FACIAL LANDMARK DETECTION ON MULTIPLE PLATFORMS USING QUANTIZED MOBILENETV2 FOR EDGE AI APPLICATIONS

1st Do Manh Dung
Computer Engineering Technology
Ho Chi Minh University of Technology and Education
Ho Chi Minh, Thu Duc
21119301@student.hcmute.edu.vn

2 nd Dang Trung Nghia
Computer Engineering Technology
Ho Chi Minh University of Technology and Education
Ho Chi Minh, Thu Duc
21119313@student.hcmute.edu.vn

Abstract: **Real-time applications such as driver monitoring, face detection, and expression analysis have become increasingly critical in modern life. Deploying facial landmark detection systems for user state and emotion assessment requires real-time processing capabilities with stringent power constraints for edge deployment. This work presents a comprehensive evaluation and optimization of facial landmark detection networks across multiple hardware platforms, with particular focus on FPGA implementation for energy-efficient edge computing applications. We demonstrate an end-to-end development workflow that transforms the PyTorch PFLD network into a hardware-accelerated CNN optimized for the Xilinx Kria KV260 platform. Our methodology begins with systematic model compression using Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT) pipelines in Vitis AI, converting all weights and activations to 8-bit fixed-point representation. This quantization strategy reduces the model footprint to 1.65 MB achieving 4× compression with minimal accuracy degradation (less than 1%). Performance evaluation was conducted across five hardware platforms: RTX 4060 GPU, i9 CPU, Xilinx Kria KV260 FPGA, Jetson Nano 2GB, and Raspberry Pi 4B. The FPGA implementation demonstrates superior energy efficiency with 10 FPS/W approximately 2.5× more efficient than the RTX 4060 (3.91 FPS/W), 7× more efficient than the Intel i9 (1.35 FPS/W), and over 11× more efficient than embedded alternatives (Jetson Nano: 0.90 FPS/W, Raspberry Pi: 0.79 FPS/W). The FPGA implementation achieves real-time inference at 30 FPS while maintaining competitive accuracy (91.74% vs 92.42% for other platforms) with a total power consumption of just 3W. This represents a ten-fold energy efficiency improvement compared to CPU implementation, making it ideal for battery-powered edge applications.**

Keywords: Facial Landmark Detection, Mobilenetv2, FPGA, Quantization-Aware Training, Post Training Quantization, Vitis AI, Edge Computing.

I. INTRODUCTION

The Internet of Things has evolved significantly, with an increasing number of devices being connected to networks and equipped with small computing capabilities [1]. Traditionally, these devices using cloud computing services to process data received from their sensors. However, many emerging applications require computation to be performed at the edge - the connection point between the network and the real world. This necessity has given rise to edge computing as a critical paradigm shift. Edge computing focuses on processing data efficiently at the network's edge rather than relying solely on cloud computing [2]. This approach addresses several critical needs: faster data processing speeds compared to using broadband to transmit data to the cloud and waiting for computation results, and the transformation from data consumers to data producers as more people generate new data rather than simply consuming it on their devices. A prime example of this need is autonomous vehicles, which generate massive amounts of data every second that must be processed in real-time, making cloud service dependency impractical [3].

Within this edge computing context, one of the most important usecase for traffic accidents caused by driver drowsiness and fatigue represent a critical challenge for road safety. These incidents account for 3–30% of global accidents and approximately 10 - 20% of severe crashes in Vietnam, resulting in thousands of collisions and hundreds of fatalities annually [4]. The urgency of this problem aligns perfectly with edge computing requirements, as driver monitoring systems demand real-time processing with minimal latency to provide immediate safety interventions. Existing solutions, such as steering wheel sensors, basic surveillance cameras, or algorithms running on CPU/GPU platforms, are often limited by high processing latency, significant power consumption, and costly hardware. These limitations make deployment on resource-constrained embedded systems - typical of edge computing environments - particularly challenging. In response to these challenges, there is an urgent need for a non-intrusive, real-time monitoring approach capable of operating on low-power edge platforms. Facial landmark detection offers a promising solution by accurately identifying key facial features such as eyelids, mouth corners, and head pose. This method enables real-time analysis of blink frequency, mouth opening degree, and head tilt to detect early signs of drowsiness. However, traditional convolutional neural networks (CNNs) typically demand substantial computational resources and memory, posing difficulties for implementation on FPGAs with limited logic capacity and bandwidth [5]- common constraints in edge computing scenarios.

This research proposes a lightweight CNN model specifically tailored for facial landmark detection and optimized for edge deployment. The model incorporates quantization and pruning techniques to significantly reduce model size and computational load, making it suitable for resource-constrained edge devices. The proposed architecture using the parallel processing capabilities of FPGAs to achieve real-time inference with minimal latency and optimal power efficiency. The system is designed for deployment on embedded edge platforms, meeting stringent requirements for cost and energy efficiency while eliminating the need for cloud connectivity during critical safety operations. This approach directly addresses the edge computing paradigm's core principles: processing data locally for immediate response, reducing bandwidth requirements, and ensuring system reliability independent of network connectivity.

Upon completion, the system will provide instantaneous facial landmark detection that serves as the foundation for drowsiness alerts, embodying the edge computing principle of real-time local processing. Additionally, the system enables broader edge applications, including attention monitoring, emotion analysis, AR/VR calibration, and facial biometrics. The proposed solution not only enhances road safety through effective driver state monitoring but also establishes a foundation for deploying real-time computer vision systems on edge devices, optimized for performance, cost-effectiveness, and local processing capabilities. This work contributes to the growing ecosystem of edge computing applications that prioritize immediate response times and autonomous operation, essential characteristics for safety-critical systems in connected vehicle environments.

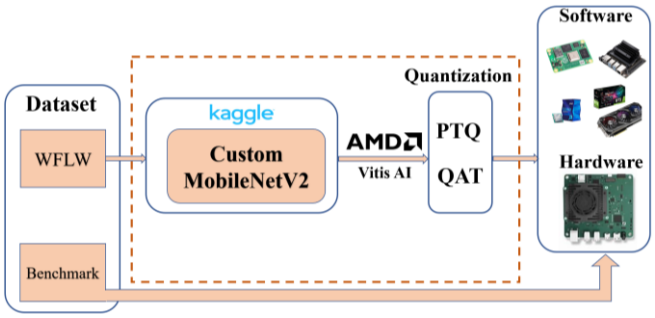

Figure 1. A view of entire system

A. *Related work*

Guo et al.'s pioneering research introduces the PFLD (Practical Facial Landmark Detector) model with an architecture specifically designed for real-world applications, featuring an extremely compact model size of only 2.1 MB while maintaining exceptional accuracy of 92.5%. This solution is deployed on high-end RTX 2080 Ti GPU without applying quantization techniques, enabling impressive processing performance of up to 120 FPS - a speed perfectly suitable for real-time applications requiring ultra-low latency. However, the main limitation of this solution is its high power consumption of approximately 250W, making it only suitable for desktop systems or workstations with stable power supply. The primary target application is facial alignment systems in Augmented Reality (AR) environments, where high accuracy and fast processing speed are mandatory requirements to create smooth and natural user experiences.

Wu et al.'s work focuses on optimization solutions for edge computing devices through the utilization of MobileNet V2 model with a 14 MB size - although larger than PFLD, it is still considered compact for a deep learning model. The breakthrough of this research lies in applying Post-Training Quantization (PTQ) technique with INT8 precision, allowing significant model size reduction and inference acceleration without requiring retraining from scratch. The solution is deployed on Jetson TX2 platform - a System-on-Chip (SoC) specifically designed for AI edge computing, achieving an impressive balance between performance (15 FPS) and energy efficiency (only 7W), with 91.2% accuracy still ensuring application quality. The target application of this solution is driver monitoring systems, where facial landmark detection is used to assess driver alertness, attention level, and emotional expressions, requiring continuous operation in automotive environments with limited power resources.

Yang et al.'s research focuses on solutions for real-time video conferencing applications through deploying the MTCNN (Multi-task Cascaded Convolutional Networks) model with a large size of 68 MB on Intel i7-8700K CPU platform. Despite not using quantization techniques for model optimization, this solution still achieves 89.7% accuracy and 8 FPS processing speed with 80W power consumption. The choice of CPU over GPU or specialized accelerators reflects the goal of creating a highly compatible solution that can run on most personal computers and laptops without requiring special hardware. The primary application of this research is real-time video conferencing systems, where facial landmark detection is used to track facial movements, automatically adjust framing, apply AR filters effects, and improve video call quality through optimizing viewing angles and lighting based on detected facial landmark positions.

B. *Dataset*

The Wider Facial Landmarks in the Wild (WFLW) dataset represents a comprehensive benchmark for facial landmark detection, comprising 10,000 high-variability facial images (7,500 training, 2,500 testing) with 98 manually annotated landmarks covering eyes, eyebrows, nose, lips, jawline, and facial contour, along with detailed attribute annotations for occlusion, pose, illumination, blur, expression, and makeup conditions. This dataset's dense landmark layout and real-world complexity make it particularly valuable for challenging applications such as drowsiness detection, head pose estimation, emotion recognition, AR facial tracking, and driver monitoring systems, where precise measurement of features like eye aspect ratio (EAR), mouth aspect ratio (MAR), head tilt, and subtle facial expressions are critical for accurate performance assessment under practical deployment conditions.

II. METHOD

In the work, we aims to develop a comprehensive Edge-AI pipeline for real-time facial landmark detection optimized MobileNetV2 for FPGA deployment. The primary objective is to create an energy-efficient, hardware-accelerated solution for driver drowsiness detection that operates independently on edge devices without cloud connectivity. The work establishes a complete development workflow transforming the PyTorch PFLD network into an optimized hardware accelerator. The network undergoes systematic optimization through Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT) using Vitis AI, converting weights and activations to 8-bit fixed-point representation to achieve significant model compression with minimal accuracy degradation. The quantized model is translated into the inference engine for real-time inference.

A. *Dataset Augmentation*

Data augmentation represents a critical methodology for enhancing dataset diversity without requiring additional original data acquisition, functioning through diverse modification techniques applied to existing datasets to generate altered versions that improve model generalization capabilities. Image-based data augmentation operates by applying multiple transformation methods to source images while preserving original labels, simultaneously producing enhanced training datasets through various approaches

including: Geometric Transformations, Color Space Augmentations, Kernel Filters. These augmentation techniques collectively enable models to learn from artificially diversified training data, improving robustness to real-world variations in lighting, orientation, and visual conditions while maintaining the integrity of original landmark annotations essential for facial detection applications.

B. Custom MobileNetV2 and backbone architecture

Sandler et al. developed MobileNetV2 as a convolutional neural network architecture specifically optimized for mobile device deployment. The original implementation was designed for various computer vision tasks, such as image classification, and benchmarked on standard datasets like ImageNet. The architecture's key innovations include the implementation of inverted residual blocks and linear bottleneck layers, which enabled the model to achieve cutting-edge performance while maintaining an optimal balance between computational efficiency and accuracy. A typical MobileNetV2 implementation is known for its lightweight design, making it suitable for resource-constrained mobile environments.

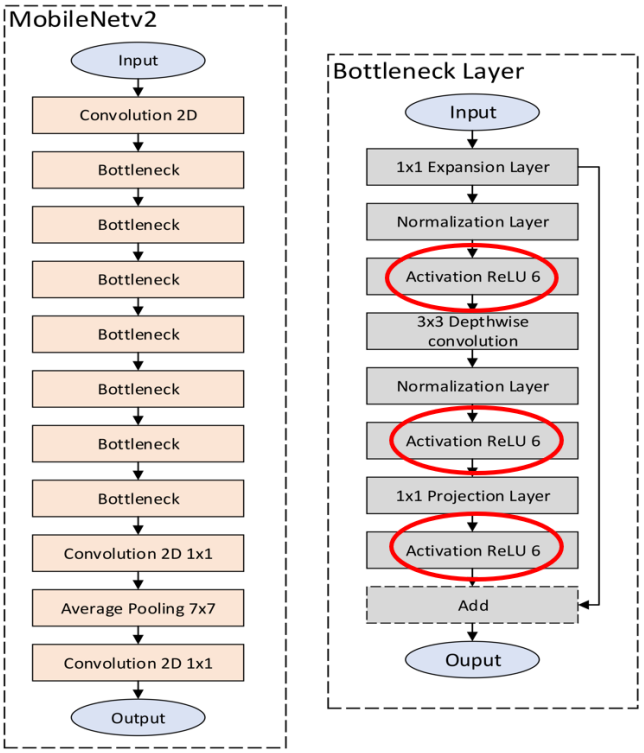

Figure 2. Original MobileNetV2

This Figure 2 illustrates the architecture of the standard MobileNetV2 and its core building block, the Bottleneck Layer. The diagram is presented in two parts: Standard MobileNetV2 Architecture (leftmost) displays the conventional structure, beginning with a 2D convolution layer, followed by a sequence of multiple bottleneck blocks, and concluding with convolution and pooling layers for output generation.

Bottleneck Layer Detail (rightmost) provides an expanded view of the core building block. It showcases the inverted residual structure, which includes a 1x1 expansion layer, a 3x3 depthwise convolution, and a 1x1 projection layer. A key characteristic of the original architecture is the use of ReLU6 as the activation function, as highlighted by the red circles, in the expansion and depthwise convolution layers. The final 1x1 projection layer is followed by a linear activation (no activation function), a key principle of the "linear bottleneck" to preserve low-dimensional information.

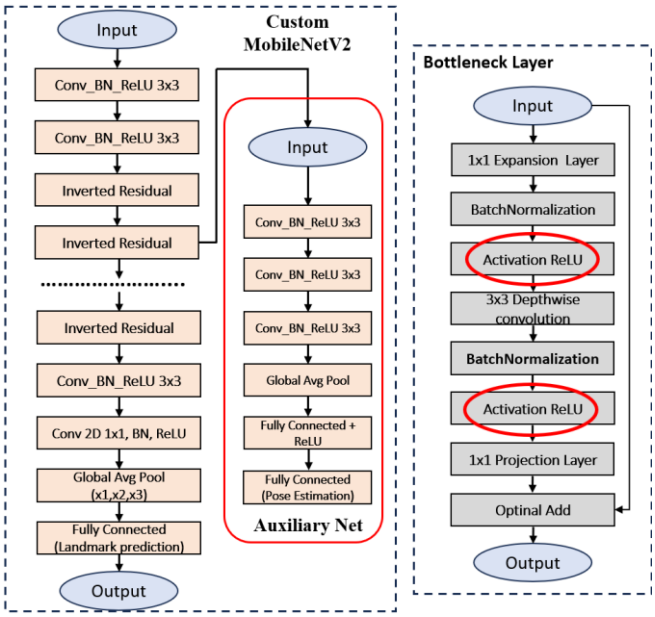

Figure 3. Custom MobileNetV2

This figure illustrates a Custom MobileNetV2 architecture adapted for facial landmark detection and pose estimation. The diagram is presented in two parts:

Custom MobileNetV2 (leftmost) shows the main network branch, which is a modified version of the standard MobileNetV2 backbone. It uses inverted residual blocks and additional Conv_BN_ReLU 3x3 layers. The key modification is the replacement of ReLU6 with standard ReLU activation functions, as highlighted by the red circles in the Bottleneck Layer detail. The final layers include a Global Avg Pool and a Fully Connected layer for landmark prediction.

The Bottleneck Layer (rightmost) shows the detail of the modified bottleneck block used in this custom architecture. It retains the inverted residual structure but replaces the original ReLU6 activations with standard ReLU activations and the Normalization layers with BatchNormalization layers.

C. Quantization.

Quantization has emerged as a leading strategy in addressing the memory and computational demands of deep learning (DL) models. Traditionally, DL networks represent parameters using 32-bit floating-point (FP32) precision. Quantization compresses these representations by substituting them with low-bitwidth values, thereby reducing both memory consumption and computational overhead. When integrated into convolutional neural networks (CNNs), the result is a Quantized Neural Network (QNN) a specialized model family with a growing record of practical successes [9].

A notable benefit of QNNs lies in their ability to enhance inference speed and energy efficiency, owing to the reduced complexity of quantized arithmetic operations. However, this compression introduces a trade-off: loss in numerical precision may lead to degraded accuracy. It is therefore essential to minimize such degradation when designing quantized models.

In the literature, two primary quantization strategies have been established: Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT). These methodologies are illustrated in Figure 4.

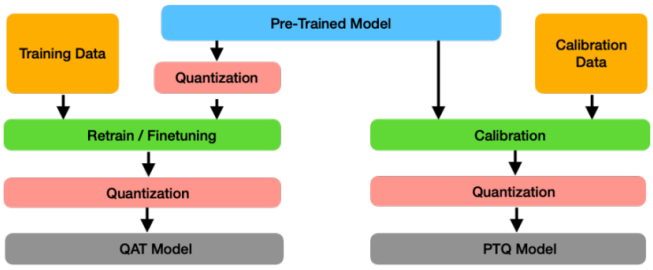

Figure 4. The running mechanism of QAT and PTQ quantization directions

Mathematical Framework underlying both approaches is defined by four fundamental equations: (1) Scale calculation determines the mapping ratio between FP32 and INT8 ranges using their respective minimum and maximum values

$$\text{Scale} = \frac{\max(\text{INT8}) - \min(\text{INT8})}{\max(\text{FP32}) - \min(\text{FP32})} \quad (1)$$

(2) Zero Point computation establishes the offset for asymmetric quantization by rounding the minimum FP32 value divided by the scale factor:

$$\text{Zero Point} = \text{round}\left(\frac{\min(\text{FP32})}{\text{Scale}}\right) \quad (2)$$

(3) Quantization formula converts FP32 values to INT8 by subtracting the zero point and dividing by the scale factor, and

$$\text{Quantized} = \text{round}\left(\frac{\text{FP32 Value} - \text{Zero Point}}{\text{Scale}}\right) \quad (3)$$

(4) Dequantization equation reverses the process by multiplying the quantized value by the scale and adding the zero point.

$$\text{Dequantized} = \text{Scale} \times \text{Quantized Value} + \text{Zero Point} \quad (4)$$

D. Quantization in Vitis AI

The Vitis AI framework utilizes precision reduction methodologies to transform deep learning models from high-precision floating-point data types (typically FP32) to lower-precision fixed-point numerical formats. This transformation process is designed to enable enhanced inference performance on Deep-learning Processing Units (DPUs), which are engineered specifically for fixed-point arithmetic operations.

The Vitis AI development environment supports precision conversion procedures for deep neural network models derived from various machine learning frameworks. These quantized models can be deployed on various AMD/Xilinx hardware platforms including Versal VEK280, Alveo V70 data center cards, and embedded platforms [20].

TABLE I. VITIS AI SUPPORTS QUANTIZATION

Framework	Post-Training Quantization	Quantization-Aware Training	Supported Data Types
PyTorch	✓	✓	INT8
TensorFlow 1.x	✓	✓	INT8
TensorFlow 2.x	✓	✓	INT8

E. Pytorch Framework

PyTorch represents a machine learning framework built upon the Torch library, designed for applications including computer vision and natural language processing tasks. Initially created by Meta AI and currently maintained under the Linux Foundation's governance, it stands as one of the leading deep learning platforms alongside frameworks like TensorFlow, providing open-source software distributed under the modified BSD license. While the Python interface receives primary development attention and offers enhanced functionality, PyTorch also supports C++ programming interfaces [41].

PyTorch serves as the cornerstone framework for this research, leveraging its two fundamental capabilities: tensor computing with robust GPU acceleration similar to NumPy, and deep neural networks constructed on a tape-based automatic differentiation system. Selected for its dynamic computational graph support, flexibility, and extensive community ecosystem that aligns with iterative edge AI development requirements, PyTorch facilitates the complete research workflow from model development through hardware deployment.

The framework enables efficient implementation and training of the MobileNetV2 backbone and auxiliary networks on 300W, AFLW, and WFLW datasets while providing seamless GPU acceleration for rapid experimentation with different architectures and hyperparameters. PyTorch's integration with the Brevitas library supports both Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT) pipelines, producing highly compressed models suitable for fixed-point hardware deployment. The framework's compatibility with Xilinx's Vitis AI toolchain streamlines the transition from software training environments to hardware-accelerated inference, enabling direct conversion of trained and quantized models into .xmodel format for deployment on the Kria KV260 FPGA platform.

F. Overview of Hardware Platforms for Comparison

Comprehensive hardware platform comparison establishes a diverse evaluation framework encompassing five representative computing architectures, each selected for distinct deployment scenarios and performance characteristics that collectively provide insights into AI inference trade-offs across varying computational environments.

The NVIDIA RTX 4060 GPU, constructed on a 5nm process node, represents the high-performance baseline with massive parallelism through CUDA cores optimized for deep learning, targeting data center inference, cloud-edge

hybrid AI systems, and video analytics applications requiring ultra-low latency and exceptional throughput.

The Intel Core i9-13900K CPU from the Raptor Lake generation features a hybrid architecture combining performance and efficiency cores, serving as a general-purpose inference engine for desktop/workstation deployments and CPU-based optimization testing where GPU acceleration is unavailable, offering balanced performance without dedicated accelerators.

The NVIDIA Jetson Nano 2GB provides an entry-level edge AI platform with quad-core ARM Cortex-A57 CPU and 128-core Maxwell GPU, supporting lightweight frameworks like TensorRT and ONNX Runtime within a limited power envelope, making it ideal for prototyping, hobbyist projects, and simple computer vision tasks requiring compact, power-efficient design.

The Raspberry Pi 4 Model B represents resource-constrained systems as a widely adopted single-board computer with extensive community support, included to assess CNN inference feasibility without hardware acceleration for educational purposes, proof-of-concept deployments, and simple embedded vision applications. This cross-platform selection enables balanced assessment of performance, energy efficiency, and deployment feasibility, illustrating trade-offs from high-end GPUs to resource-limited SBCs while highlighting the advantages of FPGA-based solutions in real-world edge AI scenarios through comprehensive comparison

III. IMPLEMENTATION

An incoming Grayscale frame is first processed by a face detector, which yields a tight bounding box around the face. The detected patch is converted to grayscale to reduce input dimensionality and normalized to the [0, 1] range. Resizing to 112 × 112 pixels standardizes spatial resolution across all samples.

As showns in Figure 3, the input graycale frame undergoes face detection, grayscale conversion, normalization, and resizing to 112×112. During training, online augmentation (geometric transforms, photometric adjustments, and filtering) is applied. The augmented image is passed through the Backbone module (MobileNetV2), then the Neck for multi-scale feature fusion, and the Head to regress 98 landmark coordinates. A training-only Auxiliary Pose branch predicts Euler angles for geometric regularization; this branch is removed during inference to ensure low latency.

During training, each crop undergoes a sequence of augmentations geometric (rotation, translation, shear, flip), photometric (brightness, contrast, saturation, hue), and kernel-based filtering (Gaussian blur, sharpening, edge enhancement) to simulate real-world variations such as head tilt, illumination shifts, and motion blur. Augmented images are fed into the core model, which consists of:

- A Backbone (MobileNetV2-inspired) extracting hierarchical features.
- A Neck module fusing multi-scale descriptors into a unified representation.
- A Head regressing 98 landmark coordinate pairs via a lightweight fully connected layer.
- An Auxiliary Pose Branch (training-only) predicting yaw, pitch, and roll for geometric regularization.
- At inference, the auxiliary branch is pruned to preserve low latency, allowing real-time landmark detection on embedded hardware.

A. Pre-processing

Face detection employs a cascade classifier or small CNN to produce a bounding box. The face region is cropped with a small margin to include peripheral features (outline of jaw, hairline). Grayscale conversion followed by histogram equalization or contrast normalization standardizes appearance across lighting conditions. Finally, resizing to 112 × 112 uses bilinear interpolation, and pixel values are scaled to [0, 1]. Landmark annotations, originally in text format, are loaded as NumPy arrays of shape (98, 2).

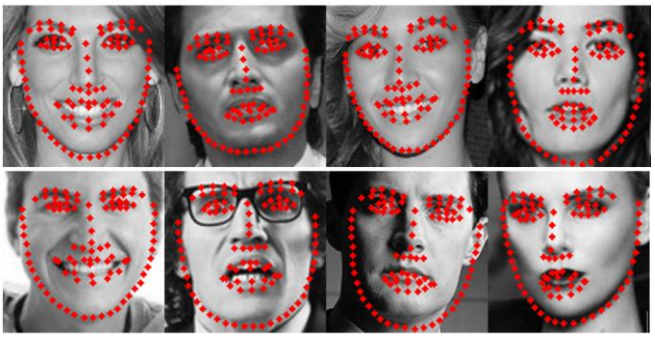

Figure 5. 98 Facial Landmarks Images

B. Model Architecture

The landmark detection network employs a Backbone–Neck–Head design, augmented by a training-only Auxiliary Pose branch. This modular structure balances accuracy, efficiency, and ease of hardware mapping.

Backbone Network (MobileNetV2 Blocks) - Recognizing that the backbone subnetwork is the only module active during inference, it becomes the primary bottleneck in both runtime and model footprint. To optimize for speed and memory efficiency, we adopt MobileNet [8] as our base architecture. MobileNet's architectural innovations including depthwise separable convolutions, linear bottlenecks, and inverted residual connections have proven effective in reducing complexity without compromising performance [44, 45]. Accordingly, we replace standard convolution operations with MobileNet blocks to significantly lower computational demands and accelerate the network. By doing so, the computational load of our backbone network is significantly reduced and the speed is thus accelerated.

The backbone extracts hierarchical features via depthwise-separable convolutions and inverted residual bottlenecks, following MobileNetV2 principles. Early downsampling broadens the receptive field; subsequent blocks learn compact embeddings.

Auxiliary Pose Estimation Branch - Unlike prior methods that directly regress a 3D – 2D projection matrix [28], infer part - wise dendritic structures [46], or exploit boundary lines [45], PFLD ' s auxiliary subnet independently estimates the three Euler angles (yaw, pitch, roll) to regularize landmark learning under large pose variations

TABLE II. THE BACKBONE NETWORK CONFIGURATION

Input	Operator	t	c	n	s
$112^2 \times 1$	Conv3 × 3	-	64	1	2
$56^2 \times 64$	Conv 3 × 3	-	64	1	1
$56^2 \times 64$	Bottleneck	2	64	1	2
$28^2 \times 64$	Bottleneck	2	64	4	1
$28^2 \times 64$	Bottleneck	2	128	1	2
$14^2 \times 128$	Bottleneck	4	128	5	1
$14^2 \times 128$	Bottleneck	2	128	1	1
$14^2 \times 128$	Bottleneck	2	16	1	1
(S1) $14^2 \times 16$	Conv3 × 3	-	32	1	2
(S2) $7^2 \times 32$	Conv1 × 1	-	64	1	1
(S3) $1^2 \times 128$	Global AvgPool	-	-	-	-
S1,S2,S3	Fully Connected	-	196	1	-

To prevent unstable angle estimation from noisy landmark outputs especially early in training we define a fixed, average frontal‐face template with 11 reference landmarks and compute each sample's rotation matrix against this template; the corresponding Euler angles then serve as auxiliary supervision without requiring any extra pose annotations.

TABLE III. THE AUXILIARY BRANCH CONFIGURATION

Input	Operator	c	s
$28^2 \times 64$	Conv3 × 3	128	2
$14^2 \times 128$	Conv3 × 3	128	1
$14^2 \times 128$	Conv3 × 3	32	2
$7^2 \times 32$	Global AvgPool	-	-
$1^2 \times 32$	Fully Connected	32	-
$1^2 \times 32$	Fully Connected	3	-

C. Algorithms Explaining the Complete Pipeline

The proposed approach has been comprehensively outlined through two distinct algorithms presented below. Initially, images underwent preprocessing procedures and were utilized for training across the complete dataset. Subsequently, the model developed in the initial phase was employed to identify facial landmarks with suitable precision levels.

The Algorithm 1 describes a simple, interactive pipeline for running a facial‐landmark (or face‐classification) model on a live camera feed, with two main modes one "per‐face" step and one continuous real‐time display.

Algorithm 1: Deployment of Facial Landmark Detector

1: **If** the face is detected **then**
2: Get prediction from the face classifier model
3: Show predictions and save resultant image
4: **Else**
5: Show no output
6: **End if**
7: **If** the choice is classification in real-time **then**
8: Load real-time feed from MTCNN
9: Convert to grayscale image
10: Read the feed frame by frame
11: Perform inference using the model
12: Show output in real-time feed
13: **Else**
14: Show normal feed
15: **End if**
16: End stream when 'Esc' is pressed

After following the completion of neural network training for facial landmark identification, this trained network can subsequently be deployed across any images containing human faces. Since the neural network requires input data in the form of a Tensor with specific dimensional requirements, facial detection necessitates initial preprocessing operations.

1. Identify all facial regions within an image utilizing a face detection algorithm (in this implementation, we will employ a Haar Cascade detector).

2. Transform the detected facial images through preprocessing steps including grayscale conversion and reshaping into Tensor format with dimensions matching the network's input specifications.

3. Deploy the trained model to identify facial landmarks within the processed image data.

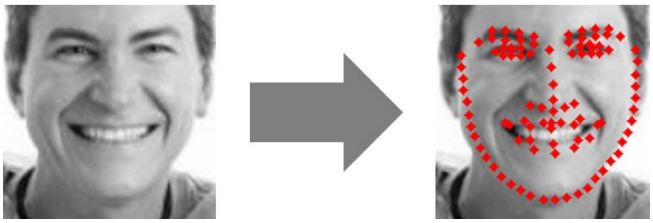

Figure 6. Inference facial landmark detection Pipeline

D. Post-Training Quantization (PTQ)

Dynamic range quantization represents a post-training quantization methodology that converts model parameters to 8-bit integer representations while maintaining activation values in floating-point precision. This quantization approach involves transforming floating-point numerical values into a finite set of integer representations based on the minimum and maximum parameter values within each network layer.

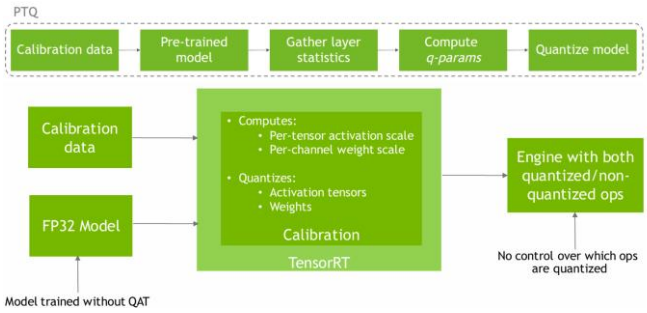

Figure 7. Figure PTQ workflow

Modern neural networks often interleave convolution (or linear) layers with batch normalization and activation functions. To minimize rounding error and intermediate data movement, PTQ fuses sequences such as:

Conv2d → BatchNorm → ReLU or Linear → ReLU

into a single fused operation. This eliminates redundant float-to-int conversions between layers and shrinks the computational graph. After fusion, the model's behavior remains identical, but the kernels are now tailored for quantized execution.

We perform Post-Training Quantization (PTQ) using TensorRT. First, we feed a small set of calibration data and the original FP32 model into the TensorRT calibration workflow. During this step, TensorRT gathers layer-wise statistics computing per-tensor activation scales and per-channel weight scales. It then quantizes both weights and activations to INT8 using those scale (and zero-point) parameters. Finally, TensorRT builds an inference engine that mixes quantized and non-quantized operations to execute the model in INT8 wherever possible. Note that, because this is a one-shot conversion, We don't get control over exactly which ops get quantized; TensorRT decides this automatically to maximize performance.

E. Quantization-Aware Training (QAT)

In Quantization-Aware Training, we insert two helper nodes around each layer we want to quantize. First, a Quantize (Q) node right after the FP32 input uses the learned scale and zero-point to round and offset each value into the INT8 range [–128, 127]. The marked convolutional or fully-connected layer then runs directly on these INT8 weights and activations because we fine-tuned the network with these Q→DQ pairs during training, it learns to compensate for any rounding errors and retains almost the same accuracy.

Finally, a Dequantize (DQ) node converts the INT8 output back to FP32 for any downstream operations that require full precision. This pipeline gives us precise control over which operations get quantized, and which remain in 32-bit, delivering an inference engine that is efficient edge devices.

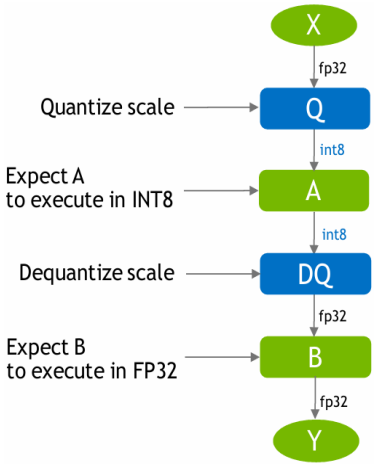

Figure 9. QAT workflow

IV. Result & Discussion

A. Training Model

We trained entirely in PyTorch on a CUDA - enabled GPU for 100 epochs, using a batch size of 64 and the Adam optimizer (learning rate 1e-4 with weight decay 1e-5). The plot on the right tracks training (blue) and validation (orange) accuracy over those 100 epochs. Notice how both curves climb rapidly in the first 20 epochs, then gradually converge around 92 % demonstrating strong generalization and minimal overfitting. This FP32 model forms the foundation for our subsequent quantization experiments.

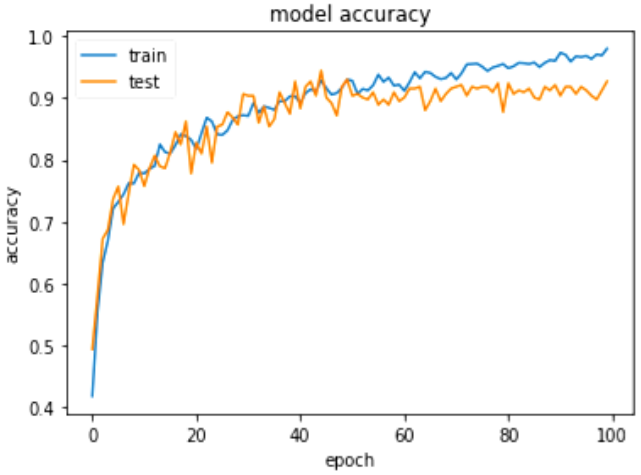

Figure 8. Traning result

This chart above displays the model's training and testing performance. In the accuracy chart, both the training and test accuracy show a clear upward trend, stabilizing towards the later stages of training. Although the validation accuracy is slightly lower, it remains relatively stable. The test accuracy exhibits some fluctu- ation in the early and middle stages, suggesting that there might have been some disturbances during the training process, while the training accuracy remains more consistent. Overall, accuracy steadily improves as training progresses, and around 60 epochs, the model shows signs of convergence, especially with the gap between training and test accuracy narrowing, indicating good generalization ability.

B. Quantization

After performing quantization using Vitis AI, we utilized the Netron tool to inspect the model weights and observe the changes. This allowed to visually compare the original FP32 (32-bit floating point) format with the quantized INT8 (8-bit integer) format.

To assess how post - training quantization (PTQ), and quantization - aware training (QAT) affect our baseline CNN (6.59 MB, 92.42 % FP32 accuracy), we applied each quantization strategy and recorded four metrics model size, inference accuracy, end-to-end pipeline time, and robustness relative to the unmodified network. The comparative results are presented in Table IV.

TABLE IV. THE QUANTIZED METHOD RESULTS

Quantization Method	Accuracy (%)	Model Size (MB)	Quantization Time	Compression Ratio
Original (FP32)	92.42	6.59	-	1x
QAT (INT8)	91.74	1.65	Approximately 3 hours (fine-tune)	4x
PTQ (INT8)	90.94	1.65	Approximately 10 minutes (calibration)	4x

The comparison clearly shows that while the full-precision Float model retains the highest accuracy (92.42 %) and "Very High" robustness, it comes at the cost of a large 6.59 MB footprint and requires no quantization time. Quantization-Aware Training (QAT) reduces model size by 75 % (to 1.65 MB) with only a modest accuracy drop to 91.74 %, and achieves "High" robustness by exposing the network to quantization effects during training though it demands a substantial fine-tuning investment of roughly three hours. Post-Training Quantization (PTQ) matches QAT's compact 1.65 MB size and can be completed in around ten minutes, making it ideal for rapid deployment; however, its accuracy falls further to 90.94 % and its "Medium" robustness means it is more sensitive to distribution shifts.

In practice, QAT is the preferred choice when both accuracy and resilience to varied inputs are critical, whereas PTQ offers a fast, resource-light solution when development speed outweighs the smallest performance penalty.Next, run each batch from a calibration dataset - a smaller but diverse and representative subset of your training data through the model so that the observers update their recorded x_min and x_max for every layer. Once the observers have collected each tensor's minimum (x_min) and maximum (x_min) values, we move on to computing the quantization parameters.

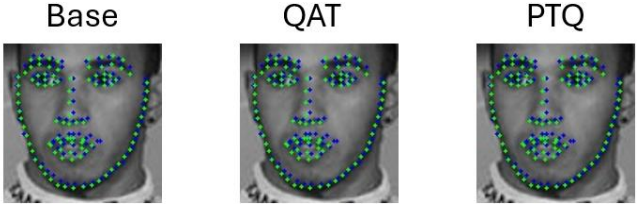

Figure 10. Visualization images after quantization by QAT and PTQ with green landmarks is groud truth and blue landmarks is predicted

In Figure 10, we compare our baseline FP32 model against the two INT8 quantized variants. The original FP32 checkpoint achieves 92.42 % accuracy and occupies 6.59 MB on disk. After Quantization-Aware Training (QAT), the model shrinks by 4× to just 1.65 MB, and accuracy only dips slightly to 91.74 % at the cost of about three hours of fine-tuning. With Post-Training Quantization (PTQ) we achieve the same 4× compression and 1.65 MB footprint but in under ten minutes of calibration; here accuracy drops a bit more to 90.94 %. This table clearly illustrates the trade-off: QAT gives you the highest retained accuracy, while PTQ delivers rapid turnaround for deployment.

C. Target Platforms Deployment

The primary objective of this framework is to detect and recognize objects positioned within the camera's field of view. The PFLD dataset, as previously described, was employed for model training and calibration procedures outlined in the preceding chapter. This dataset comprises approximately 80,000 pre-annotated images with comprehensive information ready for training purposes. Additionally, this dataset has been extensively utilized across numerous research initiatives. The training phase utilized 75,000 images, while the remaining 2,500 images were reserved for calibration activities performed throughout the quantization process.

For this experimental investigation, a single deep neural network architecture, specifically MobileNetV2, was implemented. This choice was made to incorporate diverse performance characteristics and architectural variations within the research framework. Each configuration underwent evaluation using all models in a standardized testing environment through three 60-second test cycles, extending to 20-minute sessions when irregularities were detected.

The experimental methodology incorporated three distinct hardware acceleration platforms: NVIDIA's Jetson Nano single-board GPU system, a Raspberry Pi Embedded Computer, AMD's Kria SOM KV260 FPGA-based platform, an Intel Core i9 CPU, and an RTX-4060 GPU. These accelerators functioned both as edge computing solutions and hosted the required operating systems for each respective case. The rationale for utilizing them as independent units stems from the necessity to accurately assess power consumption and thermal characteristics without external system interference.

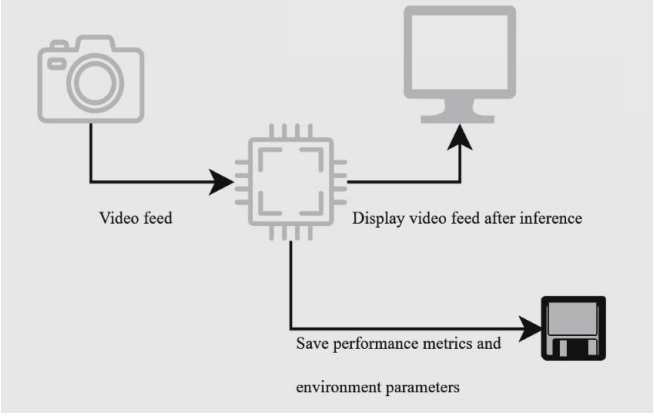

Figure 11. System workflow

A detailed breakdown of the inference methodology for Jetson Nano, Raspberry Pi, KV260 platforms is provided in Algorithms 2 and 3. For collecting experimental metrics including thermal readings, power consumption, and memory utilization, platform-specific monitoring utilities were utilized. Both the Jetson Nano and Kria KV260 incorporate dedicated integrated circuits designed for measuring these parameters and offer application programming interfaces that enable users to retrieve data through built-in monitoring applications.

ALGORITHM 2: Kria KV260 inference

```
Load Xmodel.
Get camera input as batch of images.
while True do
    Perform inference using the model.
    Calculate FPS.
    if model == MTCNN then
        Identify object and draw a box.
    else
        if model == MobileNetV2 then
            Identify object.
        end if
    end if
end while
```

ALGORITHM 3: Jetson Nano inference

```
model ← trained_model
device ← cuda
Load trained model to device.
Initialize the pipeline.
Get camera input as source.
Connect to the device and start the pipeline.
while True do
    Convert the Grayscale frame to a PyTorch tensor.
    Load the tensor to the GPU.
    Perform inference using the model.
    Calculate FPS.
    if model == MTCNN then
        Identify face and draw a box.
    else
        if model == MobileNetV2 then
            Facial landmark detection
        end if
    end if
end while
```

Based on our comprehensive evaluation, the FPGA Kria KV260 emerges as the optimal Edge AI platform, delivering exceptional energy efficiency of 10 FPS/W 2.5× more efficient than GPU and 7× more efficient than CPU implementations. The system achieves real-time performance at 30 FPS while consuming only 3W, maintaining competitive accuracy (91.74%) with a compact footprint ideal for safety-critical automotive applications where driver drowsiness detection requires continuous, low-power operation as shown in Table V.

TABLE V. PLATFORM EXPERIMENT RESULTS

Platform		FPS	Power (W)	FPS/W	Accuracy (%)
Software	RTX 4060 (1.83 – 2.46 GHz)	430	110	3.91	92.42
	CPU i9 (3.20 – 5.50 GHz)	108	80	1.35	92.42
	Jetson Nano (0.475 GHz)	9	10	0.90	92.42
	Raspberry (0.600 GHz)	6	7	0.79	92.42
Hardware	Kria KV260 (0.300 – 0.400 GHz)	30	3	10	91.74

V. CONCLUSION

This research has successfully demonstrated that lightweight PFLD facial landmark detection can be effectively deployed across diverse hardware platforms, from cloud-class GPUs and desktop CPUs to embedded ARM boards and ultra-efficient FPGAs. Through systematic evaluation and optimization, we have established comprehensive performance benchmarks that guide platform selection for edge AI applications. The comparative analysis reveals significant variations in energy efficiency, computational capability, and deployment suitability across different hardware architectures as shown in Table 9.

Based Based on our comprehensive evaluation, the FPGA Kria KV260 emerges as the optimal Edge AI platform, delivering exceptional energy efficiency of 10 FPS/W 2.5× more efficient than GPU and 7× more efficient than CPU implementations. The system achieves real-time performance at 30 FPS while consuming only 3W, maintaining competitive accuracy (91.74%) with a compact footprint ideal for safety-critical automotive applications where driver drowsiness detection requires continuous, low-power operation. When examining the relationship between transistor count and energy performance, the KV260 demonstrates superior architectural optimization:

- RTX 4060 (18.9 billion transistors, 5nm): Despite having the most advanced technology and highest transistor density, achieves only 3.91 FPS/W due to generalized GPU architecture.
- CPU I9 (15.4 billion transistors, 10nm): With substantial transistor count but low energy efficiency (1.35 FPS/W) due to general-purpose design.
- KV260 (1.15 billion transistors, 16nm): Achieves highest energy efficiency (10 FPS/W) with significantly fewer transistors thanks to specialized FPGA architecture optimization.
- Jetson Nano (2 billion transistors, 20nm): Moderate performance (0.9 FPS/W) suitable for prototyping.
- Raspberry Pi (1.5 billion transistors, 28nm): Lowest performance (0.79 FPS/W) but suitable for educational purposes.

This demonstrates that energy efficiency depends not solely on transistor count or technology node, but more critically on architectural optimization for specific tasks. The FPGA KV260 with its reconfigurable architecture maximizes utilization of each transistor for drowsiness detection, achieving 8.7× higher performance per transistor compared to the RTX 4060.

The demonstrated FPGA implementation addresses the critical need for energy-efficient driver drowsiness detection, where fatigue-related accidents account for 3-30% of global incidents. Beyond automotive safety, this work establishes a versatile foundation for edge-vision applications including human-computer interaction, telemedicine monitoring, AR/VR calibration, and biometric authentication systems. Moreover, using a MobileNetV2 backbone combined with a PTQ + QAT strategy yields an ultra-light CNN without sacrificing accuracy:

- MobileNetV2 leverages depthwise-separable blocks to drastically cut parameters and computation versus traditional CNNs.
- QAT tightly optimizes to 8-bit, producing a final model just 6.59 MB in size with almost no accuracy loss (92.42% → 91.74%).

In summary, the FPGA KV260 SOM paired with MobileN.etV2 (PTQ + QAT) offers an exceptional balance of size, performance, accuracy, and power efficiency, perfectly meeting the demands of lightweight, always-on, high-performance Edge AI systems. This architecture sets a precedent for edge AI deployment with optimal performance per transistor ratios, paving the way for a new generation of energy-efficient AI solutions.

ACKNOWLEDGMENT

We would like to express our deepest gratitude to Ph.D. Phạm Văn Khoa, our esteemed supervisor and mentor, for his invaluable guidance and unwavering support throughout this research project. His profound expertise in hardware-software co-design and FPGA-based AI acceleration has been instrumental in steering our research toward meaningful contributions and empowering us to navigate complex technical challenges with confidence.

This work focused on developing an energy-efficient Edge AI pipeline for real-time facial landmark detection with comprehensive evaluation across multiple hardware platforms including GPU, CPU, FPGA, and embedded systems. Under Ph.D. Khoa's expert supervision, we successfully implemented systematic model compression through Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT), achieving INT8 precision with minimal accuracy degradation. The successful deployment on the Kria KV260 platform, achieving high FPS with superior energy efficiency, directly reflects his exceptional mentorship.

Ph.D. Khoa's expertise in quantization strategies and hardware-aware optimization proved invaluable in transforming our theoretical framework into a practical, high-performance prototype. His constructive feedback on performance evaluation and comparative analysis significantly enhanced the rigor of our research contributions. We are deeply grateful for his continuous encouragement and technical guidance, which played a pivotal role in achieving our thesis objectives. His constructive feedback on multi-platform performance evaluation and comparative analysis significantly enhanced the rigor of our research contributions.

REFERENCES

[1] [1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog Computing and Its Role in the Internet of Things,” Proceedings of the MCC Workshop on Mobile Cloud Computing, 2012.

[2] [2] M. Satyanarayanan, “The Emergence of Edge Computing,” IEEE Computer, vol. 50, no. 1, pp. 30–39,2017.

[3] [3] F. Bonomi, “Cloudlets: Bringing the Cloud to the Mobile User,” Proceedings of the 3rd ACM Workshop on Mobile Cloud Computing and Services, 2012.

[4] [4] National Highway Traffic Safety Administration (NHTSA), “Traffic Safety Facts: Drowsy Driving,” U.S. Department of Transportation, Report No. DOT HS 812 333, 2017.

[5] [5] R. Krishnamoorthi, “Quantizing Deep Convolutional Networks for Efficient Inference: A White Paper,” Facebook AI Research, 2018.

[6] [6]: W. Wu, C. Qian, S. Yang, Q. Wang, Y. Cai, and Q. Zhou, “Look at boundary: A boundary-aware

[7] [7]: [29]: A. Tragoudaras, P. Stoikos, K. Fanaras, A. Tziouvaras, G. Floros, G. Dimitriou, K. Kolomvatsos, and G. Stamoulis, “Design Space Exploration of a Sparse MobileNetV2 Using High-Level Synthesis and Sparse Matrix Techniques on FPGAs,” Sensors, vol. 22, no. 12, p. 4318, Jun. 2022. https://doi.org/10.3390/s22124318

[8] [8]: M. Sandle, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and lin ear bottlenecks. CoRR, abs/1801.04381, 2018.

[9] [9]: Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. J. Mach. Learn. Res. 2017, 18, 6869–6898.

[10] [10]: Cai, Y., Yao, Z., Dong, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. (2020). Zeroq: A novel zero shot quantization framework.

[11] [11]: O. Nagel, M. Eichner, and P. Schiele, “Data‐Free Quantization through Weight Equalization and Bias Correction,” Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops, 2019.

[12] [12]: Y. Choukroun, M. Katz, and A. Weller, “Outlier Channel Splitting for Post‐Training Quantization of Deep Neural Networks,” International Conference on Learning Representations (ICLR), 2019.

[13] [13]: Maarten Grootendorst, “A Visual Guide to Quantization”, A Visual Guide to Quantization - by Maarten Grootendorst, 2024.

[14] [14]: Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv preprint arXiv:1806.08342, 2018.

[15] [15]: Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, Dmitry Kalenichenko, “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference”, 2017.

[16] [16]: Raghuraman Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A white paper”, 2018

[17] [17]: Moran Shkolnik, Brian Chmiel, Ron Banner, Gil Shomron, YuryNahshan, AlexBronstein, UriWeiser, “Robust Quantization: One Model to Rule Them All”, Department of Electrical Engineering-Technion, Haifa, Israel (2019)

[18] [18]: Yefei He, Jing Liu, Weijia Wu, Hong Zhou1, Bohan Zhuang, “EfficientDM: Efficient Quantization-Aware Fine-Tuning of Low-Bit Diffusion Models”, Zhejiang University, China ZIP Lab, Monash University, Australia, 2024.

[19] [19]: Y. Choukroun, M. Katz, and A. Weller, “Outlier Channel Splitting for Post‐Training Quantization of Deep Neural Networks,” International Conference on Learning Representations (ICLR), 2019.

[20] [20]: AMD, Vitis AI User guide, UG1414 (v3.5) September 28, 202, https://docs.amd.com/r/en-US/ug1414-vitis-ai

[21] [21] Khan, W. Z., Ahmed, E., Hakak, S., Yaqoob, I., & Ahmed, A. (2019). Edge computing: A survey. Future Generation Computer Systems, 97, 219-235.

[22] [22]: D. M. L. Barbato and O. Kinouchi, “Optimal pruning in neural networks”, Physical Review E, vol. 62, no. 6, pp. 8387–8394, Dec. 2000, Publisher: American Physical Society. DOI: 10.1103/PhysRevE.62.8387. (visited on 05/01/2024).

[23] [23]: Vitis ai optimizer, github.io documentation, [visited:02.04.2024]. [Online]. Avail- able: https://xilinx.github.io/Vitis- AI/3.0/html/docs/ workflow-model-development.html?cv=1.

[24] [24]: “Vitis ai optimizer • vitis ai user guide (ug1414) • reader • amd technical infor- mation portal”. [visited:22.04.2024]. (), [Online]. Available: https://docs. amd.com/r/en-US/ug1414-vitis-ai/Vitis-AI-Optimizer.

[25] [25]: R. Gray and D. Neuhoff, “Quantization”, IEEE Transactions on Information The- ory, vol. 44, no. 6, pp. 2325–2383, 1998. DOI: 10.1109/18.720541.

[26] [26]: M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, “A white paper on neural network quantization”, arXiv preprint arXiv:2106.08295, 2021.

[27] [27] Satyanarayanan, M. (2017). The emergence of edge computing. Computer, 50(1), 30-39.

[28] [28]: A. Jourabloo and X. Liu. Pose-invariant 3d face align ment. In ICCV, 2015.

[29] [29] Yu, W., Liang, F., He, X., Hatcher, W. G., Lu, C., Lin, J., & Yang, X. (2018). A survey on the edge computing for the Internet of Things. IEEE Access, 6, 6900-6919.

[30] [30] Porambage, P., Okwuibe, J., Liyanage, M., Ylianttila, M., & Taleb, T. (2018). Survey on multi-access edge computing for internet of things realization. IEEE Communications Surveys & Tutorials, 20(4), 2961-2991.

[31] [31] Roman, R., Lopez, J., & Mambo, M. (2018). Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. Future Generation Computer Systems, 78, 680-698.

[32] [32] Khan, W. Z., Ahmed, E., Hakak, S., Yaqoob, I., & Ahmed, A. (2019). Edge computing: A survey. Future Generation Computer Systems, 97, 219-235.

[33] [33]: Mao, Y., You, C., Zhang, J., Huang, K., & Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. IEEE Communications Surveys & Tutorials, 19(4), 2322-2358.

[34] [34]: Kuon, I., Tessier, R., & Rose, J. (2008). FPGA architecture: Survey and challenges. Foundations and Trends® in Electronic Design Automation, 2(2), 135-253.

[35] [35] Nurvitadhi, E., Venkatesh, G., Sim, J., Marr, D., Huang, R., Ong Gee Hock, J., ... & Deborah, M. (2017). Can FPGAs beat GPUs in accelerating next-generation deep learning? In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (pp. 5-14).

[36] [36] Betz, V., Rose, J., & Marquardt, A. (1999). Architecture and CAD for deep-submicron FPGAs. Springer Science & Business Media

[37] [37] Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., & Cong, J. (2015). Optimizing FPGA-based accelerator design for deep convolutional neural networks. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (pp. 161-170).

[38] [38] Lemieux, G. G., & Lewis, D. (2004). Design of interconnection networks for programmable logic. Springer Science & Business Media.

[39] [39] Farabet, C., Martini, B., Akselrod, P., Talay, S., LeCun, Y., & Culurciello, E. (2010). Hardware accelerated convolutional neural networks for synthetic vision systems. In Proceedings of 2010 IEEE International Symposium on Circuits and Systems (pp. 257-260).

[40] [40] U. Farooq et al., Tree-Based Heterogeneous FPGA Architectures , FPGA Architectures: An Overview, 2012

[41] [41] Ketkar, N., & Moolayil, J. (2021). Deep learning with Python: Learn best practices of deep learning models with PyTorch. Apress.

[42] [42]: Guo, X., Li, S., Yu, J., Zhang, J., Ma, J., Ma, L., Liu, W., & Ling, H. (2019). PFLD: A Practical Facial Landmark Detector [Preprint]. arXiv:1902.10859 v2

[43] [43]: Xiang Wang, Kai Wang, Shiguo Lian, “A Survey on Face Data Augmentation”, CloudMinds Technologies Inc, 2019.

[44] [44]: A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural net works for mobile vision applications. CoRR, abs/1704.04861, 2017.

[45] [45]: W. Wu, C. Qian, S. Yang, Q. Wang, Y. Cai, and Q. Zhou. Look at boundary: A boundary-aware face alignment algorithm. In CVPR, 2018.

[46] [46]: A. Kumar and R. Chellappa. Disentangling 3d pose in a dendritic cnn for unconstrained 2d face alignment. In CVPR, 2018

[47] [47]: A. Jourabloo, X. Liu, M. Ye, and L. Ren. Pose invariant face alignment with a single cnn. In ICCV, 2017.

[48]: H. Yang, W. Mou, Y. Zhang, I. Patras, H. Gunes, and P. Robinson. Face alignment assisted by head pose estimation. In BMVC, 2015.