A Structured Inference Optimization Approach for Vision-Based DNN Deployment on Legacy Systems

Devi Darshini Manickam*†, Sajid Mohamed*, Vibhor Jain†, Dip Goswami†, Leonard Lensink*
*ITEC B.V., †Eindhoven University of Technology
Email: d.d.manickam@student.tue.nl, {sajid.mohamed, leonard.lensink}@itecequipment.com, {v.jain,d.goswami}@tue.nl

***Abstract*—With the growing demand for semiconductor products, the semiconductor manufacturing industries are trying to increase their production capacities. Additional requirements and constraints are also enforced on semiconductor manufacturing equipment, particularly on robustness for visual inspections and vision-based alignment. Deep neural networks (DNNs) are prominently used for vision-based tasks to improve robustness. The challenge, however, is that semiconductor manufacturing industries still use brownfield systems and equipment with legacy hardware and software. The legacy systems introduce challenging requirements and constraints on the DNN deployment and the traditional approach to inference optimization results in poor inference performance. This paper presents a structured approach to optimize the inference of DNNs for vision-based tasks for industrial brownfield architectures with existing legacy hardware, software, and the associated requirements and constraints. Four directions in the machine learning operations (MLOps) pipeline are explored in this approach - DNN architecture selection, DNN model optimization, target deployment platform, and inference engine - while adhering to the legacy systems' requirements and constraints. We present our approach using the case study from the semiconductor manufacturing industry that deploys DNNs for vision-based position detection in their legacy equipment. The results of the optimized DNN deployment are compared with a baseline implementation, and up to 44% improvement in inference timing performance is achieved without compromising on inference accuracy.**

***Index Terms*—Deep Neural Networks, Legacy Systems, Inference Optimisation, Neural Compute Stick, Google Coral Accelerator, TensorFlow, TensorFlow Lite**

I. Introduction

Vision-based algorithms play a major role in the growing trend for automation [1], [2], enabling the extraction of valuable information from visual data. Vision-based algorithms are used in various sectors like manufacturing [3], automotive [4], medical [5], and agriculture [6]. The semiconductor manufacturing industry [7] relies heavily on vision-based algorithms for monitoring, object detection, inspection, defect detection, alignment, etc. These vision-based applications traditionally use conventional image processing techniques [8] such as template matching, pixel counting, or finding simple features such as blobs, corners, and lines. For example, in [9], a template matching algorithm is used for chip localization in combination with image segmentation, blob analysis, and the dominant orientation techniques. However, the advent of deep neural networks (DNNs) has revolutionized vision processing by offering enhanced robustness and accuracy [10], and they are increasingly being adopted in the semiconductor manufacturing industry [3], [11].

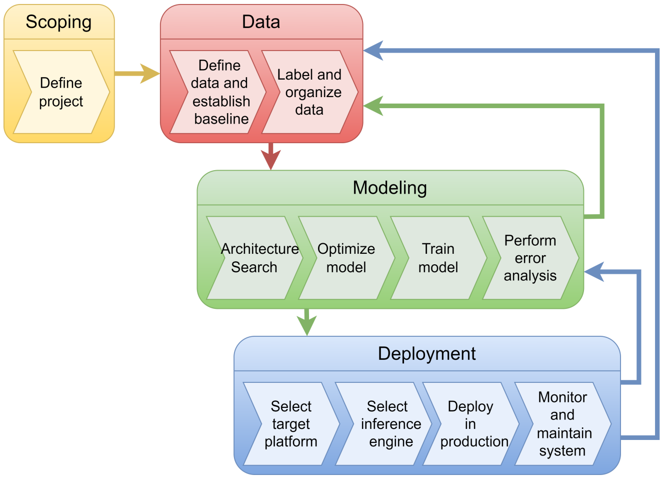

Fig. 1. Machine Learning Operations (MLOps) Pipeline

Semiconductor assembly equipment, such as die bonders and pick-and-place machines, operates at high speeds with a throughput of 72000 units per hour [12]. Vision-based algorithms deployed in such assembly equipment must process captured images quickly to meet this throughput requirement. Adopting DNNs to improve the robustness and accuracy of vision-based algorithms for high throughput assembly equipment [11], thus, requires fast inference (in the range of a few ms). There are a lot of approaches for designing vision-based DNNs and deploying them [3]–[6]. Techniques are also present for inference time optimization considering the DNN model [13] and edge platforms [14].

A typical DNN deployment uses standard machine learning operations (MLOps) pipeline [15], [16]. In the MLOps pipeline (see Figure 1), the first step is to define the scope of the project. Next, the data for the project is defined, collected, and prepared for modeling. In the modeling step, the architecture for training the model is designed, trained and error analysis is performed. Once the model is ready, it is deployed in production using a selected platform and inference engine. Finally, the system is monitored for maintenance.

The challenge, here, is that the semiconductor manufacturing industry still uses brownfield systems and assembly equipment with legacy hardware and software. Such legacy systems impose their own set of constraints and requirements and thereby limit the flexibility of DNN deployment and inference optimization. Also, a typical MLOps pipeline does not cater directly to legacy systems, and a customized inference pipeline is necessary for DNN deployment in legacy systems. The question we address in this paper is: How to optimize the inference performance of a DNN deployment while satisfying the application requirements and the constraints imposed by the legacy systems?

Contributions: This paper proposes a structured approach for optimizing the inference performance for vision-based DNNs deployed on legacy systems with their own set of constraints and requirements. The focus is on the modeling and deployment of DNNs as per the MLOps pipeline shown in Fig. 1. Four directions are explored - DNN architecture selection, DNN model optimization, target deployment platform, and inference engine. We present and validate our structured approach using a case study of a vision-based position detection algorithm for semiconductor assembly equipment. The approach starts with identifying and characterizing the constraints of the legacy system. Based on these constraints and requirements on the vision algorithm, a Kepner-Tregoe (KT) decision analysis [17] using a weighted criteria matrix is performed. The feasible choices for the case study are identified and the results are used in the KT decision analysis. The choices considered for the case study include multiple DNN model optimization techniques targeting Intel PC, Intel neural compute stick and Google coral accelerator as the hardware platforms, and using customized TensorFlow, TensorFlow Lite, OpenVINO and custom OpenCV-based inference engines. Using these techniques, a design space exploration is performed to obtain the best inference performance.

This paper is organized as follows. Section II describes the legacy system under study, and details the constraints and requirements imposed by the legacy system. Section III presents the structured approach for optimizing the DNN inference for a legacy system, based on its constraints and requirements. Section IV discusses how DNN architecture, DNN model optimization, target platform and inference engine affect the inference speed of DNN on a given platform. Section V presents the design space exploration of the chosen optimization techniques and its experimentation results. In Section VI, the results of the design space exploration are discussed and the best optimization technique is concluded. Finally, Section VII provides a conclusion for this paper.

II. Motivation and Problem Description

A. *Motivating case study: Semiconductor assembly equipment*

The case study considered is vision-based position detection of LED in ITEC's ADAT3XF PiXelect mini-LED die bonder (shown in Fig. 2). PiXelect enables manufacturing the next-generation LED direct view displays. PiXelect is ITEC's high-speed high accuracy mini-LED bonding solution that can handle the smallest LED sizes on the market. The objective of the die bonder is to bond the LED to a substrate at high speed (bonding 72000 units per hour) and high accuracy (standard deviation of 3 µm). The accuracy refers to the accuracy of the placement of LED on the desired position in the substrate.

Vision-based position detection algorithm is used to detect both LED and substrate positions. For the scope of this paper, only the LED position detection is considered. The traditional LED position detection detects - (X, Y) position and rotational angle R, as shown in Figure 3. X and Y values are used to detect the LED position and the rotation angle R is used to predict the rotation of the LED, all with respect to a fixed frame reference. Traditional vision processing algorithms [18] have poor position detection accuracy since the LEDs typically have jagged or coarse edges and there may be artifacts present in the captured LED images (e.g. defects). A DNN-based position detection algorithm is considered for deployment to improve the LED placement accuracy.

Fig. 2. ITEC's ADAT3-XF PiXelect bonder for high-speed high accuracy mini-LED bonding.

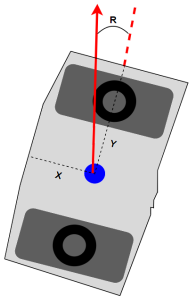

Fig. 3. A representative LED showing predicted X, Y and R Values

The challenge, however, is that the PiXelect uses legacy camera systems for capturing the images and a legacy computing platform for vision processing, which imposes requirements and constraints on the DNN deployment. Multiple applications are running on the shared legacy computing platform. Increasing the load of the vision processing task on the shared legacy platform will adversely affect the throughput of the PiXelect equipment.

B. *Constraints for DNN Deployment*

The following are some of the constraints imposed by the brownfield legacy system of PiXelect and the requirements on the vision-based position detection algorithm design and deployment. These constraints must be taken into account when choosing a technique for modeling and deploying the DNN model for the current case study. Similar to these constraints, each brownfield system will have its own constraints which must be considered while designing and deploying the DNNs.

1) **Precision of each prediction** - The Mean absolute error (MAE) of X and Y values must not exceed 1 unit for each prediction. Similarly, the MAE of R value must not exceed 0.5 units for each prediction.
2) **Operating system (OS)** - The solution must support the OS deployed in the legacy system.
3) **Processor support** - The solution must support the legacy system's processor architecture.
4) **Physical space constraints** - The interface and the size of any hardware required for the solution must be compatible with the legacy equipment.
5) **Availability in the market** - The solution must be easily available in the market without any supply chain risks.
6) **Power Requirement** - Any hardware required for the solution should draw minimal power since high power-consuming devices might affect the stability of the legacy system.
7) **Latency** - Latency requirements of the legacy platform should be met.
8) **Bandwidth** - The hardware solution with the highest number of operations per second is preferred as this will improve the inference time.
9) **Cost** - Cost of the solution must be reasonable as this will affect the PiXelect cost price.
10) **Additional software** - Software that needs to be installed in the legacy system should not affect other processes or applications already running on the legacy platform.
11) **Ease of use** - The solution which can be easily experimented is preferred as this reduces the development time.
12) **Availability of development support** - The solution for which online support is easily available is preferred as this reduces the development time.

C. Problem Statement

The main aim of this paper is to achieve the maximum inference speed for the given brownfield architecture being employed in the machines using a structured approach. In doing so, the constraints listed for the current architecture must be taken into consideration. This structured approach can be applied to any legacy system with adaptations and choices based on the constraints and requirements imposed by the brownfield architecture and legacy hardware and software.

The decision for which solution should be integrated with the legacy system is made using Kepner-Tregoe decision analysis [17] through a weighted criteria matrix using the constraints and requirements mentioned in II-B. The techniques with the best scores are filtered and experimented on in the chosen case study. Design space explorations are performed to achieve the best inference time.

III. Structured Approach for DNN Optimization

This section details the structured approach proposed for optimizing the inference performance of the vision-based DNN in legacy systems. The overall flow of the structured approach focusing on the modeling and deployment of the MLOps pipeline is shown in Figure 4.

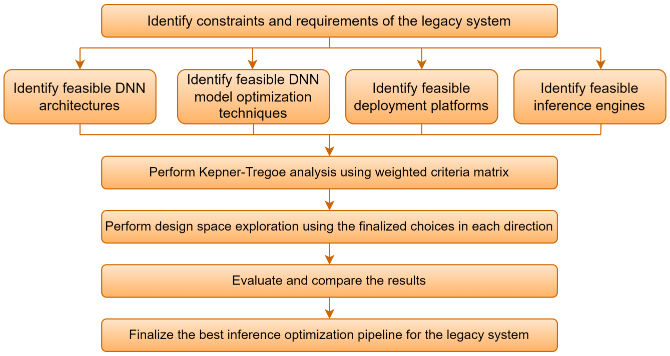
Identify constraints and requirements of the legacy system
Identify feasible DNN architectures
Identify feasible DNN model optimization techniques
Identify feasible deployment platforms
Identify feasible inference engines
Perform Kepner-Tregoe analysis using weighted criteria matrix
Perform design space exploration using the finalized choices in each direction
Evaluate and compare the results
Finalize the best inference optimization pipeline for the legacy system

Fig. 4. Structured approach for DNN inference optimization of legacy systems

The first step in this approach is to identify the requirements and constraints imposed by the chosen legacy system's hardware and software. After defining the requirements, identify the available and feasible DNN architectures, DNN model optimization techniques, targeted deployment platforms, and inference engines. These different techniques are then filtered using KT decision analysis based on the defined requirements and constraints. The KT decision analysis is performed using a weighted criteria matrix where different techniques are weighted with the requirements and constraints as parameters. The optimization techniques with the best scores are chosen and experimented.

Design space exploration is performed among the filtered techniques and the results are compared with the baseline implementation. Finally, the optimization pipeline giving the best inference results is chosen for implementation in the legacy system.

IV. DNN Inference Optimization

The four design aspects - DNN architecture, DNN model optimization, target deployment platform, and inference engine - which will influence the inference of DNN on a legacy system are detailed in this section. Figure 5 shows different alternatives under these aspects that can be opted for inference optimization. The effect of choices made on DNN inference is explained in the following subsections.

A. DNN Architecture Selection

The DNN architecture is a key design aspect that directly influences inference time. The number of parameters and the number of operations (multiplication and addition) affect the inferencing of the DNN. Hence, choosing the correct architecture for any given application is essential.

Table I shows the relation between the number of parameters, the number of math operations (multiplications and additions), and the inference time for different commonly used DNN architectures. The results are taken for ImageNet Classification on the CPU core of a Pixel 1 Phone. From this table, it is clear that the number of parameters and mathematical operations significantly varies over different DNN

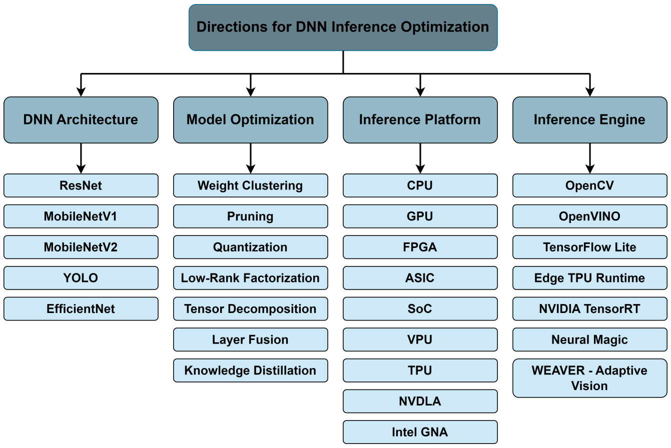

Fig. 5. Design aspects for DNN inference optimization

TABLE I
INFERENCE TIME OF COMMON DNN ARCHITECTURES [19]

DNN Architecture	No. of Parameters	No. of Mul-Adds	Inference Time
MobileNetV1 [20]	4.2M	575M	113ms
MobileNetV2 [21]	3.4M	300M	75ms
MobileNetV2 (1.4x)	6.9M	585M	143ms
NASNet-A [22]	5.3M	564M	183ms
MnasNet-A1 [19]	3.9M	312M	78ms

architectures and this further leads to substantially different inference times.

B. Model Optimizations

The next level of optimization is to optimize the model of DNN. There are many techniques, for example, pruning, clustering, and quantization, which will optimize the architecture of the model to speed up the inference time of the network. Further, each technique can be employed using different algorithms according to the use case. Some of the model optimization techniques are shown in Figure 6.

These techniques have proved to reduce the inference time in various literature. For example, in [23], various DNN architectures have been pruned and tested in different hardware platforms, such as CPU, GPU, and Jetson TK1. Each platform has shown significant speed-ups depending on the architecture used.

Similarly, using quantization, the network size was reduced by 15-20 times with 4x-6x speed up in [24] for deployment in mobile devices.

C. Target Inference Platforms

The next direction of optimization is in the hardware platform in which the DNN model will be deployed for inference. There are various hardware platforms for deploying DNN such as CPU, GPU, VPU, ASIC, FPGA, etc., each having its advantages and disadvantages, and compromising on speed and energy. The hardware platforms available in the market have features such as increased bandwidth, multiple cores, multiple multiplier units for parallelization, etc. Using these features, the inference speed of the DNN can be improved.

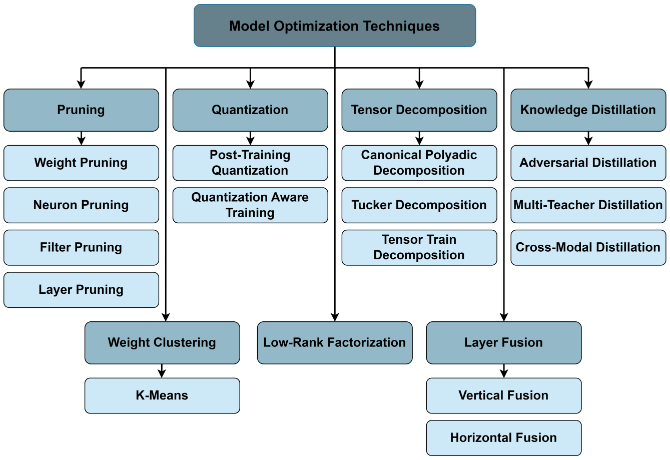

Fig. 6. Various model optimization techniques

The architecture of the inference platforms has an impact on the inference speed of the DNN. The multi-core architecture with enhanced cache in a CPU, parallelization in a GPU, ASIC architecture of a TPU with directly connected arithmetic logic units, and parallelizable vector processors in a VPU contribute to the inferencing of a neural network.

In [25], a performance comparison of Neural compute stick 2 (NCS2) and Coral USB accelerator is done for network architectures MobileNetV1 and InceptionV1. It is shown that Coral USB accelerator has better performance in terms of inference time per image compared to NCS2. However, there is a trade-off for accuracy. For MobileNetV1, NCS2 provided an accuracy of 73.7% while the Coral USB accelerator provided an accuracy of 70.6%. Similarly, for InceptionV1, NCS2 gave an accuracy of 69% while the Coral USB accelerator provided an accuracy of 65.9%. This decrease in accuracy is due to the quantization of INT8/UINT8 in the Coral USB accelerator, which is also the reason for the increased performance.

Therefore, while choosing an inference platform, the trade-off between speed and accuracy must be analyzed, and depending on the application requirements, an appropriate platform can be used for inference. In this paper, platforms such as CPU, VPU and TPU are experimented. However, due to constraints imposed by the legacy system on the inference platform II-B, GPUs and FPGA platforms are not evaluated.

D. Inference Engines

Inference in DNN is to use the trained model to make predictions on new data. The method of compiling the DNN model targeting the hardware platform considering the scheduling [26], parallelism and pipelining [27] capabilities, and deployment plays a major role in the resulting inference time of the network.

There are different inference engines available in the market, each of which uses a different technology to deploy the DNN model as shown in Figure 7. For example, Neural magic extensively makes use of caching in the CPU to improve the speed of inference. These inference engines provide different

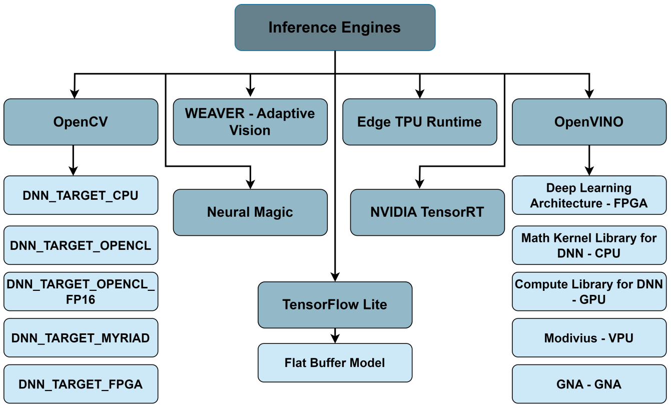

Fig. 7. Inference engines for DNN optimization

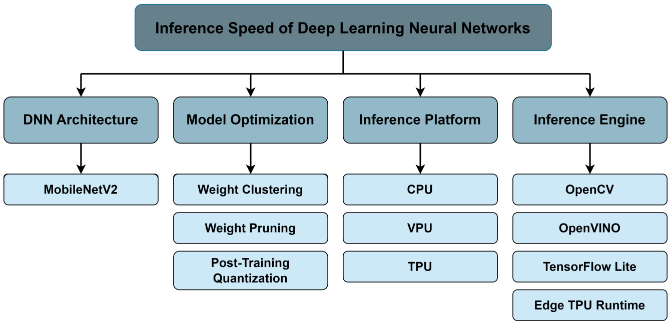

Fig. 8. Chosen optimization techniques for the use case

techniques for deployment, some specific to certain hardware platforms and optimization techniques. For example, NCS1 requires either OpenCV or OpenVINO inference engine, Google edge TPU accelerator requires Edge TPU runtime and TFLite requires TensorFlow Lite inference engine for inferencing. Clearly, the choice of inference engine majorly depends on the inference platform and the model optimization technique chosen. Choosing the correct inference engine optimizes the implementation of DNN and improves the inference speed.

V. Design Space Exploration and Optimization

Design space exploration on chosen techniques and their results are discussed in this section. After KT decision analysis on techniques listed in Figure 5, the filtered techniques under each direction are shown in Figure 8.

A. Baseline Experimentation Results

The baseline implementation is done on an Intel CPU. During the comparison of the optimization results with the baseline results, two important criteria are taken into consideration.

- Mean absolute error (MAE) between the predicted and actual label values. MAE is calculated for a dataset containing approximately 3700 images. It is calculated using the following formula,

$$MAE(L,P)_O = \frac{1}{n}\sum_{i=0}^{n-1} \mid L_i - P_i \mid \qquad (1)$$

where,

TABLE II
TensorFlow MobileNetV2 baseline model

Model	Inference Engine	MAE - X and Y	MAE - R	Inference Time (ms)
MobileNetV2	OpenCV	0.04485	0.0403	9.531

  - L is the actual label value of the image
  - P is the predicted value from the baseline TensorFlow model
  - o represents the output value - X, Y or R
  - n is the number of samples (≈3700 images)
  - i is the iteration counter

  For the chosen use case, the MAE of X and Y values must not exceed 1 unit for each prediction. Similarly, the MAE of R value must not exceed 0.5 units for each prediction.
- Average inference time for 100 iterations. The inference time here is the total time taken for copying the image to device memory, performing inference, and returning the output values. Since transferring the input and output to and from the inference platform is significant and different for each platform, this is also considered for comparing the optimized inference performance.

The MAE for the baseline model is tabulated in Table II. The average inference time for the baseline model is 9.531 ms. The inference timing of the baseline implementation is taken as the reference inference time. Any further inference timings after optimizations are compared and improved with respect to reference inference time.

The results of the optimization techniques are presented with respect to the inference platforms - CPU, VPU, and TPU.

B. Inference Optimizations for CPU Platform

First, we choose 8^{th} Generation Intel i7 CPU for experimenting with the inference of the neural network. The inference flow on the CPU is shown in Figures 9 and 10.

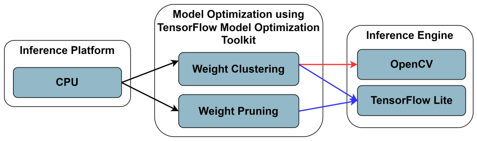

Fig. 9. Inference flow in the CPU using TensorFlow model optimization toolkit

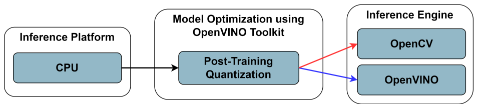
Inference Platform
CPU
Model Optimization using OpenVINO Toolkit
Post-Training Quantization
Inference Engine
OpenCV
OpenVINO

Fig. 10. Inference flow in the CPU using OpenVINO toolkit

*1) **TensorFlow Lite Base Model:*** The first optimization performed on the CPU is converting the TensorFlow model into the TFLite model. This optimization is done using the TensorFlow model optimization toolkit. The MAE and inference time of the TensorFlow Lite base model, when inferred using the TensorFlow Lite inference engine, is shown in Table III. The inference time is improved because while

TABLE III
TENSORFLOW LITE BASE MODEL ON CPU

Model Optimization	Inference Engine	MAE - X and Y	MAE - R	Inference Time (ms)
TFLite Base	TensorFlow Lite	0.04467	0.0393	5.265

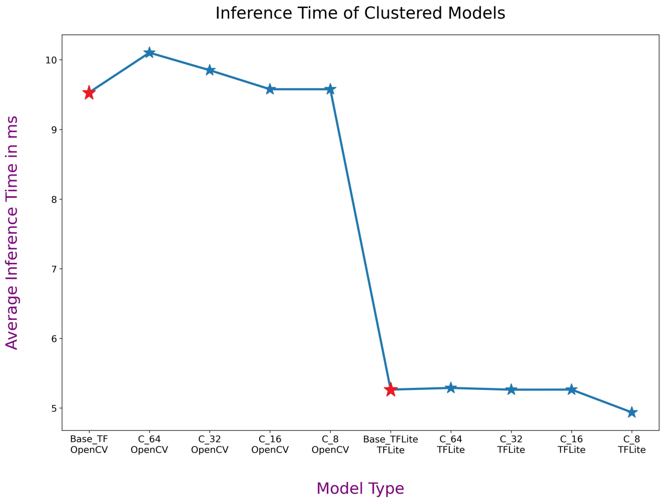

Fig. 11. Inference timing - weight clustering on CPU using OpenCV and TensorFlow Lite inference engine [where 'C' stands for cluster size]

converting a TensorFlow model into a TensorFlow Lite model, graph optimizations such as constant folding and operation fusing are done. In addition to the conversion optimizations, the TFLite inference engine also has a static memory plan and static execution plan which results in faster inference.

2) ***Weight Clustering**:* In weight clustering, instead of storing unique values in the weight matrix, similar values are grouped and replaced with the same centroid value using a clustering algorithm. Then, the values in the actual weight matrix are replaced with the centroid index which reduces the memory footprint of the model.

Weight clustering is performed using the TensorFlow model optimization toolkit. Weight clustered model can be inferred using both OpenCV and TensorFlow Lite inference engines. Hence, the results of both the models for different cluster sizes are shown in Figure 11. Cluster size defines the number of weights grouped together for finding the centroid value. In Figure 11, the red star indicates the baseline inference time of TensorFlow and TensorFlow Lite models, while the blue star indicates the inference time of clustered models. In Figure 12, the red star indicates the baseline MAE values. It can be seen from Figure 12 that the MAE values are within the acceptable error range till cluster size 16. Reducing the cluster size below 16 gives higher error values.

3) ***Weight Pruning**:* After the training of DNNs, there will be a few nodes whose weights do not make any significant change to the final output. These nodes can be removed and the method of cutting down low-impact neurons is called pruning. Weight Pruning is performed using the TensorFlow model optimization toolkit. It is performed by defining the sparsity value which defines the ratio of weights removed when compared to the original matrix. The MAE and inference time of the weight pruning model, when inferred using the TensorFlow Lite inference engine, is shown in Table IV.

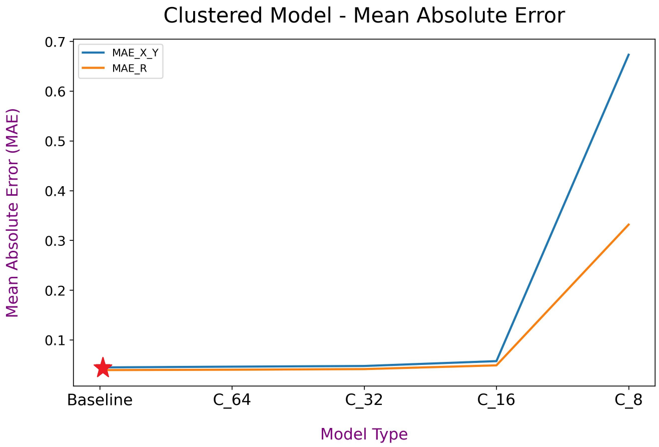

Fig. 12. MAE - weight clustering on CPU using OpenCV and TensorFlow Lite inference engine [where 'C' stands for cluster size]

TABLE IV
WEIGHT PRUNING ON CPU

Model Optimization	Inference Engine	MAE - X and Y	MAE - R	Inference Time (ms)
0.50 Sparsity	TensorFlow Lite	0.04407	0.03399	5.263
0.80 Sparsity	TensorFlow Lite	0.05067	0.03685	5.265

4) ***Post-Training Quantization**:* Quantization is the process of reducing the precision of weights and biases in the network. The floating points used in the networks can be reduced, for example from 32-bit to 16-bit or 8-bit which will reduce the network size. Quantization is performed using the Intel OpenVINO toolkit. The results of the quantized model, when inferred using OpenCV and Intel OpenVINO inference engine, are shown in Figure 13.

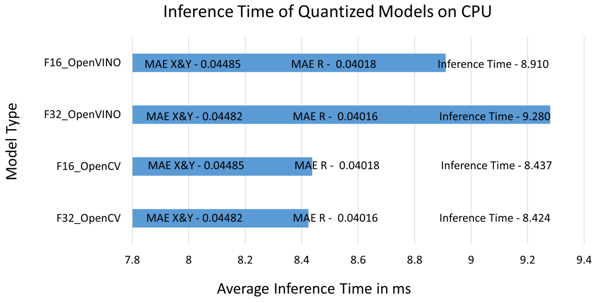

Fig. 13. Inference timing - quantization on CPU using OpenCV and OpenVINO inference engine [where 'F' stands for float value]

C. Inference Optimizations for VPU Platform

The next platform chosen for inferencing is the VPU (Vision Processing Unit). Here, the VPU used is the Intel neural compute stick 1(NCS1). The neural network model optimized using the Intel OpenVINO toolkit will be inferred using the OpenVINO inference engine on the NCS1. Since the

TABLE V
QUANTIZATION ON VPU AND TPU

Model Optimization	Inference Platform	Inference Engine	MAE - X and Y	MAE - R	Inference Time (ms)
OpenVINO float16	VPU	Intel OpenVINO	0.04485	0.04018	28.965
TFLite uint8	TPU	TensorFlow Lite and Edge TPU Runtime	0.06024	0.04567	7.115

OpenVINO toolkit does not support weight clustering and weight pruning, only post-training quantization is performed on NCS1.

Post-Training Quantization: Since only float16 models are supported on VPU, only the float16 model is tested on the NCS1. The MAE and inference timing of a float16 model inferred on the Neural compute stick 1 using the Intel OpenVINO toolkit is shown in Table V.

D. Inference Optimizations for TPU Platform

The final platform chosen for inferencing is the TPU (Tensor Processing Unit). In this paper, the TPU used is the Google coral USB accelerator, which is a Google edge TPU accelerator. The neural network model optimized using the TensorFlow model optimization toolkit will be inferred using the TensorFlow Lite inference engine in combination with Edge TPU runtime on the Google coral USB accelerator.

Post-Training Quantization: Only a full integer model in TFLite format is supported for inferencing on a Google edge TPU accelerator. This quantization is done using the post-training quantization available with the TensorFlow model optimization toolkit. Inference on the Google coral USB accelerator is done using the TensorFlow Lite inference engine and Edge TPU runtime. While the TensorFlow Lite inference engine is used for reading the network and performing inference, Edge TPU runtime is used for interfacing with the Google edge TPU accelerator.

The MAE and inference timing of a full integer uint8 model inferred on the Google coral USB accelerator using the TensorFlow Lite inference engine and Edge TPU runtime are detailed in Table V.

VI. DISCUSSION

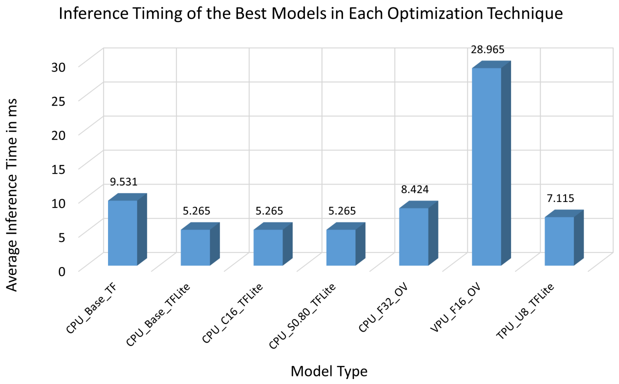

Fig. 14. Inference timing of the best models in each optimization technique

Models with the lowest inference timing from each optimization technique are chosen for analysis to compare the inference timing of different optimization techniques. The chosen model is shown in Figure 14.

From Figure 14, it can be seen that the inference of the TFLite base model, TFLite clustered model, and TFLite pruned model on the CPU with the TensorFlow Lite inference engine has given the lowest inference time of 5.265 ms. The main reason for TFLite models to have faster inference is that the TFLite models are stored as FlatBuffers. It can be accessed directly without parsing, reducing the time taken for inference.

The difference in inference timing for the first inferred image and the average inference time between the baseline model and the optimized model can be seen in Figure 15.

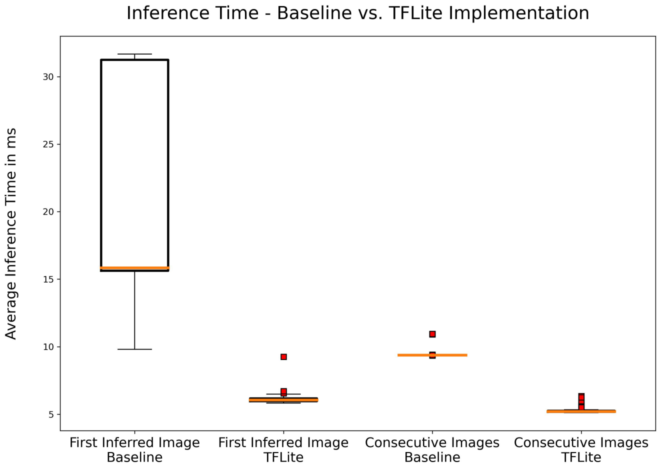

Fig. 15. Baseline model vs. optimized model inference time

However, the model optimization techniques such as weight clustering and weight pruning do not improve the inferencing of the model. Though these models modify the architecture and reduce the size and memory footprint of the network, it does not highly impact the inferencing speed of the neural network model. In weight clustering, unique weight values are replaced by centroid values. Similarly, in weight pruning, low-magnitude weight values are replaced with zero. In these techniques, the number of operations to be carried out remains the same. Hence, the inferencing time is not improved with these optimization techniques. Different kinds of clustering and pruning algorithms can be experimented with to see if they improve the inference speed of the network.

With regard to quantization, the difference in inference time between the float32 and float16 models is not significant. In quantization, similar to the above-mentioned techniques, the network size is reduced by changing the precision of the parameters of the network and not the number of operations. However, quantization of the model further to uint8 or int8 might improve the inference speed as this reduces the computation time when compared to floating point numbers.

The next is inferencing on the VPU - Neural compute stick platform. It can be seen that the inference time on the NCS1 is 28.965 ms which is higher than the base model time of 9.531 ms. This increased time is mainly due to the re-connection of the NCS1 for each inference. The time to connect the NCS1 to the PC, the time to transfer the image to the NCS1, the time to perform inference, and the time to transfer the output back to the PC add up to give this increased time. Hence, NCS1 is not a suitable inference platform for the given use case. An alternative is to experiment with Neural compute stick 2 which Intel claims to be 8 times faster than the NCS1.

Finally, we discuss the Google coral USB accelerator platform. The inference of the TFLite model on the Coral USB accelerator is 7.115 ms which is higher than the inference time of the TFLite model on the CPU. Coral USB accelerator has a USB-C interface connected to the PC's USB-B port through a connector cable. The transfer of the input image and the output through this wire length increases the inference time of the network. An alternative is to experiment with Coral accelerators with the PCIe interface. This will eliminate the delay due to wire transfer as it is directly interfaced with the PC.

In general, it can be seen from this study that the inference engine and how the model is mapped to the inference platform play a major role in determining the inference speed of the neural networks. Each inference platform requires a particular model format which depends on the inference engine being employed. Hence, choosing the correct inference engine and a suitable model format for an inference platform will help achieve good inference performance.

VII. Conclusion

This paper proposes a structured inference optimization approach for vision-based deep neural network (DNN) design and deployment on legacy systems. Four aspects of a machine learning operations (MLOps) pipeline are explicitly considered - DNN architecture selection, DNN model optimization, target deployment platform and inference engine. The structured approach for DNN design and deployment is explained using a case study from the semiconductor manufacturing industry using ITEC's PiXelect mini-LED bonder legacy system.

The structured approach starts with identifying and characterizing the constraints of the legacy system and requirements on the vision algorithm. A Kepner-Tregoe (KT) decision analysis using a weighted criteria matrix is performed for identifying the inference optimization techniques that can be integrated with the legacy system using the four aspects of the MLOps pipeline. Design space exploration is also performed for the identified inference optimization techniques. Using our approach, the shortest inference time was achieved by deploying the TFLite DNN model on the CPU using the TensorFlow Lite inference engine. The inference time achieved is $\approx 44\%$ faster than the baseline implementation. However, the inference time still needs improvement to achieve the required throughput of the equipment and will be investigated in future work. Future work also involves studying the impact of other aspects in the MLOps pipeline for inference optimization.

Acknowledgment

This work was supported by ECSEL Joint Undertaking in the H2020 project IMOCO4.E [1], grant agreement No.101007311.

References

[1] S. Mohamed *et al.*, "The IMOCO4.E reference framework for intelligent motion control systems," in *ETFA*, 2023.

[2] M. Čech, A.-J. Beltman, and K. Ozols, "Digital twins and AI in smart motion control applications," in *ETFA*, 2022.

[3] T. Schlosser, M. Friedrich *et al.*, "Improving automated visual fault inspection for semiconductor manufacturing using a hybrid multistage system of deep neural networks," *Journal of Intelligent Manufacturing*, vol. 33, no. 4, pp. 1099–1123, 2022.

[4] D. Vajak, M. Vranješ *et al.*, "Recent advances in vision-based lane detection solutions for automotive applications," in *International Symposium ELMAR*. IEEE, 2019, pp. 45–50.

[5] A. Esteva, K. Chou *et al.*, "Deep learning-enabled medical computer vision," *NPJ digital medicine*, vol. 4, no. 1, p. 5, 2021.

[6] T. R. Gadekallu, D. S. Rajput *et al.*, "A novel pca–whale optimization-based deep neural network model for classification of tomato plant diseases using gpu," *Journal of Real-Time Image Processing*, vol. 18, pp. 1383–1396, 2021.

[7] H. Geng, *Semiconductor manufacturing handbook*. McGraw-Hill Education, 2018.

[8] C. Jugade, D. Hartgers *et al.*, "An evaluation framework for vision-in-the-loop motion control systems," in *ETFA*, 2022.

[9] F. Zhong, S. He, and J. Yi, "A fast template matching method for led chip localization," in *MATEC web of conferences*, vol. 34. EDP Sciences, 2015, p. 04002.

[10] S. De, Y. Huang *et al.*, "Hardware- and situation-aware sensing for robust closed-loop control systems," in *DATE*, 2021.

[11] V. Jain, S. Mohamed *et al.*, "Vision-based multi-size object positioning," in *DSD*, 2023.

[12] G. van der Veen, J. Stokkermans *et al.*, "How learning control supports industry 4.0 in semiconductor manufacturing," in *ASPE Spring Topical Meeting on Design and Control of Precision Mechatronic Systems*, 2020.

[13] B. Taylor, V. S. Marco *et al.*, "Adaptive deep learning model selection on embedded systems," *ACM SIGPLAN Notices*, 2018.

[14] X. Xu, Y. Ding *et al.*, "Scaling for edge inference of deep neural networks," *Nature Electronics*, vol. 1, no. 4, pp. 216–222, 2018.

[15] D. Baylor, E. Breck *et al.*, "Tfx: A tensorflow-based production-scale machine learning platform," in *KDD*, 2017.

[16] M. Zaharia, A. Chen *et al.*, "Accelerating the machine learning lifecycle with mlflow," *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.

[17] J. S. Parker and J. D. Moseley, "Kepner-tregoe decision analysis as a tool to aid route selection. part 1," *Organic Process Research & Development*, vol. 12, no. 6, pp. 1041–1043, 2008.

[18] C. Jugade, D. Hartgers *et al.*, "Improved positioning precision using a multi-rate multi-sensor in industrial motion control systems," in *European Control Conference (ECC)*, 2023.

[19] M. Tan, B. Chen *et al.*, "Mnasnet: Platform-aware neural architecture search for mobile," in *CVPR*, 2019.

[20] A. G. Howard, M. Zhu *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[21] M. Sandler, A. Howard *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018.

[22] B. Zoph, V. Vasudevan *et al.*, "Learning transferable architectures for scalable image recognition," in *CVPR*, 2018.

[23] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[24] J. Wu, C. Leng *et al.*, "Quantized convolutional neural networks for mobile devices," in *CVPR*, 2016.

[25] L. A. Libutti, F. D. Igual *et al.*, "Benchmarking performance and power of USB accelerators for inference with MLPerf," in *2nd Workshop Accelerated Mach. Learn.(AccML)*, 2020, pp. 1–15.

[26] S. Mohamed, *Multiprocessor Image-Based Control: Model-Driven Optimisation*. Eindhoven University of Technology, 2022.

[27] S. Mohamed, D. Goswami *et al.*, "Optimising multiprocessor image-based control through pipelining and parallelism," *IEEE Access*, 2021.