Application Note

Vision AI-Based Defect Detection on AM62A Using TI Edge AI Studio

Qutaiba Saleh and Tarkesh Pande

ABSTRACT

AM62A system-on-chip (SoC) is used to build an end-end application for defect detection in manufacturing. AM62A is a heterogeneous processor equipped with a 2 TOPS Deep Learning Accelerator and up to four Arm® Cortex® A53 processors in addition to various other accelerators for Video and Vision processing. The various compute cores and rich peripheral set make AM62A an ideal option for applications where advanced sensor processing capability is required in real-time. This document describes the complete process of building a defect detection application starting from data collection, deep learning model selection, model training and model deployment. It shows how TI's EdgeAI Studio tools simplify this process. System level performance analysis of the application, resource utilization and power profiling using TI's tools are presented. Source code and a step-by step guide in TI's github repository are also available and links are provided for the interested developer at: https://github.com/TexasInstruments/edgeai-gst-apps-defect-detection.

Table of Contents

List of Figures

List of Tables

Trademarks

Sitara™ is a trademark of Texas Instruments.
Arm® and Cortex® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
All trademarks are the property of their respective owners.

1 Introduction

1.1 Defect Detection Demo Summary

Defect detection is a crucial part of quality assurance in the manufacturing process. This demo uses AM62A to run a vision based artificial intelligence model for defect detection for manufacturing applications. The model tests the produced units as they move on a conveyor belt, to recognize the accepted and the defected units. Figure 1-1 shows a screenshot of the application.

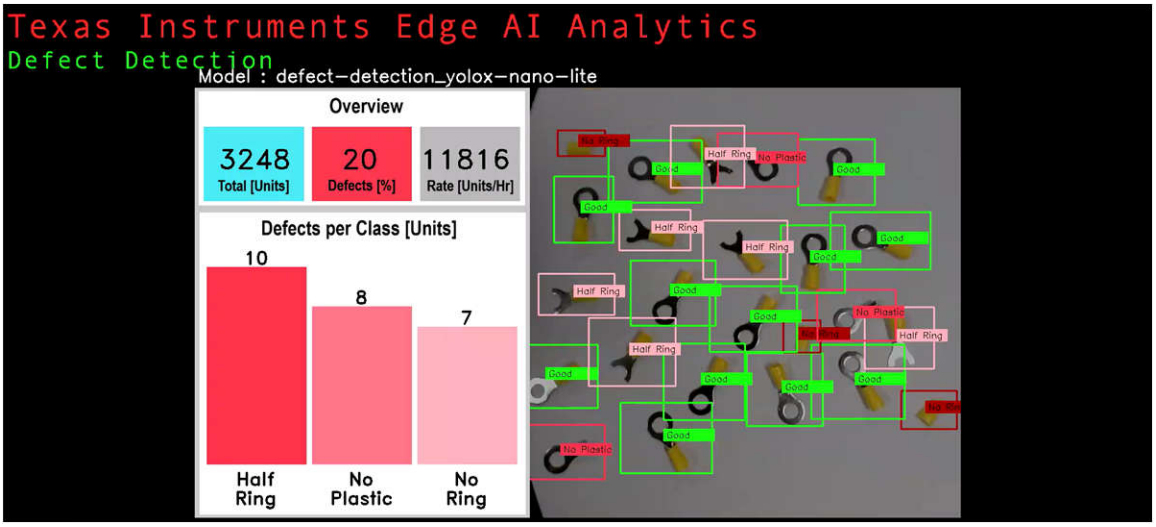

Figure 1-1. Screenshot of the Defect Detection End Application Using AM62A (Right side is a live video feed with color boxes marking the detected objects. Left side is a graphical quality control dashboard.)

An object tracker is developed for this demo to provide accurate coordinates of the units for sorting and filtering. A live video is displayed on the screen. The units are marked on the screen using green boxes for good (accepted) units while defected units are marked with boxes with different shades of red to distinguish the types of defects. The screen also includes a graphical dashboard showing live statistics about total products, defect percentage, production rate, and a histogram of the types of defect. The object tracker and the graphical dashboard are built using Python. The code base and details of how to run the demo are available on TI marketplace as a Github repo: https://github.com/TexasInstruments/edgeai-gst-apps-defect-detection.

The steps followed to develop this application includes:

- Data collection and preparation
- Model selection, training, and compilation
- Model evaluation and deployment
- Application code development including object tracker and graphical dashboard
- System performance analysis and power consumption estimation

1.2 AM62A Processor

The AM62A Edge AI Microprocessor, shown in Figure 1-2, is a heterogeneous processor designed for analytics applications. There are different hardware accelerators optimized for different tasks thus enabling an optimized power and cost footprint for different machine vision tasks. Figure 1-2 shows a simplified block diagram for AM62Ax.

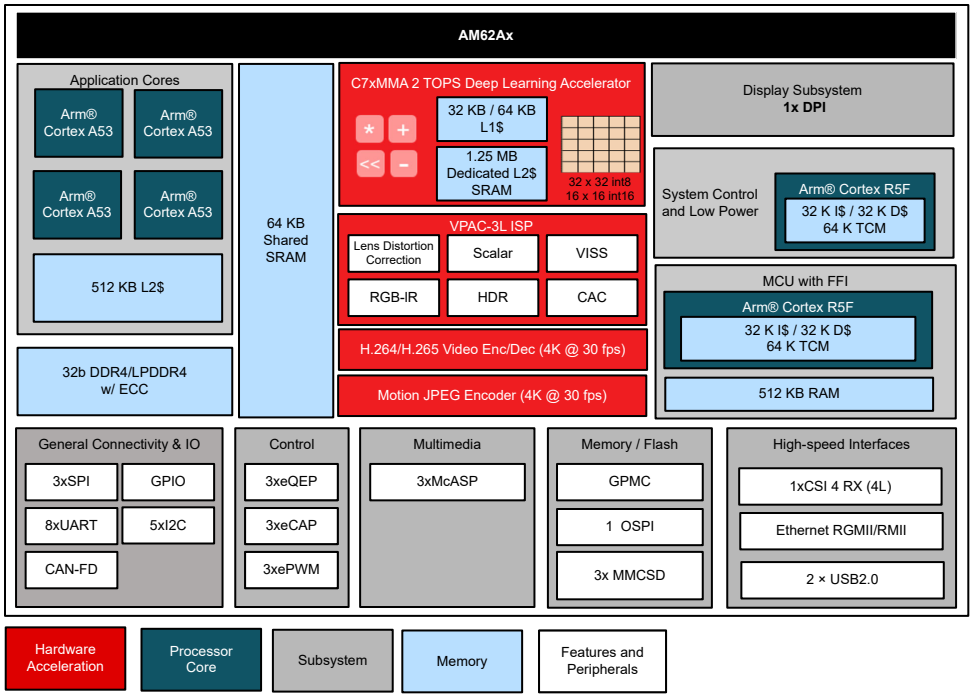

Figure 1-2. AN62Ax Simplified Block Diagram

The main compute and interface subsystems from a machine vision context in AM62A are as follows:

- Quad-A53 ARM cores: These can run up to 1.4 GHz. The *Sitara™ AM62Ax Benchmarks* presents performance details of various components of the AM62Ax SoC.
- C7 Digital Signal Processor (DSP) and Matrix Multiplication Accelerator (MMA): TI's deep learning accelerator on AM62A is capable of 2 TOPs operations when clocked at 1 GHz.
- H.264/H.265 Encoder and Decoder: This has a total encode/decode capability of 240 MP/s and can support multiple channels concurrently.
- Vision Processing Accelerator (VPAC3L): The latest generation in TI ISP technology for performing image operations some examples of which are color conversions, pyramid scaling and lens distortion correction. It has a total throughput of up to 315 MP/s with support for RGBIR cameras.

For more details, see the *AM62Ax Sitara Processors Data Sheet*.

1.3 Defect Detection Systems

A defect detection systems is used to inspect products and detect abnormalities such as irregular shape, broken part, cracks, and so forth. It feeds information to a filtering system (for example, robotic arm) to separate units that are not accepted for packaging or for the following production processes. Depending on the types of the products and the expected defects, various types of input can be used including camera, laser, ultrasound, and so forth. This demo focuses on vision-based inputs. A typical defect detection system consists of the following components:

- A camera with appropriate resolution and frame rate.
- Computational unite to perform AI model inference and other processes such as logging results, perform statistics calculation, connect to server if required to log information, networking, and so forth. This demo uses AM62A SoC.
- Conveyor system to deliver units under test.
- Screen (monitor) to show information as needed. Live feed of camera with detection results overlaid is a common example.
- A mechanical system (could be a robotic arm) to filter rejected units based on AI model decision.
- Other parts such as alarm and networking solution as needed.

Using deep learning to process the vision-based input in defect detection systems has several advantages over using conventional machine vision algorithms. The next section presents a detailed comparison between the two options.

1.4 Conventional Machine Vision vs Deep Learning

Defect detection in conventional machine vision uses rules-based algorithms. Such systems require direct engagement of experts in image processing to define a set of rules to develop application specific algorithms. These algorithms usually consist of multiple classical feature detectors followed by a series of conditional decisions. Some examples of the rules might include existence of a specific shape or dimensional relation between certain features. Embedded system engineers have to program algorithms to the desired system. This process takes months of work. On the other hand, deep learning models can be easily trained with the appropriate dataset with no need to specify features or rules. The trained models can be easily ported to the desired embedded system. TI provides a suit of tools to train, compile, and benchmark deep learning models Edge AI Studio.

Table 1-1. A Comparison Between Conventional Rules-Based Machine Vision Systems and Deep Learning Using TI Edge AI.

Conventional Rules-Based Systems	Deep Learning Using TI Edge AI
Requires image processing expertise	Models can be trained using Edge AI Studio with little to no previous deep learning experience
Requires HW specific algorithm programming expertise	Model can be directly imported to AM62A
Algorithms are application specific	TI EdgeAI-ModelZoo provides hundreds of models that can be easily retrained for different applications
Longer development time	Shorter development time
Usually requires general purpose processor	Models can be off loaded to the C7x/MMA deep learning accelerator
Requires less computation resources compared to deep learning	Requires more computation resources compared to rules-based
Requires smaller dataset compared to deep learning	Requires bigger dataset to train the model
Generally used for simpler tasks such as object tracking	Used for more complex tasks such as object detection and sematic segmentation
Less robust to environmental changes such as lighting condition and camera angle	More robust to environmental changes
Hard to update and tune after development	The model can be easily re-trained using Edge AI Studio

2 Data Set Preparation

A custom dataset is collected and used in this demo. A total of 400 unique pictures are collected. Data augmentation is used to expand the dataset to a total of 4800 pictures. This section describes the steps followed to collect, annotate, and augment the dataset.

2.1 Test Samples

Insulated Ring Crimp Terminals Connectors are used as samples for the defect detection application. Figure 2-1 shows pictures of the samples with measuring rulers as a reference. The industrial size of the terminals is M8 12-10 AWG with yellow cap. The actual dimensions are:

- Length ~= 34 mm
- Metal circle external diameter ~= 15 mm
- Plastic cap diameter ~= 8 mm

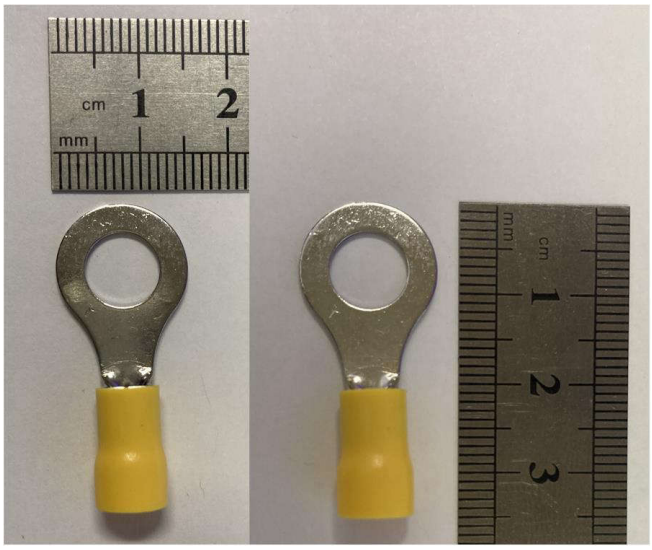

Figure 2-1. Tests Samples Used for the Defect Detection Demo, Ring Crimp Terminals Connectors

These objects have several appealing characteristics for an AI based defect detection demo to show its capability compared to conventional rules-based machine vision algorithm. The small size allows including tens of pieces in one frame to show application's capability to detect and track high number of objects. The two types of materials in the objects provide more options to generate artificial defects for demonstration purposes. The shiny metal part looks different depending on the lighting condition which show system capacities to work with challenging to detect objects.

The yolox-nano-lite is used in this demo and it can be trained for more classes.

2.2 Data Collection

Four classes are used to train the model: Good (accepted) and three classes of defects including Half Ring, No Plastic, No Ring. Figure 2-2 shows examples of the four classes. The images in the figure are cropped for clarity purposes. The four classes here are selected for demo purposes.

Figure 2-2. Examples of Pictures From the Four Classes (the pictures are cropped for clarity purposes)

A custom data collection protocol is followed to simplify the image capturing and annotation. The pictures are taken with top view angel and the camera is positioned at a height that is approximate to the height expected in the actual demo setup. The pictures are captured with a resolution of 720x720. 100 pictures were taken for each class (total 400 pictures). Only one sample is used in the 100 pictures for each class. The samples are positioned at the same orientation while the lighting condition is changed for each picture. This setup helps in the annotation of the pictures where the annotation (bounding box and class label) can be copied between pictures of the same class. In the same time, it provides a comprehensive set of pictures which cover various lighting conditions. While all objects are positioned in one place in the picture, the model can generalize to other areas of the picture. Figure 2-3 shows samples of the pictures captured for the good class.

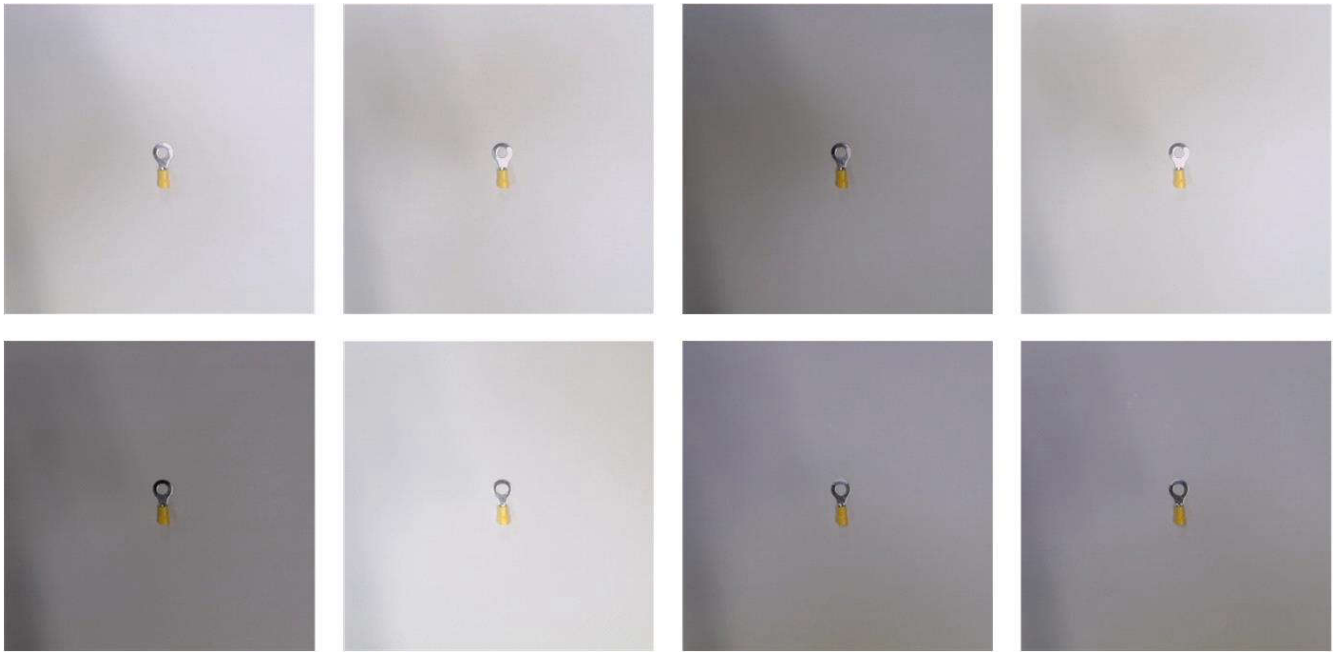

Figure 2-3. Samples of Pictures Captured for the Good Class (the pictures are captured at 720x720 resolution)

2.3 Data Annotation

In object detection, data annotation includes defining abounding box around the objects in the pictures and associate a class label for each bounding box. The dataset in this demo is collected in a way which simplifies the annotation process. The pictures are taken with only one object in each one. The objects in the pictures in each class are located at the exact position and orientation. For this reason, the bounding box and class label on one picture can be copied to the rest of the pictures in the same class. In such case, only four pictures (one of each class) are annotated and their annotation are copied to the rest of the pictures in their respective classes. The coco annotation standard is followed in this demo.

2.4 Data Augmentation

Data augmentation in machine learning includes generating altered copies of the pictures in a dataset. Data augmentation can be as simple as adding noise to the dataset in order to generate enough variation to prevent model overfitting. It is also used to expand the dataset by adding those altered copies. Two geometrical augmentation methods are applied in this demo: flip right-left and rotation. First flipped copies are created for each picture, which brings the total number of pictures to 400x2=800. Then five rotated copies of each picture are created, which brings the total number of pictures up to 800+800x5=4800 pictures. The rotation angle is randomly selected for each picture. This step substantially increases the total number of pictures in the dataset without the burden of data capturing and annotation.

3 Model Selection and Training

3.1 Model Selection

In order to execute the model on the C7x/MMA deep learning accelerator, it must be compiled/exported to a friendly format. TI's EdgeAI-ModelZoo provides hundreds of state-of-the-art models which are converted/exported from their original training frameworks to an embedded friendly format. These models have been slightly modified to ensure the highest performance when executed on TI Deep Learning Accelerators. Some of the tasks supported by the models in the ModelZoo are image classification, object detection, semantic segmentation, human position, any several more.

The cloud-based Edge AI Studio Model Analyzer provides an easy to use Model Selection tool. It is dynamically updated to include all models supported in TI EdgeAI-ModelZoo. The tool requires no previous experience and provides an easy to use interface to enter the features required in the desired model. The "Model Selection" tool suggests several object detection models for AM62A. Selecting the final model depends on the specific application and tasks complexity. The ONR-OD-8200-yolox-nano-lite-mmdetcoco-416x416 model is chosen for the defect detection demo. This model has several appealing features including low latency with the resolution that is adequate for this application. Table 3-1 lists the important features of the model selected for the defect detection demo. The details are for the model when trained on the coco dataset with 80 classes.

Table 3-1. Highlight Features of the Model yolox-nano-lite That are Used in the Defect Detection Demo (The details are for the model when trained on the coco dataset with 80 classes,)

Model	Task	Resolution	AP 50% Accuracy On COCO	Latency/Frame (ms)	DDR BW Utilization (MB/Frame)
YoloX-Nano-Lite	Multi Object Detection	416x416	40.1	8.88	22

3.2 Model Training and Compilation

The model is trained using TI Edge AI Studio Model Composer, an online application that provides a full suite of tools required for development of edge AI models including data capturing, labeling, training, compilation and deployment. For a detailed tutorial about using the Model Composer, see the Quick Start Guide. The Model Composer user interface shows tabs on the top of the window which are logically sorted to match the normal steps of model development for Edge AI applications. Users with no or low AI experience can simply follow these tabs to train and compile their model. Next are the steps followed to train and compile the model using Model Composer:

1. Open Model Composer and create a new project with "Object Detection" as Task Type as shown in Figure 3-1.

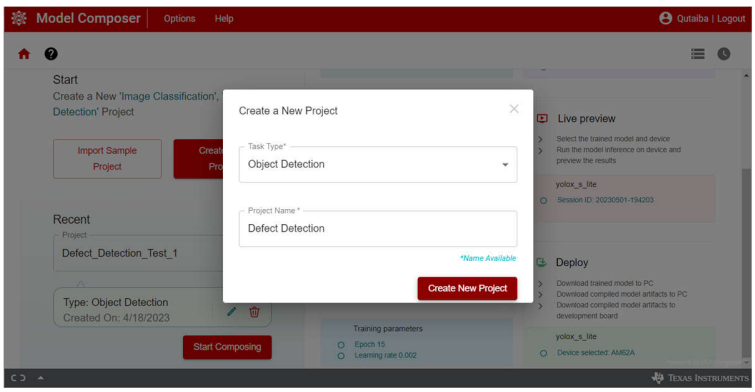

Figure 3-1. TI Edge AI Studio: Model Composer, Create a New Project

2. Upload the dataset to the project. On the “Capture” tab, open the “Input Source” menu and choose “Import Annotated Archive dataset” option as shown in Figure 3-2. Select the dataset and upload it to the project. The dataset should be compressed in tar or zip format. The defect detection dataset of 4800 pictures with the associated coco format annotation json file are compressed as a tar file and used in this step.

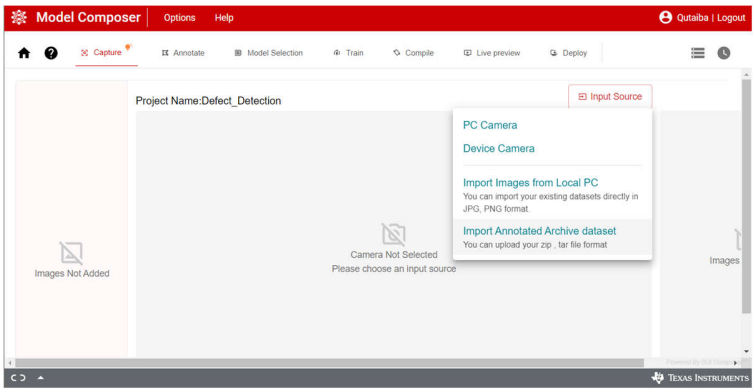

Figure 3-2. TI Edge AI Studio: Model Composer, Import Dataset

3. The Model Composer directly recognizes the coco format annotation json file and add the annotations to their respective files as can be seen in the “Annotation” tab as shown in Figure 3-3. Note that the Model composer provides tools for data capture and annotation which are handy but they are not used in this project as a custom augmentation process is used out of the model composer.

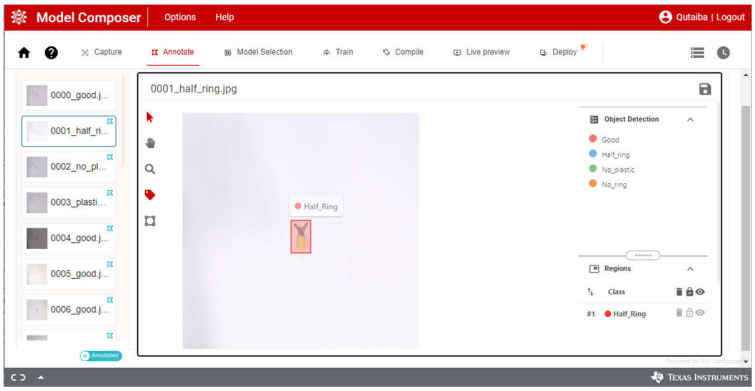

Figure 3-3. TI Edge AI Studio: Model Composer, Data Annotation

4. Move to the “Model Selection” tab and select AM62A in the Device selection panel and the yolox_nano_lite in the Model selection panel as shown in Figure 3-4.

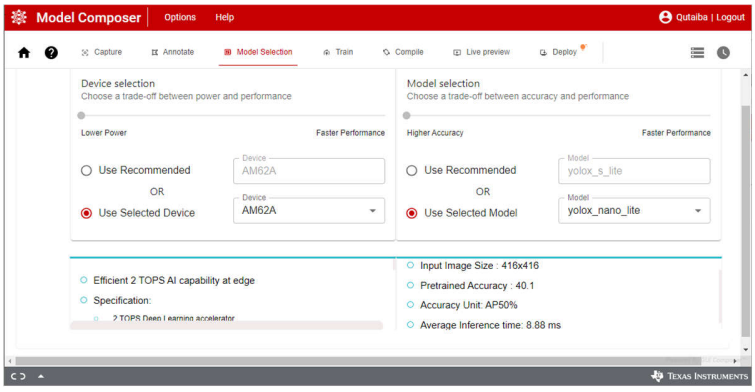

Figure 3-4. TI Edge AI Studio: Model Composer, Model Selection

5. Move to the “Train” tab and select the desired training parameters as shown in Figure 3-5. The following are the parameters used to train the model in this project. Feel free to experiment with other parameters which might fit your model and tasks.
   - Epochs: 10
   - Learning Rate: 0.002
   - Batch size: 8
   - Weight decay: 0.0001

   When satisfied with the parameter, click “Start Training” icon. The Model Composer, in the background, divides the dataset into three parts for training, testing and validation. As the training is underway, the performance is shown as a graph of Accuracy vs Epoch. The model in this project achieved 100% accuracy on the training.

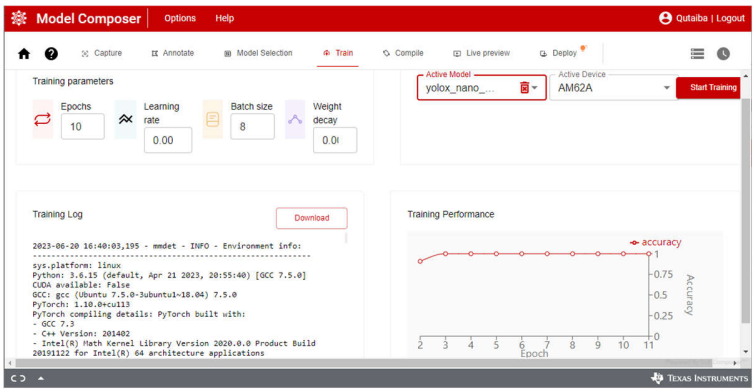

Figure 3-5. TI Edge AI Studio: Model Composer, Model Training

6. After training is completed, the mode is compiled to generate the artifacts for the model which are required to execute on the Deep Learning Accelerator of AM62A. Move to the “Compile” tab and select the desired compilation parameters as shown in Figure 3-6. Several factors are considered when selecting the compilation parameters including model type, the targeted accuracy, performance, and size of dataset. The model in this project is compiled with the default preset parameters as follows:
   - Calibration Frames: 10
   - Calibration Iterations: 10
   - Detection Threshold: 0.6
   - Detection Top K: 200
   - Sensor Bits: 8

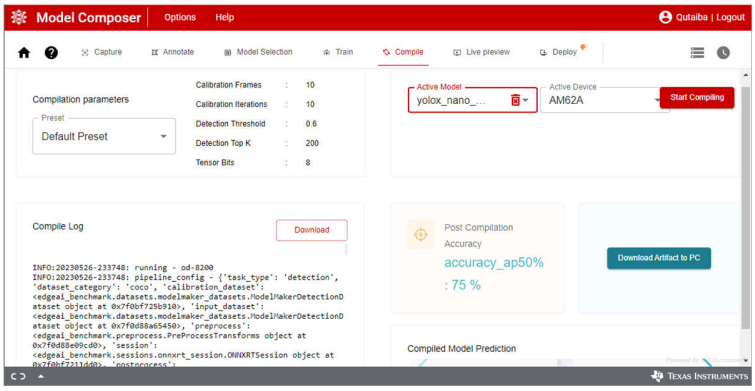

Figure 3-6. TI Edge AI Studio: Model Composer, Model Compilation

7. When compilation is completed, the artifacts are downloaded to AM62A. The model composer has tools for Live Preview and Deployment. The Live preview is used to test the model directly on the app as shown in Figure 3-7. This tool provides an easy method to check the model before deployment. This requires a camera to be connected to the AM62A EVM and that the AM62A EVM is connected to the same network as the hosting PC. The “Deploy” tool is used to download the compiled model artifacts directly to the EVM assuming that it is connected to the same network as the hosting PC. Alternatively, the model artifacts can be downloaded to the hosting PC as a tar file and then it can be transferred to the desired EVM.

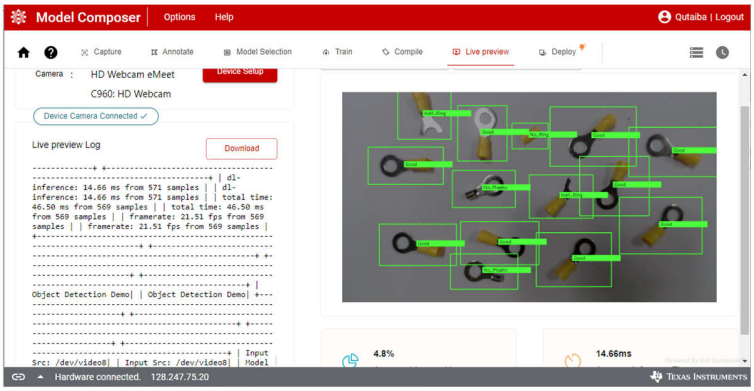

Figure 3-7. TI Edge AI Studio: Model Composer, Live Preview

The steps presented above provided comprehensive details to train and compile the model using Edge AI Studio model composer. At this point, the model artifacts are downloaded to the targeted EVM and are ready to be used in the end application.

4 Application Development

4.1 System Flow

This application is built based on TI’s edgeai-gst-apps project that includes all required infrastructures for an end-to-end processing application. It configures the model, creates the video processing pipeline using gstreamer, makes inference calls to the compiled model, and post processes the model’s output for visualization.

GStreamer is an open-source multimedia platform that links media processing systems as a pipeline. Some of the media handling processes include video capturing, recording, and streaming. TI provides a suit of gstreamer plugins that allow offloading some of the media processing and the deep learning inference to the hardware accelerators. The edgeai-gst-apps provides required software to configure and use the gstreamer pipeline. It is used to run the out of box demos on the AM62A EVM and its source code is available for developers to use.

The defect detection demo updates the post processing application code while keeping the gstreamer wrapper and the inference call part the same. The features added to the application code include object tracker, performance evaluation, graphical dashboard, and colorful bounding boxes. Figure 4-1 shows the full system flow updated for the defect detection demo with the application code and the gstreamer. The application has been tested with two types of cameras including a USB camera and an IMX219 camera with a CSI interface.

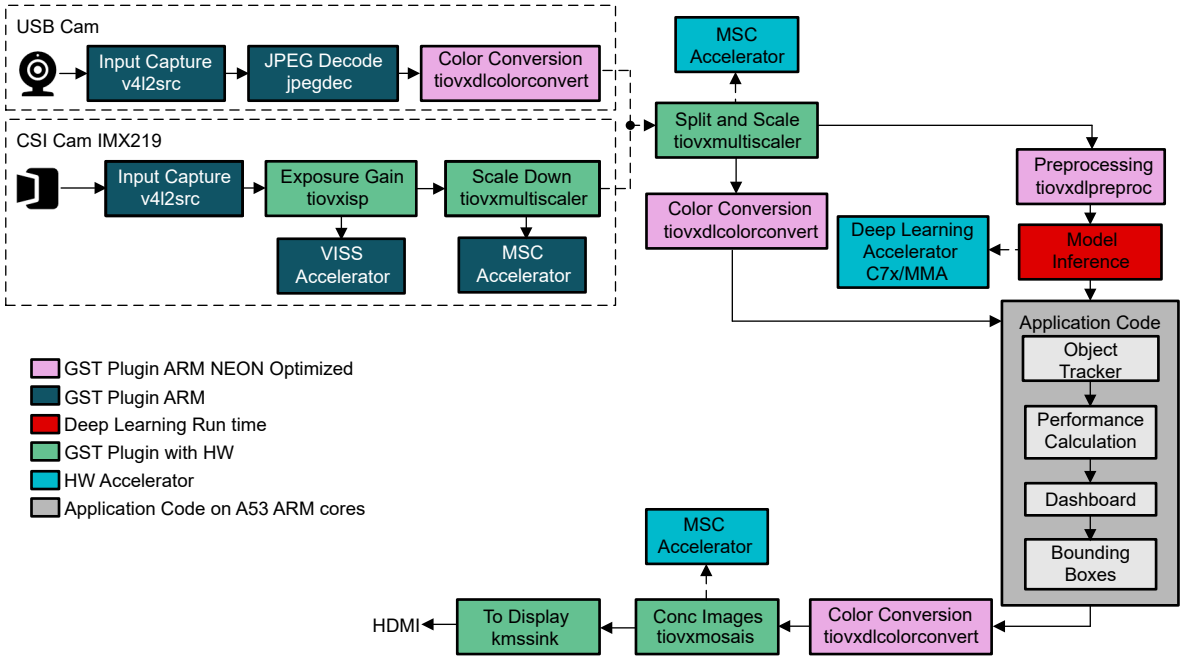

Figure 4-1. Full System Flow Including Application Code and gstreamer Pipeline With Two input Options: a USB camera or a CSI IMX219 camera. The defect detection application supports the two input options.

4.2 Object Tracker

The object tracker is used to provide accurate coordinates of the units detected in the frame. This information is used to count the total number of units and the number of units for each class. More important, the coordinates produced by the object tracker can be fed to a sorting and filtering mechanism in the production line. Details on the object tracker can be found in the source code in the github repository in objects_tracker.py.

4.3 Dashboard and Bounding Boxes Drawing

The dashboard graphically shows an overview of the performance of the whole manufacturing system including the total number of units, the percentage of the defected units, and the rate of production in units per hour. It also shows a histogram of the types of defects. Such information is useful to analyze the manufacturing system and select the most common types of defects. The dashboard code is contained in its own class which is saved in the dashboard.py file. A new class is added to the post_process.py to control all post process work related to the defect detection demo including calling the object tracker, performance statistics calculation, calling dashboard generator, and draw bounding boxes.

4.4 Physical Demo Setup

The defect detection application is designed to work on a production line where units are moving on a conveyor belt. For demo purpose, a rotating table is used to emulate the conveyor belt. Figure 4-2 shows the physical setup of the defect detection demo. The camera is positioned at a height to include only a square part of the rotating table in the frame. When the table rotates, the units on it will appear to the camera as if they were placed on a linear conveyor belt. This setup simplifies the demo to eliminate the need to reload the units.

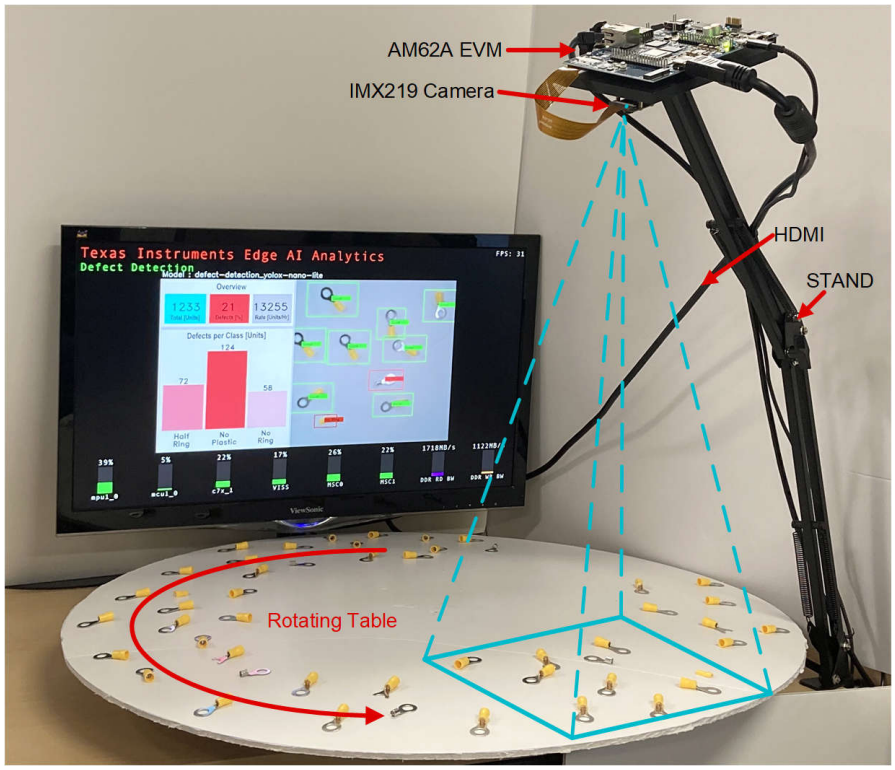

Figure 4-2. Defect Detection Demo Setup (Only part of the rotating table is within camera frame to emulate a conveyor belt.)

5 Performance Analysis

5.1 System Accuracy

Several experiments were implemented to extensively test the accuracy of the entire system which includes the accuracy of both the yolox-nano-lite model, trained on the defect detection data, the object tracker, and the graphical dashboard. The experiments test the defect detection application live on testing samples (ring terminals). A collection of samples with predefined combination of classes (good, half ring, no plastic, and no ring) are placed the rotating table to simulate moving on a conveyor belt. The application is used to detect the samples and the results shown on the dashboard are compared with the actual statistics of the samples.

Table 5-1 shows the details of one experiment which includes a total of 50 samples with 20 % defected samples distributed as following: 3 half ring, 5 no plastic, and 2 no ring.

Table 5-1. Accuracy Experiment Details of Defect Detection Application (it shows 100 % accuracy with 50 samples and 10 repeated tests)

Class	Ground Truth No. and [%]	Application Results After 10 Rounds	Accuracy
Total Samples (Not a class)	50	500	NA
Good	40 [80%]	400 [80%]	100 %
Half Ring	3 [6%]	30 [6%]	100 %
No Plastic	5 [10%]	50 [10%]	100 %
No Ring	2 [4%]	20 [4%]	100 %

The samples are randomly placed on the rotating table and the application is used to detect them. The table with the samples were rotated for 10 rounds for repeatability assurance. Figure 5-1 shows the dashboard at the end of the tenth repeated tests. Comparing the results on the dashboard generated by the application with the ground truth input samples showed that the application successfully detected all good and defected units in all ten rounds as shown in Table 5-1.

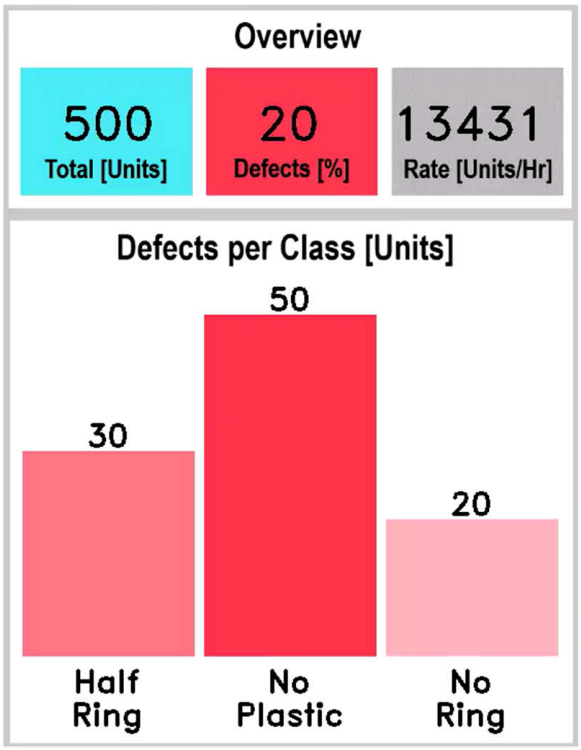

Figure 5-1. Dashboard Results of the Accuracy Experiment With 50 Samples and 10 Repeated Tests (the defect detection application achieved 100 % accuracy)

5.2 Frame Rate

The defect detection application showed a frame rate of 30 FPS. This rate is limited by the performance of the camera used in the application which does not exceed 30 FPS. For this reason, only 22 % of the deep learning accelerator C7x/MMA is utilized. In the same time, only 39 % of the main processing cores is utilized. This shows a high expansion margin for both the main processing power and the deep learning accelerator to support a camera with a higher frame rate. The next section discusses core loading utilization in details.

5.3 Cores Utilization

The AM62A SoC consists of various processing cores and hardware accelerators. Monitoring the loads on these components is important to explore the whole system capabilities and the expansion opportunity. The defect detection demo uses tiperfoverlay gstreamer plugin to show core loads as a bar graph at the bottom of the screen. Figure 5-2 shows a screenshot of the core loads graph of AM62A while running the defect detection demo. By default, the graph is updated every two seconds to show the loads as a utilization percentage. In addition to the tiperfoverlay gstreamer plugin, the perf_stats tool is a second option to show cores performance directly on the terminal with an option for file save. This option is more accurate compared to the tiperfoverlay as the later adds extra load on the Arm cores and the DDR to draw the graph and overlay it on the screen.

Figure 5-2. Core Load Graph Bar Shown at the Bottom of the Defect Detection Demo Using tiperfoverlay gstreamer Plugin (the figure is edited to appropriately fit in the page)

The graph shown in Figure 5-2 shows that the defect detection demo in addition to the whole supporting Linux processes utilizes about only 39% of the Arm cores capacity (averaged across four A53 cores). In the same time the yolox-nano-lite used in the application utilizes about 22% of the C7xMMA deep learning accelerator. It is important to note that in this experiment, the C7xMMA is clocked at 850 MHz instead of 1000 MHz. In other words, if the C7xMMA accelerator was clocked at 1000 MHz, its utilization will be less than the reported 22%. The DDR used for read operations is 1706 MB/s and for write operations is 1118 MB/s resulting in a total of 2824 MB/s operations. The AM62A supports a total DDR band of 12.8 GB/s when using 32 bit DDR4 at 3200 MT/s. The total 2824 MB/s utilizes about 22 % of the total DDR bandwidth.

These low utilization values of the Arm cores, accelerators, and DDR bandwidth indicate that there is a big room for expansion on the AM62A to run additional applications or to expand the defect detection application itself such as increasing the frame rate by using another faster camera. In addition, the low cores utilization provides flexibility to select the right SoC variant of AM6A. The core loads shown in Figure 5-2 are for the AM62A74 variant of the SOC AM62A family. This variant is equipped with four A53 Arm cores and a C7xMMA deep learning accelerators capable of executing two TOPS. The low utilization values suggest that the defect detection demo in its current form can be implemented on other lower end variants of the AM62A such as AM62A3, which includes two Arm cores and one TOPS deep learning accelerator.

5.4 Power Consumption

The power consumption of the AM62A SoC while running the defect detection demo can be estimated using the Power Estimation Tool (PET). This tool is built based on measured and simulated data. Most of the measured data are collected from bare-metal tests with no operating system. The tool estimates power based on clock frequency and utilization of the various components of the AM62A in addition to other factors such as the expected temperature. The cores utilization data presented in the previous section is used to estimate the power consumption of the entire system.

Table 5-2 lists the estimated power for the AM62A running defect detection demo with a summary of the important core utilization used for the power estimation. The PET estimates a total of 1.43 W consumed by the AM62A based on the core utilization for the defect detection application.

Table 5-2. Cores Loading Utilization and Power Estimation of AM62A While Running the Defect Detection Application

Main IP	Core Loading Utilization/Power
ARM A-53	39 [%] at 1.25 [GHz]
Deep Learning C7x/MMA	22 [%] at 850 [MHz]
DDR BW	22 [%]
VPAC (ISP)	20 [%]
Power Consumption Est using PET at 85°C Core Voltage at 0.75 V	1435 [mW]

6 Summary

The AM62A is an ideal option for defect detection for manufacturing applications. It is equipped with different hardware accelerators optimized for edge AI applications. This document presents the steps to develop a defect detection demo using AM62A. The application shows high accuracy and low estimated power active consumption. The source code for the demo is publicly available and can be accessed at https://github.com/TexasInstruments/edgeai-gst-apps-defect-detection.

Note

This demo is originally created and tested using Linux SDK 8.06. Submit any issues with newer SDK releases to TI E2E Support Forums.

7 References

1. Defect Detection Source Code
2. AM62A7, AM62A3
3. SK-AM62A-LP Starter Kit (SK) Evaluation Module (EVM)
4. Texas Instruments: *Sitara AM62Ax Benchmarks*
5. Texas Instruments: *AM62Ax Sitara Processor Data Sheet*
6. Edge AI ModelZoo
7. Edge AI Studio: Model Analyzer
8. Edge AI Studio: Model Composer
9. Performance Statistics
10. Texas Instruments: *AM62A Power Estimation Tool*

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265