

CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

Edge AI on Kria KR260: Implementing ResNet-50 Classification Using DPU TRD and YOLOX
Detection with DPU-PYNQ

A graduate project submitted in partial fulfillment of the requirements
For the degree of Master of Science in Electrical Engineering

By
Madhusudhan Chandavare Gowda

May 2025

The graduate project of Madhusudhan Chandavare Gowda is approved:

Professor Saba Janamian

Date

Dr. Shahnam Mirzaei

Date

Professor Xojun Geng, Chair

Date

California State University, Northridge

Acknowledgements

I would like to express my sincere gratitude to Professor Saba Janamian for serving as my graduate advisor throughout my time at California State University, Northridge. His guidance, support, and invaluable input were instrumental to the success of this project.

I would also like to thank my parents for their unwavering support, encouragement, and belief in me. Without their constant motivation and direction, none of my opportunities for success would have been possible.

Table of Contents

Signatures	ii
Acknowledgements	iii
Abstract	iv
Chapter 1: Introduction.....	1
Chapter 2: Literature Review.....	3
Chapter 3: System Design and Methodology.....	7
Chapter 4: Implementation.....	10
Chapter 5: Result and Future Work.....	12
Conclusion.....	13
References.....	14

Abstract
Hardware-Accelerated Deep Learning for Robotic Vision Using
Xilinx DPU on Kria KR260

By
Madhusudhan Chandavare Gowda
Master of Science in Electrical Engineering

The rapid development of artificial intelligence at the edge has created a lot of demand for specialized hardware accelerators capable for supporting deep learning models with minimal latency and power consumption. This project focuses on deploying a custom-designed Deep Learning Processing Unit (DPU) on the Kria KR260 to accelerate the workload of deep learning models: image classification using ResNet-50 and object detection using YOLOX.

The primary purpose of the project is to evaluate the easiness in implementation, reusability and performance of DPU across multiple deployment environments specifically petalinux for embedded applications and Ubuntu with Python based interfaces via DPU-PYNQ. A DPU overlay was generated using Vivado with the use of DPUCZDX8G and modifying it for KR260 platform. After synthesis and implementation runs the .xsa , .bit and handoff files were exported for further use. Two deployments were done with the exported files: the first involving building of a petalinux image with integrated Vitis AI 3.0 runtime to execute Resnet-50 model, the second used Ubuntu 22.04 to run the DPU-Pynq 3.5 framework allowing real-time YOLOX interface through a PYNQ interface.

Performance evaluation showed that the DPU provided low-latency, high-throughput inference in both workflows. The ResNet-50 model achieved fast classification times on static images, while the YOLOX pipeline supported real-time object detection at approximately 15–20 FPS. Both implementations demonstrated successful integration of DPU architecture.

To sum up, this project shows how capable and adaptable the DPU is on the Kria KR260, making it a solid foundation for both low-level embedded tasks and more advanced AI applications. These results point to its strong potential in real-world edge computing scenarios.

Chapter 1: Introduction

1.1 Overview

The field of robotic vision has progressed exponentially due to the advent of deep learning, which has significantly improved perception, decision-making, and autonomy. Robotic systems are becoming deployed more frequently in dynamic environments, where fast and accurate recognition of objects and situational awareness are required. Due to resource capabilities and then power constraints, satisfying these requirements is especially difficult on edge devices.

However, new edge AI hardware such as the Xilinx Deep Processing Unit (DPU) allow for a powerful but efficient way to deploy deep neural networks in real time. The KR260 is focused on edge deployment and is paired with the Vitis AI tools, with support for hardware acceleration to provide a simplified stack for Deep learning model development and deployment — from Resnet50 and beyond.

1.2 Motivation and Objectives

The purpose of this project is to provide a whole, efficient and deployable AI solution for robotic vision using widespread commercial hardware. Therefore, the main focus is to utilize the Xilinx DPU overlay on the Kria KR260 platform to execute the Resnet50-Object classification YOLOX-object identification model in real-time using a USB camera using different frameworks

This project aims to:

- Design a hardware design with DPU IP using Xilinx Vivado 2022.2
- DPU TRD Petalinux framework:
 - Build an operating system with DPU Overlay using Petalinux 2022.2
 - Add Vitis AI 3.0 to the build of the project
 - Deploying the Resnet50 Deep Learning model using Vitis AI Model Zoo for object using the DPU
- DPU PYNQ framework
 - Deploy Ubuntu 2022.04 image and download respective required libraries.

- Installing GStreamer, libuvc and v4l2loopback-dkms libraries for enabling live streaming
- Connect a USB camera to the system for live inferences
- Build out the end-to-end pipeline in Python for running the model, post-processing for interpretable outputs, and visualization

1.4 Scope and Contributions

This graduate project is designed from ground up for embedded platforms, it offers the implementation of a hardware accelerated robotic vision system. It does not contain a mobile robotic platform for physical deployment but facilitates inference from video streams to mimic real-world application.

Key contributions include:

- Successful creation of DPU overlay of B4096 for Kria KR260 platform
- Accelerated hardware inference with Vitis AI 3.0: a real-world example
- YOLOX based USB camera based vision system
- Resnet50 based image classification app
- Quantitative analysis of the deployed model on Kria KR260
- Documentation and discussion of a reproducible robotic pipeline

This paper shows the promise of edge computing in improving robotic vision and perception by proving that modern AI models can be efficiently used on embedded systems with limited resources.

Chapter 2: Literature Review

2.1 Deep Learning in Embedded Platforms

Deep learning is one of the defining technologies of our time, fueling breakthroughs in computer vision, natural language processing, robotics and more. Nevertheless, most conventional deep learning architectures utilize cloud-centric resources, which renders them unsuitable for applications that demand low-latency and real-time processing capabilities. Cloud computing brings the computational power needed to process and analyze data, however it has certain limitations in terms of bandwidth, latency and privacy.

Model Pruning, Quantization and Neural Architecture Search (NAS) are some of many techniques proposed to create lightweight deep neural networks (DNNS) that are able to perform well on edge devices. Goals for edge AI are performance, power consumption, and flexibility, with popular platforms being the NVIDIA Jetson family, Intel Movidius Myriad X, and the Xilinx. These advancements have enabled us to deploy state-of-the-art models in real-time through embedded systems.

2.2 DPUs for Accelerating Deep Learning Workloads

The Xilinx DPU Deep Processing Unit is a custom hardware accelerator built for running convolutional neural networks (CNN) on Xilinx Field Gate Arrays (FPGA) and SoC. Vitis AI -- tightly integrated with the Vitis AI development environment that provides high-level model compilation, optimization, and deployment. The DPU is fully compatible with a broad spectrum of neural network architecture and supports efficient computation for convolutional layers, pooling layers, and activation functions.

The architecture of the DPU is application-agnostic as it can be customized with varying kernel sizes, multiple compute engines, and wide memory accesses, among other architectural parameters. It uses quantized 8-bit integer models, which require much less memory and compute resources compared with their floating-point counterparts. It makes ideal for low-power, high-performance embedded inference tasks.

2.3 Petalinux 2022.2 and DPU TRD

Xilinx's embedded Linux development environment Petalinux 2022.2 simplifies the building of customized Linux systems for SoCs, MPSoCs, and ACAPs like Zynq UltraScale+ and Kria Configuring and building the Linux kernel, device trees, bootloaders (U-Boot), and root filesystems is easier with this release. Developers can change system components and integrate hardware accelerators like the DPU

with petalinux-config and petalinux-build. The root filesystem can also contain custom packages, apps, and libraries in Petalinux 2022.2.

The DPU TRD (Deep Processing Unit Target Reference Design) is a pre-validated design framework provided by Xilinx (now AMD) to help developers quickly deploy AI inference on edge. It integrates complete software and hardware stack for running AI models efficiently on custom FPGA designs. Central to the TRD is the DPU core, which accelerates deep learning inference. The B4096 architecture within this design is a high-performance variant optimized for deep neural. The "B" denotes the type of DPU core, and "4096" represents the number of MACs (Multiply-Accumulate Units), enabling it to process large convolutional layers with high throughput. It supports INT8 quantized models, maximizing performance and power efficiency.

2.3 Efficient Object Detection with YOLOX

YOLOX is a high-performance object detection model that belongs to the "You Only Look Once" (YOLO) family, designed for real-time applications. Unlike its predecessors, YOLOX introduces anchor-free detection and decouples the head for classification and regression, enhancing both speed and accuracy. It uses a simplified training pipeline and supports models ranging from YOLOX-Nano to YOLOX-Large, catering to diverse computing environments. YOLOX also benefits from advanced augmentation techniques like Mosaic and MixUp, and leverages the Efficient Layer Aggregation Network (ELAN) backbone for improved feature extraction. These optimizations make YOLOX well-suited for deployment on edge devices, including FPGAs and embedded platforms, where performance per watt is critical. Its open-source nature and PyTorch implementation further promote adaptability across research and industry applications.

2.4 ResNet-50 for Image Classification

ResNet-50 is a deep convolutional neural network architecture that was introduced in the paper Deep Residual Learning for Image Recognition. The architecture, consisting of 50 layers, has achieved some state-of-the-art accuracies on various image classification tasks while also being computationally efficient. The IFN employs Residual connections that speed up the convergence of the network and improve the vanishing gradient problem, making it deployable on both cloud and embedded platforms. When optimized with quantization and pruning, ResNet-50 can achieve real-time inference throughput on the DPU and maintain classification accuracy on all dataset varieties. This provides a very strong benchmark for classification capabilities in embedded vision applications.

2.5 Trends in Robotic Vision and Edge AI Applications:

Recently, the combination of AI, embedded computing and mechatronics has been revolutionizing robotic vision. This capacity for environmental awareness is essential to a variety of robotic systems, from navigation to object manipulation and interaction. The need for real-time vision systems is growing as field robots operate in more complex and dynamic environments. Edge AI is an essential component to making this transition possible. They also leverage embedded AI accelerators, allowing the robot to process visual information locally and avoid cloud infrastructure latency, increasing its responsiveness. Many use cases illustrate this trend:

- Autonomous mobile robots for warehouse jobs
- Agricultural robots for plant detection and health monitoring
- Drones for real-time object tracking and terrain mapping and UAVs
- Robots for human interaction and gesture recognition

These cases show the acquisition for edge computing in robotic vision applications. Developers can maintain a scale of readability, power utilization, and correct functionality, through high-performance embedded platforms.

2.6 Research Gaps and Project Relevance

Although many advances are made in AI and Embedded systems, there are still some challenges to attain optimal performance without sacrificing performance while keeping the accuracy low. Most edge deployments are limited by available hardware resources, thermal limitations, or are deployed through complex pipelines. Furthermore, even though Xilinx's Deep Processing Unit (DPU) is an extremely powerful AI accelerator, its potential has not been fully realized in real-world robotic vision systems. Moreover, the scarcity of real-world examples and DPU-based inference in open-source projects and tutorials is indicative of a gap that needs to be filled in the applied research and documentation.

This project overcomes these challenges by building a full vision pipeline on the Xilinx Kria KR260 platform and leveraging two complimentary AI models with two different frameworks, YOLOX for detection on Ubuntu with DPU PYNQ framework and ResNet-50 for classification with DPU TRD using Petalinux. This work also contributes to filling the knowledge and application gap of DPU-based embedded AI systems by proposing a generic modular pipeline utilizing the DPU capabilities.

The combination of hardware-accelerated inference, real-time video processing, edge deployment, and integration into a single unified system is a practical solution to these challenges. It further adds to the existing literature by demonstrating that modern large-scale AI models are deployable on economical FPGA-based hardware.

Chapter 3: System Design and Methodology

3.1 System Architecture Overview

The system architecture is based on the deployment of deep learning models on Kria KR260 platform using DPU. The overall design consists of two separate workflows:

- Workflow A: Image classification using ResNet-50 model with the customized DPU Targeted Reference Design (TRD) under Petalinux.
- Workflow B: Real-time object detection using YOLOX-Nano model with DPU-PYNQ framework on Ubuntu using Python.

Both workflows use the same DPU Overlay but differ in software stack and deployment tools.

3.2 Hardware Setup

Both the workflows use the same hardware platform Kria KR260 . This board has a Zynq Ultrascale MPSoC Processor for processing and FPGA-based Programmable logic

Hardware components include:

- Kria KR260 Development Board
- SD card 128gb for OS and file system
- Logitech USB3.0 webcam for real time video input
- Ethernet and UART for board access
- Keyboard, Mouse and Monitor and respective cables

3.3 Software stack overview:

The two workflows use different software environments:

- Workflow A – Petalinux
 - Petalinux 2022.2
 - Vitis AI 3.0 and VART
 - Prebuilt ResNet-50 model from Vitis AI Model Zoo
- Workflow B – Ubuntu with PYNQ
 - Ubuntu 2022.04 for kria KR260
 - Vitis AI 3.5, DPU- PYNQ 3.5
 - Python3.8, OpenCV, GStreamer,libv4l2
 - YOLOX-Nano Model form Vitis AI model Zoo 3.5

Both the workflow work on the same DPU overlay but different environments and programming interfaces.

3.4 Workflow A: Resnet-50 Deployment with Petalinux DPU TRD:

In this workflow the DPU.xsa file from vivado is loaded into the petalinux project which also contains the Kria KR260 Petalinux 2022.2 BSP from Xilinx. After loading the FPGA manager is enabled, TFTPBoot copy is disabled and Image Package type set to INITRD using “petalinux config”. In the next step the kernel is configured to enable the DPU Drivers and required files are added to the petalinux project and petalinuxbsp.conf, user-rootfs.conf are edited and project is built and also the WIC image is also created for the SD card. After that the following steps are involved to successfully run ResNet-50 :

- Export Bit.bin and shell.json from the project to the board
- Load the DPU Overlay via Boot.bin and device tree overlays
- Deploy precompiled Resnet-50 Model onto the board
- Use a C++ sample application provided by VART to run inference on static image datasets.
- Output the top-5 classification results and inference time per image.

3.5 Workflow-B: YOLOX Deployment with Ubuntu and DPU-PYNQ:

In this workflow Ubuntu 2022.04 image is booted on the Kria Platform and the Kria PYNQ is installed in it and vai 3.5 is downloaded and then installed. Following this, the DPU-PYNQ 3.5 framework is configured to support Python-based real-time inference. The YOLOX-nano 640x480 is then loaded with a Python script to run the model on real-time video.

Steps involved:

- Load DPU bitstream and overlays using DPU-PYNQ
- Connect usb camera for live video input
- Preprocess video frames using OpenCV
- Run the YOLOX model using VART
- Post-processing the results using NMS and overlay bounding boxes
- Display the result with annotated detections in real time

3.6 Python file pipeline: The object detection pipeline is a single python script running on ubuntu using DPU-PYNQ and VART API's. It processes the real-time feed with the yolox model and streams the processed video feed. GStreamer is used for high-performance video output and OpenCV for processing and display.

Key Functional components:

- Model loading and Overlay Initialization: The DPU Overlay “dpu.bit” is loaded via the DPU-PYNQ framework and the YOLOX-Nano model is prepared for texecution. For the classifications the COC.txt file is also loaded.
- Video Capture: GStreamer interfaces with the webcam, which captures a 640x480 video stream. OpenCV reads the frames from this stream for inference.

- Preprocessing: The captured frame is resized and padded to 416x416 pixels which is the input size for the model, normalized and reshaped to the tensor format for DPU compatibility
- Inference execution: The VART runner handles the asynchronous execution of the input tensor on the DPU. The model returns three output tensors, which are concatenated and decoded.
- The postprocessing pipeline includes bounding box transformation, application of sigmoid and SoftMax functions, and Non-Maximum Suppression (NMS). Final bounding boxes are scaled back to the original frame dimensions.
- Visualization:
- Detected objects are visualized by drawing bounding boxes and class labels on the original frames using OpenCV. Performance stats such as FPS and latency are logged to the console

Chapter 4: Implementation

This section provides the steps that were followed to deploy each of the two workflows onto the Kria KR260 platform. Both workflows have a common factor that is the hardware foundation—a custom DPU overlay created using Vivado

Steps to create the xsa file:

- New Vivado project was created with a base platform being Kria KR260
- The DPUCZDX8G is imported and is configured to B4096 architecture.
- The required AXI interconnects, clock/reset/logic, and mem controllers.
- Integrate the DPU with the processing System using Vitis
- Add the DPU IP block from the IP catalog and configure DPUCZDX8G with B4096
- Generate output products and run synthesis and implementation
- Generate bitstream and hardware handoff files
- Export the .xsa files for further processes

The exported .xsa file is further used to build the embedded image for the Workflow A, and the .bit and .hwh files from the same vivado project is used in Workflow B

The .xsa file is center to both workflows, thus ensuring the use of same DPU overlay used for both workflows.

4.1 Workflow A: Resnet-50 on petalinux using DPU TRD

This workflow begins with the creation of the Petalinux Project. Petalinux project is created with petalinux-create, the PetaLinux project was set up using petalinux-config to include:

- DPU integration device tree overlays
- SD card boot choices
- Support for USB and UVC camera drivers
- Packages NumPy, OpenCV, and Python 3.8

Then it is built with petalinux-build, the system image creates boot files such as rootfs.ext4, image.ub, and BOOT.BIN. The KR260 was booted from these written to the SD card.

The Vitis AI DPU overlay, built in Vivado, was implemented on the KR260 as a component of the programmable logic bitstream. Upon booting, the device tree overlay initializes the DPU kernel. The ResNet-50 .xmodel files were transferred to the root disk. Execute the resnet-50.xmodel using the VART API to load the model, preprocess input images, run inference, and display classification results.

4.2 Workflow-B: YOLOX on Ubuntu Using DPU-PYNQ:

This workflow is deployed on the Ubuntu 2022.2 OS on Kria KR260. The board is set up with Vitis AI 3.5 and DPU_PYNQ 3.5 framework. DPU is enabled with the custom written python file and the VART API

Steps involved are:

- Flash the Ubuntu 2022.04 on a bootable SD card and set up the KR260
- Install the required libraries and additionally install Python3.8, OpenCV, GStreamer, libuvc and v4l2loopback for real-time video processing and streaming.
- Download and install Vitis AI 3.5 and DPU-PYNQ framework.
- Place the dpu.bit, dpu.hwh and the YOLOX.xmodel files into the correct overlay directory
- With a custom written python file load the model and initialize the overlay, capture the 640x480 video stream, preprocess captured frames to 416x416 pixels, execute the model on the frames, postprocessing the executed frames, and convert it back into original frames for video and video output
- Connect the usb camera and execute the Python file

4.3 Challenges faced

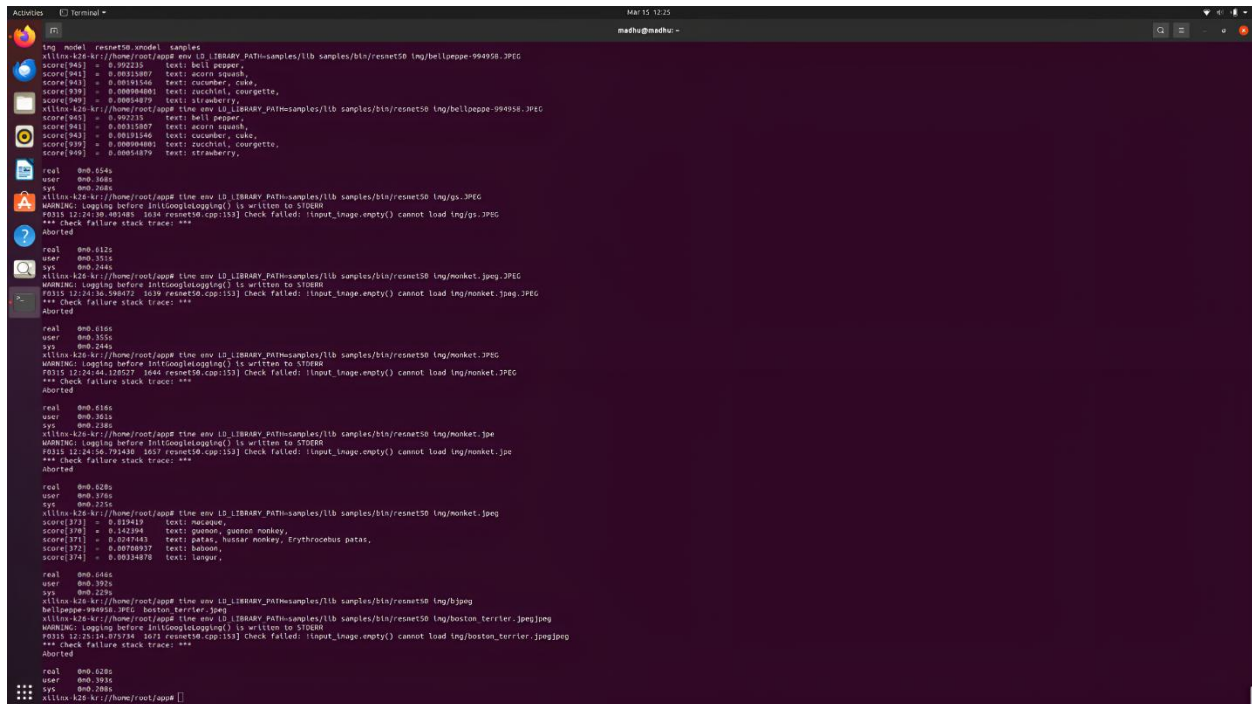
- DPU overlay timing closure: Needed careful Vivado routing and placement
- Not all UVC-compliant cameras acted consistently; firmware upgrades were required for the Kria KR260 board.
- Preprocessing the video frames compatible to the model
- Installing vai 3.5 and vai 3.0 on the Kria KR260 board as it is not available for it officially.

Chapter 5: Results and Future Work

This project aimed mostly to confirm the applicability of custom Deep Processing Unit (DPU) overlay on the AMD Kria KR260 platform for real-time vision AI applications. The implementation concentrated on two distinct workflows: one for image classification using ResNet-50 and the other for real-time object detection using YOLOX-Nano. Though they used various software environments, every workflow shared the same basic DPU architecture.

5.1 Workflow-A: ResNet-50 running on Petalinux

Using the custom .xsa file produced in Vivado, a Petalinux 2022.2 image was successfully run with the ResNet-50 model. During boot, the DPU was correctly integrated and loaded. The Vitis AI Runtime (VART) C++ APIs deployed the precompiled ResNet-50.xmodel. A batch of static photos was used to test the system; classification outcomes were shown on the terminal. The output class labels met expectations, therefore verifying the proper operation of the model. About 0.2 milliseconds per image was the average processing time, suggesting effective DPU hardware acceleration.



```
img_model resnet50_xmodel samples
all:ins-k26-kr1/home/root/app# time env LD_LIBRARY_PATH=samples/lib samples/bin/resnet50 img/bellpepper-994958.jpeg
score[0] = 0.902135 text: bell pepper,
score[941] = 0.0033887 text: acorn squash,
score[412] = 0.00181504 text: cucumber, cuke,
score[939] = 0.000804081 text: zucchini, courgette,
score[940] = 0.00054879 text: strawberry,
all:ins-k26-kr1/home/root/app# time env LD_LIBRARY_PATH=samples/lib samples/bin/resnet50 img/bellpepper-994958.jpeg
score[0] = 0.902135 text: bell pepper,
score[941] = 0.0033887 text: acorn squash,
score[942] = 0.00181504 text: cucumber, cuke,
score[939] = 0.000804081 text: zucchini, courgette,
score[940] = 0.00054879 text: strawberry,
real 0m0.614s
user 0m0.308s
sys 0m0.286s
all:ins-k26-kr1/home/root/app# time env LD_LIBRARY_PATH=samples/lib samples/bin/resnet50 img/gs.jpeg
WARNING: logging before initGoogleLogging() is written to STDERR
F0115 22:24:38.492485 1634 resnet50.cpp:153] Check failed: !input_image.empty() cannot load img/gs.jpeg
*** Check failure stack trace: ***
Aborted
real 0m0.612s
user 0m0.311s
sys 0m0.244s
all:ins-k26-kr1/home/root/app# time env LD_LIBRARY_PATH=samples/lib samples/bin/resnet50 img/monket.jpeg.jpeg
WARNING: logging before initGoogleLogging() is written to STDERR
F0115 22:24:38.596472 1639 resnet50.cpp:153] Check failed: !input_image.empty() cannot load img/monket.jpeg.jpeg
*** Check failure stack trace: ***
Aborted
real 0m0.616s
user 0m0.303s
sys 0m0.244s
all:ins-k26-kr1/home/root/app# time env LD_LIBRARY_PATH=samples/lib samples/bin/resnet50 img/monket.jpeg
WARNING: logging before initGoogleLogging() is written to STDERR
F0115 22:24:44.125227 1644 resnet50.cpp:153] Check failed: !input_image.empty() cannot load img/monket.jpeg
*** Check failure stack trace: ***
Aborted
real 0m0.616s
user 0m0.303s
sys 0m0.238s
all:ins-k26-kr1/home/root/app# time env LD_LIBRARY_PATH=samples/lib samples/bin/resnet50 img/monket.jpg
WARNING: logging before initGoogleLogging() is written to STDERR
F0115 22:24:58.793428 1657 resnet50.cpp:153] Check failed: !input_image.empty() cannot load img/monket.jpg
*** Check failure stack trace: ***
Aborted
real 0m0.628s
user 0m0.316s
sys 0m0.224s
all:ins-k26-kr1/home/root/app# time env LD_LIBRARY_PATH=samples/lib samples/bin/resnet50 img/monket.jpeg
score[273] = 0.829419 text: racoon,
score[978] = 0.8142294 text: guinea, guinea monkey,
score[271] = 0.8247463 text: patas, lesser monkey, Erythrocebus patas,
score[272] = 0.80780937 text: baboon,
score[176] = 0.80338076 text: langur,
real 0m0.646s
user 0m0.392s
sys 0m0.229s
all:ins-k26-kr1/home/root/app# time env LD_LIBRARY_PATH=samples/lib samples/bin/resnet50 img/bj.jpg
bellpepper-994958.jpeg boston_terrier.jpeg
all:ins-k26-kr1/home/root/app# time env LD_LIBRARY_PATH=samples/lib samples/bin/resnet50 img/boston_terrier.jpeg.jpeg
WARNING: logging before initGoogleLogging() is written to STDERR
F0115 22:25:16.870734 1671 resnet50.cpp:153] Check failed: !input_image.empty() cannot load img/boston_terrier.jpeg.jpeg
*** Check failure stack trace: ***
Aborted
real 0m0.628s
user 0m0.393s
sys 0m0.268s
all:ins-k26-kr1/home/root/app#
```

Figure 1: Resnet-50 output

5.2 Workflow B: DPU-PYNQ on Ubuntu running YOLOX

The YOLOX-Nano model was run in real-time on an Ubuntu 22.04 system operating on the KR260 in the second deployment. Video frames were streamed by a Python-based application running on DPU-PYNQ 3.5 using a USB webcam. The model was deployed, compiled, and successfully quantized. Running the pipeline produced a live video stream with identified objects overlaid with bounding boxes and class labels. Depending on lighting and scene complexity, the performance seen varied from 10-20 frames per second (FPS) and a total runtime of 0.02-0.09 seconds. The output's stability and reactivity verify the system's capacity for live object detection.

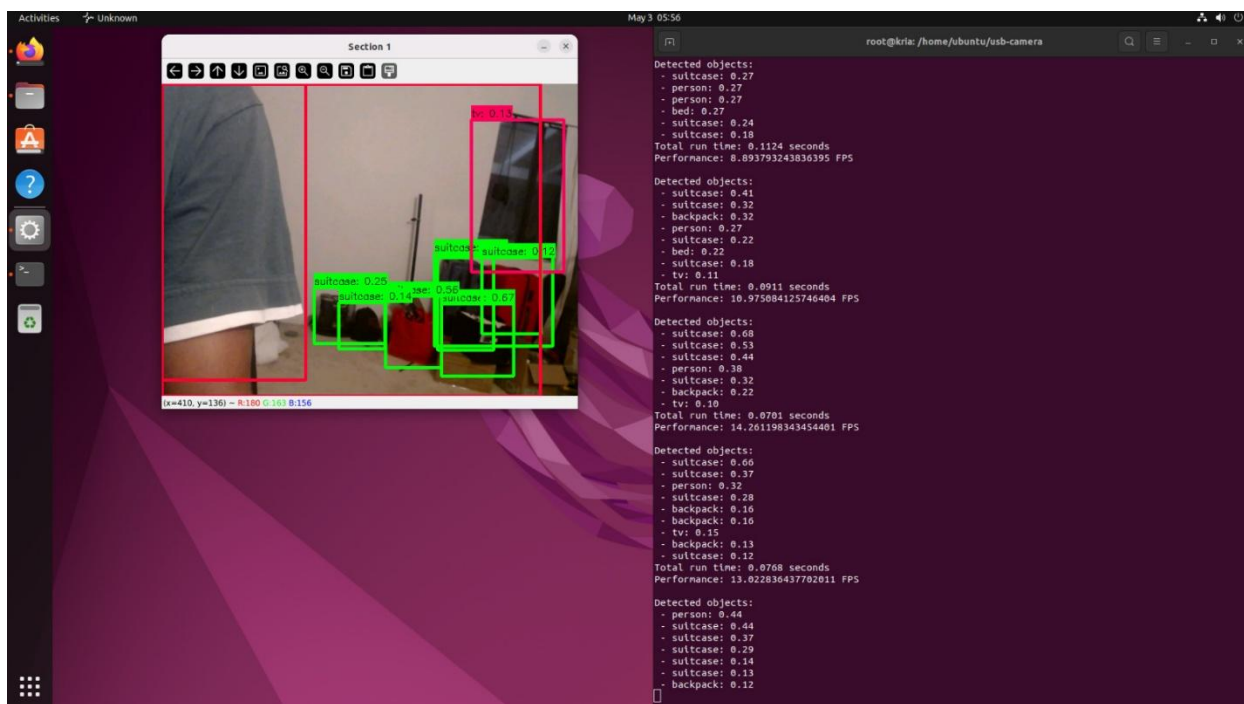


Figure 2: Yolox Output

5.3 Final Outcome

Both processes were effectively put into use, deployed, and examined. In every instance, the DPU ran as expected, running deep learning models with great efficiency. The findings show that the AMD Kria KR260 platform combined with a DPU overlay is a reasonable solution for real-time AI workloads at the edge.

5.4 Future Work

Although the present implementation was successful, future enhancements and extensions might consist of:

- Combining detection and classification into one pipeline
- Testing with additional models such as MobileNet, YOLOv5, or segmentation networks
- Extending the PetaLinux installation to enable real-time video processing
- Designing a custom web-based or GUI interface for remote monitoring
- Using the system in a robotics or surveillance application for long-term field testing

Conclusion

This graduate study shows the feasibility and efficiency of using deep learning models on integrated FPGA-based systems for robotic vision applications. The project obtained real-time inference performance using advanced models as YOLOX for object detection and ResNet-50 for image classification by employing the Xilinx Kria KR260 and its embedded Deep Processing Unit (DPU). Two separate artificial intelligence processes were used to demonstrate the DPU's adaptability and efficiency in speeding up inference operations:

- Workflow A used PetaLinux and the DPU Targeted Reference Design (TRD) to execute the ResNet-50 image classification model.
- Workflow B employed Ubuntu, DPU-PYNQ, and Python to run the YOLOX-Nano model for real-time object detection from a USB webcam stream.

Both processes were created on a common hardware platform built in Vivado and exported as .xsa file. The findings verified that the DPU was operational and properly integrated, providing quick and dependable inference in both embedded and high-level settings. While YOLOX-Nano provided real-time object detection at 10-20 frames per second and a total runtime of 0.02-0.09 seconds, ResNet-50 achieved image classification with an average processing time of 0.2 milliseconds per image.

With support for both production-grade embedded development and flexible prototyping, the project underlines the adaptability of the Kria KR260 as a platform for edge artificial intelligence. The dual-path deployment strategy also supports the reusability of DPU overlays across various use cases, therefore qualifying this approach for a wide spectrum of applications in edge computing, smart vision, and robotics.

This work can be expanded into multi-model inference systems, robotic control integration, or cloud-connected smart edge devices and lays the groundwork for more sophisticated AI deployments on FPGAs.

Finally, our study verifies that hardware-accelerated inference employing platforms such as the Kria KR260 can satisfy the needs of real-world robotic vision applications. It not only strengthens the significance of edge artificial intelligence in robotics but also offers a repeatable and modular solution to the expanding corpus of research and development in embedded deep learning systems.

References

<https://xilinx.github.io/Vitis-AI/3.0/html/index.html>

<https://xilinx.github.io/Vitis-AI/3.0/html/docs/workflow-system-integration.html>

<https://docs.amd.com/r/en-US/ug1092-kr260-starter-kit>

https://xilinx.github.io/kria-apps-docs/kv260/2022.1/build/html/docs/nlp-smartvision/docs/hw_arch_accel_nlp.html

https://xilinx.github.io/kria-apps-docs/creating_applications/2022.1/build/html/docs/kria_vitis_acceleration_flow/adding-dpu-ip.html

<https://github.com/fchollet/deep-learning-models/blob/master/resnet50.py>

<https://github.com/Xilinx/DPU-PYNQ>

<https://pynq.readthedocs.io/en/latest/>

<https://xilinx.github.io/Vitis-AI/3.5/html/index.html>

<https://github.com/amd/Kria-RoboticsAI>