The PCB-DSLR dataset consists of multiple top-down images of 165 PCBs, resulting in 748 images in total. This includes multiple images captured by physically rotating the PCBs while keeping the distance between the camera and the PCB constant. Fig. 7 shows the setups used to collect these images. Since the distance between the camera and the PCBs is constant, all the images have a fixed resolution in pixels. Each image in PCB-DSLR dataset has a resolution of 4928×3280 (about 220 pixels per inch). The conversion factor of 220 PPI allows the conversion of distances from pixels to millimeters. The labels in the dataset contain the location of all the ICs present on the PCBs in the form of oriented bounding boxes,

which are used to generate the IC dataset of 9,313 images. This is done by generating random crops around the ICs (see Fig. 8) by using the bounding box labels. The IC dataset is used to train and evaluate the performance of the models.

VI. Training and Branched Input Layers

A set of 7,313 images from the generated dataset were used to train the model. Each image in the training set is labeled with the $(\bar{x}, \bar{y})$ coordinates of the center of the IC. The Euclidean distance between the ground truth coordinates $(\bar{x}, \bar{y})$ and predicted center coordinates (x, y) is used as the loss function. This helps the model to learn to predict the location of the center as close as possible to the ground truth.

$$loss = \sqrt{(\bar{x} - x)^2 + (\bar{y} - y)^2} \tag{1}$$

For each forward pass in the training step, the input branch is selected according to the size of the image, and the corresponding loss is computed using Eq.(1). Using the loss, the weights are updated for the selected input branch along with the common layers in the backbone. Using the loss and the size of the image, the weights were updated in the respective branched input layers and the common layers. A total of 7,313 images were used for training the model, while a 1:1 split was used in the remaining 2,000 images for testing and validation. TensorFlow [11] is the choice of deep learning framework. All the models were trained using a single NVIDIA GeForce RTX 2080-Ti GPU.

The variations in the input image dimensions are shown in Fig. 9. The distribution of height and width of the images over the dataset is close, i.e. most of the images are close to being square while the individual ICs may have a rectangular shape, evident from Fig. 1. The biggest image has a dimension of (609, 636), so the input for the baseline case i.e. Case 1 with a single input branch is chosen to be (640, 640). Before inference, the image is padded up to the branch dimensions by the branch selection block. Using the procedure mentioned in Algorithm 1, equal height histograms for Case 2 and Case 3 are generated with two and three input branches, respectively. From Fig. 10, the input dimension for the two branches are chosen as (640, 640) and (164, 164), and similarly from Fig. 11, the input dimensions for Case 3 are chosen as (640, 640), (220, 220) and (164, 164). Table II summarises the input image dimensions for each branch in all three cases.

TABLE II
Input Dimensions for All Cases

Case	Input Dimensions		
	Branch 1	Branch 2	Branch 3
1	(640, 640)	-	-
2	(640, 640)	(180, 180)	-
3	(640, 640)	(220, 220)	(164, 164)

VII. Inference Results

As mentioned in Section IV there could be many possible models by varying the number of input branches. Since the training dataset contains 7,313 images, in order to ensure that each input layer is trained on sufficient data, only cases with one to three input branches are considered. **Case 1:** For the baseline single branch model, the input dimension is chosen to be (640, 640) which is bigger than the dimension of all the images in the dataset.

Case 2: For the two branch case, the input dimensions are (640, 640) and (180, 180) with 3,629 and 3,684 images in each branch respectively from the training set and 495 and 595

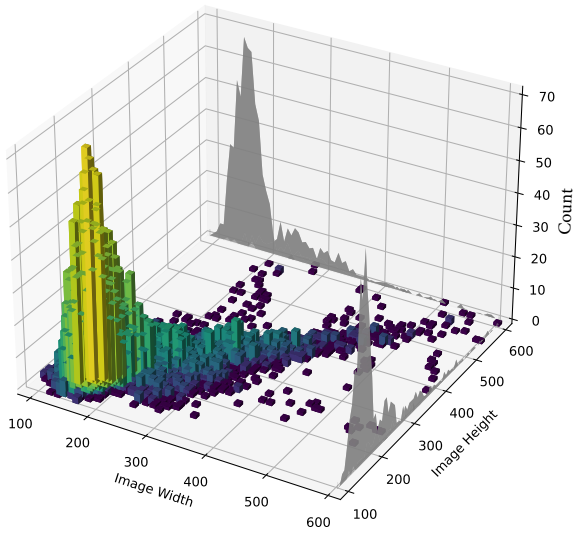

Fig. 9. Distribution of images as per their width and height in the IC Dataset

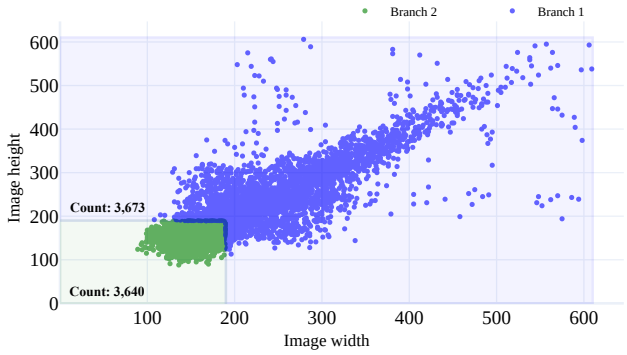

Fig. 10. Input branch dimension for Case 2

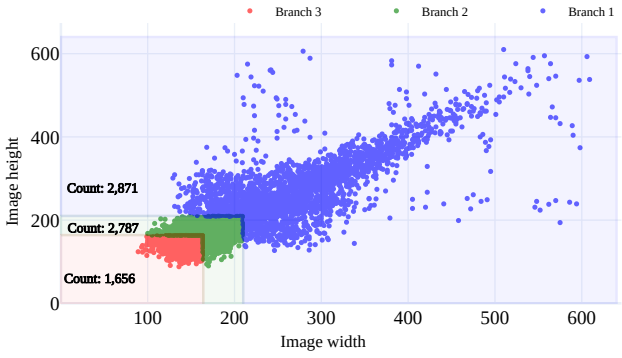

Fig. 11. Input branch dimension for Case 3

TABLE III
AVERAGE LOSS AND INFERENCE TIME FOR ALL CASES

Case	Average loss (pixels)	Average loss (mm)	Average inference time (ms)
1	7.922	0.906	16.578
2	7.439	0.851	11.516
3	6.492	0.742	9.948

images from the test set.

Case 3: For the three branch case, in addition to the dimensions used in Case 2, an extra branch is added with dimension (220, 220).

The average losses and inference times for all three cases are shown in Table III. The gain in inference time is much more significant in Case 2 and Case 3 as more input branches are added. The average loss of models in Case 2 and Case 3 is similar and much better than in Case 1, as the smaller images in Case 1 are heavily padded, leaving the model with fewer features to learn from. Table IV shows the further breakdown of inference time at different stages of all the models. These times were obtained by passing 10,000 dummy images of varied dimension through each branch. This was done to get a better indication of the average inference time over a large sample as the test set in the IC dataset is quite small and skewed towards some branches. The gain in average inference time is significant in Case 3 over the baseline case due to the introduction of additional branch with a smaller input dimension at the point where the data is densely populated.

TABLE IV
INFERENCE TIME BREAKDOWN

Case	Branch Selection (μs)	Branch	Input Branch (ms)	Backbone (ms)
1	-	1	9.178	6.353
2	0.995	2	1.615	5.103
	1.142	1	9.964	6.431
3	0.988	3	1.547	5.587
	1.126	2	2.694	5.561
	1.413	1	9.118	6.590

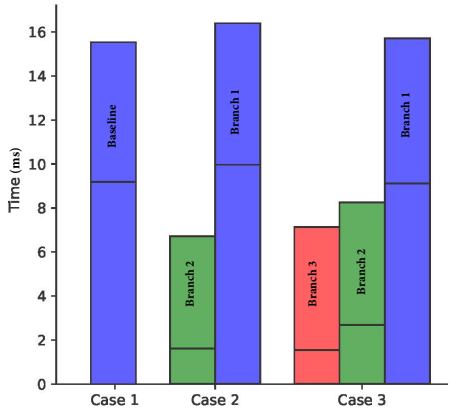

Fig. 12. Inference time breakdown as shown in Table IV

VIII. CONCLUSIONS

In this paper, we presented a flexible deep neural network design approach for object positioning tasks in industrial manufacturing applications, specifically in semiconductor manufacturing. Our proposed branched multi-input DNN considers the input image dimensions before the inference, resulting in faster inference speeds and improved accuracy. By incorporating a branched input structure, the model adapts to varying input dimensions, thus reducing unnecessary computations. As a typical semiconductor assembly machine is expected to package a large range of products with different dimensions, the effectiveness of our approach was demonstrated through its application in semiconductor manufacturing, where accurate object positioning is essential for maintaining product quality and throughput. This is done with the help of the IC dataset which contains images with varying dimensions. In summary, the proposed flexible deep neural network design offers a significant improvement over traditional sensors and existing object detectors in terms of adaptability, reliability, and performance. It addresses the challenges in(**C1:** Execution Time, **C2:** Multi-size Products and **C3:** Detection Precision) that are typically posed when incorporating the DNNs into a control loop. Future work may explore the integration of our model with other sensor fusion techniques to further enhance object positioning accuracy and reliability in industrial applications.

REFERENCES

[1] I. Alejandre and M. Artes, "Machine tool errors caused by optical linear encoders," *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 218, no. 1, pp. 113–122, 2004.

[2] J. López, M. Artés, and I. Alejandre, "Analysis of optical linear encoders' errors under vibration at different mounting conditions," *Measurement*, vol. 44, no. 8, pp. 1367–1380, 2011.

[3] C. Pramerdorfer and M. Kampel, "A dataset for computer-vision-based pcb analysis," *Proceedings of the 14th IAPR International Conference on Machine Vision Applications (MVA)*, pp. 378–381, 07 2015.

[4] G. van der Veen, J. Stokkermans, N. Mooren, and T. Oomen, "How learning control supports industry 4.0 in semiconductor manufacturing," in *Proceedings of the ASPE Spring Topical Meeting on Design and Control of Precision Mechatronic Systems*, pp. 1–5, 2020.

[5] D. H. Ballard, *Generalizing the Hough Transform to Detect Arbitrary Shapes*, p. 714–725. Morgan Kaufmann Publishers Inc., 1987.

[6] A. Sibiryakov, "Fast and high-performance template matching method," in *CVPR*, pp. 1417–1424, 2011.

[7] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," in *Computer Vision – ECCV*, pp. 21–37, Springer International Publishing, 2016.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[10] F. Lotfi, F. Faraji, and H. D. Taghirad, "Object localization through a single multiple-model convolutional neural network with a specific training approach," *CoRR*, vol. abs/2103.13339, 2021.

[11] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 265–283, 2016.