Real-time Weather Classification on Edge Devices

Introduction to Artificial Intelligence (Robotics) Final Report

Lin, Rui-Yu, Huang, Yi-Ning, Terence Lin
dept. of Electronics and Electric Engineering
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
Email: imrui.ee09@nycu.edu.tw, ynnh.ee09@nycu.edu.tw, sakana1219.ee09@nycu.edu.tw
Student ID: 109511208, 109511029, 109511219

***Abstract*—We introduce a Neural Network (NN)-based weather classifier [1] that can easily be deployed on Progarmmable Logic (PL), that is, System on Chip (SoC) Field Programmable Gate Array (FPGA). We want to make the weather classifier easily to use on edge devices that might not has large scale of computation resources like GPU or some other expensive hardware do. To accomplish this, we use a development board called Xilinx Kria™ KV260 to demonstrate another way to use a trained model without GPUs or cloud computing resources. First, we trained the model with pre-trained ResNet-18 weights. Second, we optimized the model and generated the file that can be used by PL with Vitis-AI. Finally, we designed the main function for the application and deployed the model on it. Since our main target is to make it easily usable on FPGA, we do not intend to provide a detailed discussion of the current state-of-the-art in this topic. Code is available at:** https://github.com/Terence1219/WeatherClassifierOnKV260

I. Introduction

Weather classification is an important and practical topic in real life. It helps making many decisions. For example, judging whether some product can work normally under certain weather. Specifically, determining if an image recognition product works under bad weather conditions with low visibility. It's critical to do this for ensuring the safety of many applications or products. But there is little research related to this topic. Moreover, most of them are finished in several years ago, and the precision are not comparable with other types of image recognition tasks. We have identified some factors that limit the precision of weather classification, which will be discussed in more detail in the conclusion section. We divide our project into the following parts so that it's easily to understand.

1) Train a model using PyTorch with Vitis-AI support, and save the model after training. Vitis-AI also supports other libraries such as TensorFlow and Caffe. For more information, please visit the Vitis-AI homepage [2], [3].
2) Optimize the model by inspecting, quantizing, and compiling it. After this, the model is ready for deployment on the programmable logic (PL).
3) Deploy the model on the Xilinx KV260 using the generated file from the previous step and with a few lines of Python code.

II. Related works

A. Terminology

PyTorch [4] is an open-source machine learning library that is built on the popular Torch library and provides a comprehensive set of tools for developing and training deep learning models. PyTorch has gained popularity due to its simplicity, dynamic computational graph and support for easy deployment to various platforms. PyTorch is widely used in various machine learning and deep learning tasks such as image classification, object detection, and natural language processing. It provides an easy-to-use API for creating and training neural networks, as well as support for popular libraries and frameworks, such as TensorFlow and Caffe. It's also support for a wide range of hardware including CPU, GPU and even FPGA.

OpenCV (Open Source Computer Vision) [5] is an open-source library of computer vision algorithms and tools. It provides a wide range of functionality for image and video processing, including image filtering, feature detection, and object recognition. OpenCV is written in C++ and also provides a Python API. It is widely used in both academia and industry for various computer vision applications. In this project, we use it to read the images from webcam and deal with other image processing tasks.

ResNet-18 is a deep convolutional neural network architecture that was first introduced in the 2015 paper "Deep Residual Learning for Image Recognition" [6] by He et al. This architecture was designed to improve upon the performance of deeper networks, which at the time were struggling to train effectively due to the problem of vanishing gradients. The ResNet architecture addresses this issue by introducing residual connections, which allow the gradients to flow more easily through the network.

FPGA (Field-Programmable Gate Arrays) are a type of programmable hardware that can be reconfigured to perform a wide range of digital logic functions. Unlike traditional ASICs (Application-Specific Integrated Circuits), FPGAs can be reprogrammed after they have been manufactured, allowing for flexibility in their use and the ability to adapt to changing requirements. FPGAs have become increasingly popular in recent years as a means of accelerating computationally

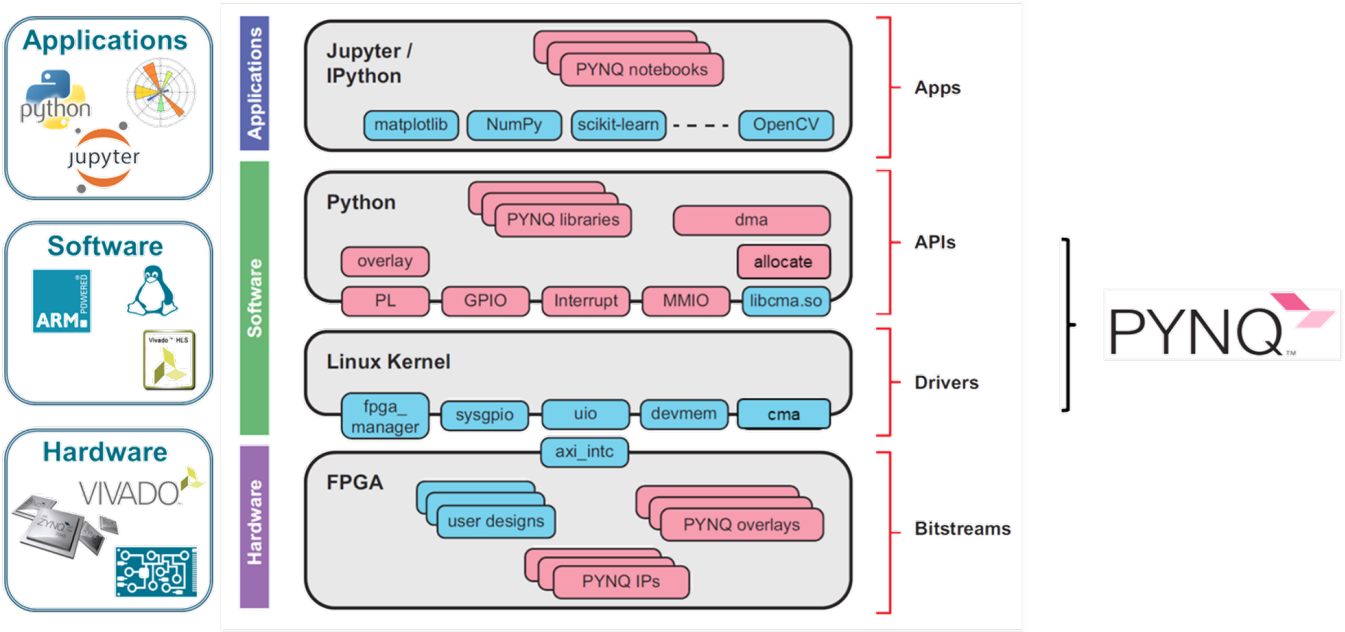

Fig. 1: PYNQ stack. [7]

intensive tasks, particularly in the field of artificial intelligence and machine learning. They are well-suited for these types of applications because they can be configured to perform high-speed parallel processing and can achieve low-latency, high-throughput performance. In addition, FPGAs are also known for their energy efficiency, which makes them particularly useful in edge computing and embedded systems.

PYNQ (Python productivity for Zynq) [8] is an open-source framework that enables the use of Xilinx FPGA for high-performance, flexible, and power-efficient embedded systems. Fig 1 shows the overall PYNQ stack. The PYNQ provides a Python-based programming interface that allows developers to take advantage of the high-level productivity of Python while still leveraging the low-level performance of FPGA. PYNQ allows developers to use pre-built, optimized intellectual property (IP) cores from Xilinx, as well as to implement their own custom logic. The framework also includes a wide range of libraries, such as for image processing, machine learning, and computer vision, which can be used to easily implement high-performance applications. This can be particularly useful for developers without experience in FPGA design, as they can take advantage of existing libraries and pre-built IP cores to quickly and easily develop FPGA-based systems.

KV260 (Xilinx Kria™ KV260) is a development board that features the Xilinx Zynq UltraScale+ MPSoC. This SoC (System-on-Chip) includes a built-in FPGA, and is comprised of two main components: a processing system (PS) and programmable logic (PL).The PS is a dual-core Arm Cortex-A53 processor, and the PL is an FPGA fabric that can be programmed to perform custom logic functions. The KV260 development board is designed for use in embedded systems and edge computing applications, and provides a range of peripheral interfaces and connectors for connecting to external devices. The Xilinx KV260 development board is an ideal platform for embedded systems and edge computing applications that require high-performance and low-power solutions. Due to the inbuilt FPGA and the support of PYNQ framework, it is a suitable platform to demonstrate the deployment of machine learning models on resource constrained devices.

Vitis-AI [9] is a software development platform from Xilinx that is designed for creating AI and machine learning applications on Xilinx devices such as FPGAs and SoCs. Vitis-AI provides a comprehensive set of tools and libraries for developing, optimizing, and deploying AI models, including support for popular frameworks such as TensorFlow and PyTorch. It includes a number of key features that make it well-suited for developing AI and machine learning applications on Xilinx devices. The detail of Vitis-AI is provided in the Experiment part of this report [2], [3].

B. Related Works

Weather Classification with Deep Convolutional Neural Networks (ICIP 2015) [10]. In [10] by Mohamed Elhoseiny et al, the authors propose a method for weather classification using convolutional neural networks (CNN) that outperforms the state-of-the-art at that time by a significant margin. They achieve a normalized classification accuracy of 82.2% on the challenging Image2Weather dataset, compared to the previous state-of-the-art accuracy of 53.1%. They also study the behavior of different layers in the CNN, and present interesting findings. The authors argue that weather classification using images is a vital task in various visual systems and the approach they provide is a cost-effective solution for this task. They also pointed out that, lack of the required human

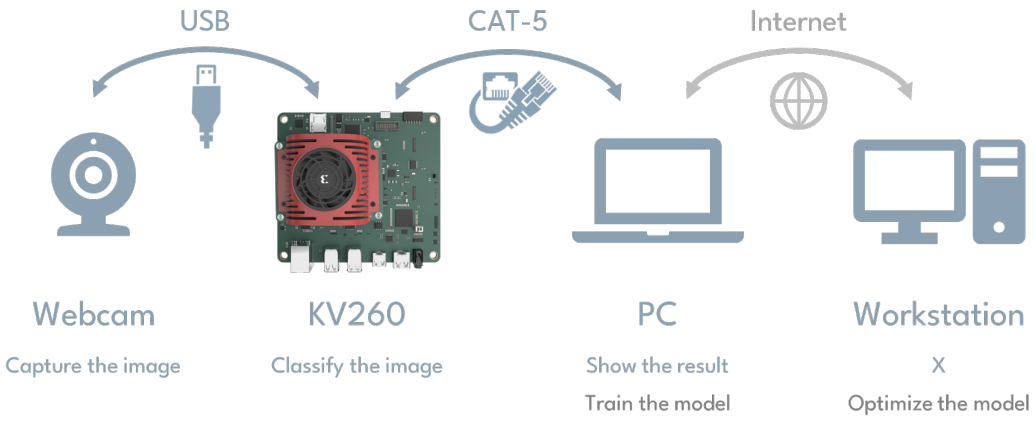

Fig. 2: The relationship of the hardware and their functions in this project.

resources and/or the expensive sensors limits the availability of local measurement of the weather condition, hence, the solution proposed in this paper can help to reduce these challenges by providing a high-accuracy model for weather classification. Since this paper was published in 2015, the accuracy is relatively low comparing with the state-of-the-art works nowadays.

A CNN-based Multi-task Framework for Weather Recognition with Multi-scale Weather Cues (ESWA 2022) [11]. In [11] the authors propose a new convolutional neural network (CNN) based multi-task framework for image-based weather recognition, which deals with the problem of recognizing weather conditions in outdoor images. The proposed approach utilizes the multi-scale weather cues in the images to improve the weather-cue segmentation and weather classification tasks by using a multi-scale weather-cue feature extraction process. Additionally, the authors introduce an adaptive weighting scheme to balance the two tasks, and evaluate their approach on two public benchmark datasets, demonstrating superior performance compared to existing methods. This article introduced a way that has good performance in weather classification, but it is hard for us to reproduce the project and deploy on KV260.

III. Experiments

A. Implementation

Part of the codes are modified from [2], [12], and the relationship between devices is shown on the Fig 2. To make the model ready for using on KV260, we first train our model on PC. Then, we upload it on the workstation which has the Vitis-AI docker. We optimized and compiled the model under an environment called vitis-ai-pytorch in Vitis-AI docker. For other hardware, the webcam is responsible for capturing the photos, and send the information to KV260 through USB. After that KV260 would predict the weather. The PC also play a role of displaying the results.

B. Datasets

In this project, we use the Image2Weather Dataset [13] for training. There are six categories in the Image2Weather Dataset: sunny, cloudy, snowy, rainy, foggy, and other. We only used sunny, cloudy, and rainy to train our model because snow and fog are rarely observed on campus, and the ”other” class is not directly related to the weather categories because it includes images such as indoor and night images that might increase the complexity of the problem. Table I shows the number of images and weight of each class in the Image2Weather Dataset. Due to different image sizes, we resized the images to 224×224 pixels and normalized them. We also used images in another dataset called Multi-class Weather Dataset [14] to test our model. The images in the Image2Weather dataset were collected in real-world scenarios, and we used them as training data. The images in the Multi-class weather dataset are exaggerated and rarely seen in the real world, so we only used them as test data.

Weather	Number	Weight
Cloudy	45,662	0.858
Rainy	1,369	28.6175
Sunny	70,501	0.5557
Total	117,532	-

Table I. The numbers of images and the corresponding weights of each class.

C. Model and Hyperparameters

We use ResNet-18 [6] as the backbone and add an output layer consisting of 3 neurons, one for each class: sunny, cloudy, and rainy. The weights is released at [15]. We use Cross Entropy Loss as our criterion to evaluate our model. Cross Entropy Loss is a method to measure the dissimilarity between predicted and true probability distributions. Adam Optimizer is used to improve our model which adapts the

learning rate of each parameter based on their gradient history. We also use the Step Learning Rate Scheduler, which is used to reduce the learning rate after a specific number of iterations to stabilize the training process. The step size is set to 5 and the gamma is set to 0.5, meaning that the learning rate will be halved every 5 epochs.

D. Handling the Imbalanced Data

The distribution of the dataset is imbalanced, often leading to decreased accuracy of the classes with fewer data. To deal with the problem, we come up with two methods.

First, weighted loss function method. We calculate the weight of each class according to its quantity. The equation 1 is shown below, and the weights are shown in Table I.

$$weight_{class} = \frac{1}{\#classes} \times \frac{\sum \#samples_{class}}{\#samples_{class}} \quad (1)$$

Where $weight_{class}$ represents the weight of the class, $\#classes$ means the number of the classes. In this case, we has 3 classes, so $\#classes = 3$. $\#samples_{class}$ is the number of samples in the class. The weights we calculated is used as a parameter in Cross Entropy Loss. This method is cost-sensitive learning, which would penalize more on high-weight classes to alleviate the negative effect of imbalanced data.

Second, Under-sampling method. It is a method to balance the data. It keeps the data of the minority class and randomly selects data from the majority class to the quantity of the minority class. We randomly selected the data and set the number of images of each class to 1369, which is equal to the original quantity of the rainy class. However, under-sampling may cause loss of valuable information.

E. Training

The images for training are split into 80% for training and 20% for validation. The model is trained on a RTX 3060 Ti GPU with a batch size of 64 for 15 epochs.

F. Running Model on KV260

To run our customized model on KV260, we use Vitis AI environment to quantize, optimize and compile our model. After transferring the model into a .xmodel file, we use VART Python APIs to load model into KV260. Fig. 3 shows the overall Vitis AI stack. Before executing the following steps, inspection is suggested, the related documents and files are available at [2].

G. Vitis AI Model Zoo

Because Vitis AI has a rich model zoo, including PyTorch, Tensorflow, Tensorflow 2 and Caffe. It provide pre-compiled models that are ready to deploy on FPGA, we apply existing ResNet-18 in vai_p_pytorch to our model.

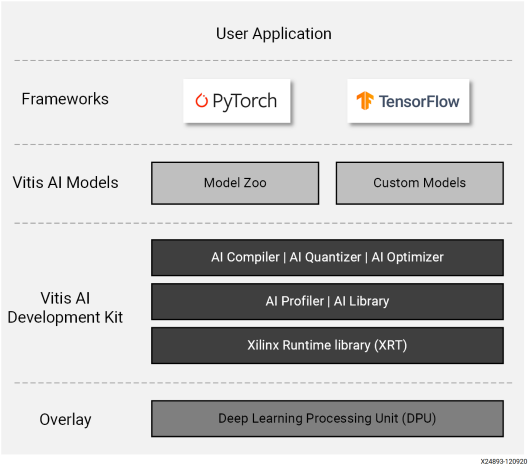

Fig. 3: Vitis AI Stack. [3]

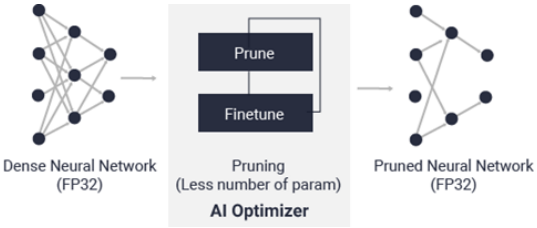

Fig. 4: Vitis AI Optimizer. [3]

H. Vitis AI Optimizer

Since the process of inference is computation intensive and requires a high memory bandwidth to satisfy the low-latency and high-throughput requirement of Edge applications, quantization is employed to compress model size and achieve high performance and high energy efficiency with little degradation in accuracy. Fig. 4 is a visualization of Vitis AI Optimizer.

I. Vitis AI Quantizer

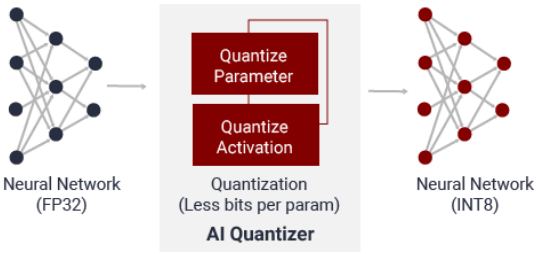

Fig. 5: Vitis AI Quantizer. [3]

Quantizer takes floating-point model as input and transforms it to fixed-point model. By converting the 32-bit floating-point weights and activations to 8-bit integer (INT8) format, which generate less memory bandwidth, perform faster running speed and have higher power efficiency. According to the documents, Vitis AI Quantizer supports convolution, pooling, fully-connected and batch-norm. Fig. 5 is a visualization of Vitis AI Quantizer.

J. Vitis AI Compiler

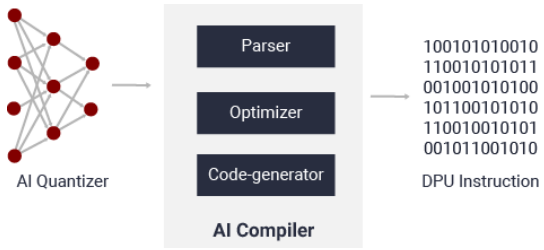

Fig. 6: Vitis AI Compiler. [3]

Compiler maps the AI model to a highly-efficient instruction set and dataflow model. It transforms the model into XIR-based computing graphs. To optimize, compiler breaks up the graph into several sub-graphs on the basis of whether the operation can be executed on the DPU, which can process data more efficiently. After compiling the model, we will get a .xmodel file to load into KV260 for later process. Fig. 6 is a visualization of Vitis AI Compiler. Fig. 18 on page 9 is the model graph of our model.

K. Deploying and running the model

Vitis AI provides VART python API to deploy model on KV260. First, we use the object DPU Overlay from module pynq_dpu to load the model. Then, we create an instance of DPU runner, which can get input and output tensor and execute the model.

L. Results

Model performance. The first model is the model we deployed on KV260, and we test the model with the images captured by the webcam connected to KV260. We use the weighted loss function to handle the imbalanced data. As shown in Fig. 7, the accuracy of training and validation is nearly 90%. However, the accuracy for each class varies from class to class as shown in Fig. 8. It shows that the rainy class, which has the fewest images, has the worst accuracy compared to other classes. The weighted loss function is not sufficient to eliminate the imbalanced problem.

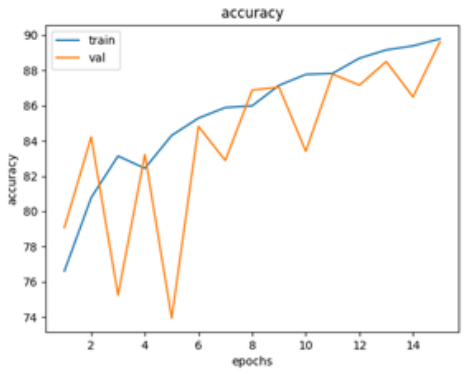

Fig. 7: Training and validation accuracy of the first model.

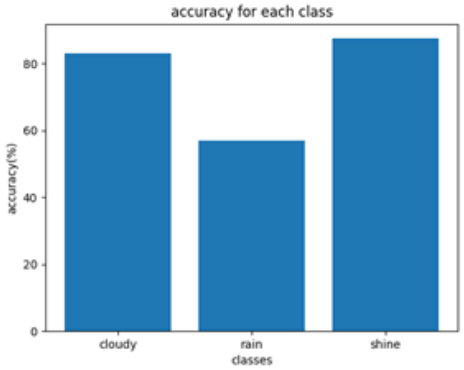

Fig. 8: Accuracy for each class of the first model.

To improve the imbalanced problem, the second model uses the under-sampling method. However, it is not deployed on KV260, we only test the model with photos we took on the computer and the result is shown in Fig. 11. As shown in Fig. 9, the accuracy of training is nearly 100% due to fewer data, and the validation data only has 85% accuracy. In Fig. 10, the accuracy for each class is more average over different classes, but the cloudy class is a little lower than other classes.

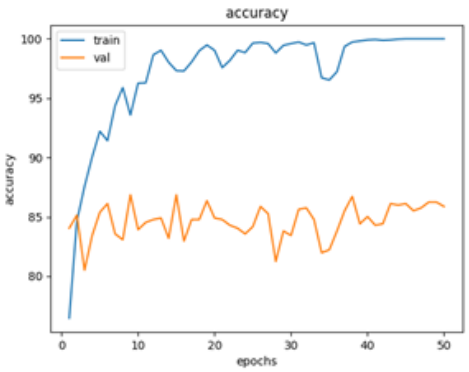

Fig. 9: Training and validation accuracy of the second model.

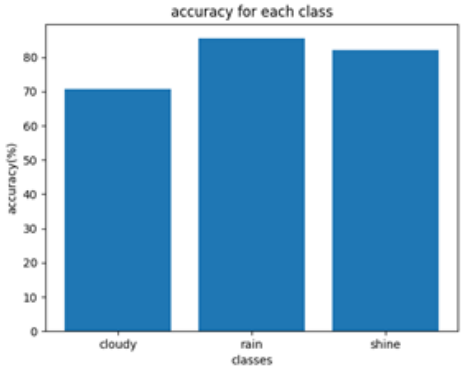

Fig. 10: Accuracy for each class of the second model.

Fig. 11: The prediction of the second model on real scene.

KV260 performance We test KV260 on (1) Image2Weather dataset, (2) Multi-class weather dataset (MWD) and (3) Real world data. We collected the real world data in the NYCU library, and Girls Dormitory 2.The way we capture the picture is to target the picture using webcam. Because of the resolution and brightness of webcam we use, the picture we capture is a kind of data augmentation from original picture. The results are shown on pages 7 and 8.

M. Obstacles

Hard to test in the real world. KV260 needs a stationary power supply, so it is not convenient to bring it outside to test our work. In addition, the weather outside is the same in a certain period. We come up with an alternative way, using our notebook to show various photos and the webcam to capture the image displayed on the screen. In the real world, there might have many factors that affect our model's performance, such as dirty windows, low-quality webcams, and window film. These factors may blur the images we captured, and our model would misclassify the image to cloudy class. The light also affects the result very much, if the light is not much enough, the result is likely to be rainy or cloudy despite the weather being sunny.

Lack of Online Resources. Most of the online resources we need have been removed, including Xilinx official documents. It's very difficult for us to cope with the problems we encountered without those resources. The only resource we can rely on is the lab materials written by TA. We trace the code and edit it to fulfill our work. Trial and Error is the way we deal with the problem.

N. Future Works

Building more robust model, we can try to apply over-sampling, Focal Loss, and label smoothing [16] to deal with the imbalanced data. By data augmentation, we can avoid overfitting. Increasing the number of classes and Open Set Recognition(OSR) [17] are also the possible solutions that can make our model able to recognize more weather and classify non-weather images into unknown class.

IV. Conclusions

In this project, we have successfully made our ResNet-18 weather classifier available on KV260 for real time weather classification. By inspecting and quantizing the model with Vitis-AI, we optimized our model, including reducing the size of the model which reduced the computation. After that, we compiled our model, making it available on KV260. Under normal conditions, our model achieve to recognize the weather most of the time. The error estimation usually occurs when the sky is masked, or with dim light due to the color variation caused by the automatic correction of the webcam. By adjusting the learning strategies, our model has the potential to perform more accurate judgement to distinguish the bad conditions and remind the user if the AI is trustworthy.

References

[1] Terence1219, yining213, and RUI030, "Weather Classifier on KV260," 2022. Available at: https://github.com/Terence1219/WeatherClassifierOnKV260.
[2] Xilinx, "Vitis AI, GitHub.," 2022. Available at: https://github.com/Xilinx/Vitis-AI.
[3] I. Advanced Micro Devices, "Vitis AI User Guide (UG1414), AMD Xilinx.," 2022. Available at: https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/Vitis-AI-Overview.
[4] T. L. Foundation, "PyTorch.." Available at: https://pytorch.org/.
[5] O. team, "OpenCV.." Available at: https://opencv.org/.
[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *CoRR*, vol. abs/1512.03385, 2015.
[7] P. Maliński, "Overview of PYNQ project offering FPGA capabilities to Python and data engineers, RkBlog.," 2020. Available at: https://rk.edu.pl/en/overview-pynq-project-offering-fpga-capabilities-python-and-data-engineers/.
[8] "PYNQ: PYTHON PRODUCTIVITY.." Available at: http://www.pynq.io/.
[9] I. Advanced Micro Devices, "Vitis-AI, AMD Xilinx.," 2023. Available at: https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html.
[10] M. Elhoseiny, S. Huang, and A. Elgammal, "Weather classification with deep convolutional neural networks," 09 2015.
[11] K. Xie, L. Huang, W. Zhang, Q. Qin, and L. Lyu, "A CNN-based multi-task framework for weather recognition with multi-scale weather cues," *Expert Systems with Applications*, vol. 198, p. 116689, 2022.
[12] Louis5228, "kv260-lane-following, GitHub.," 2022. Available at: https://github.com/Louis5228/kv260-lane-following.
[13] W.-T. Chu, X.-Y. Zheng, and D.-S. Ding, "Image2Weather: A Large-Scale Image Dataset for Weather Property Estimation," pp. 137–144, 04 2016.
[14] Z. Zhang, H. Ma, H. Fu, and C. Zhang, "Scene-free multi-class weather classification on single images," *Neurocomputing*, vol. 207, pp. 365–373, 2016.
[15] "ResNet18_Weights.IMAGENET1K_V1, PyTorch.." Available at: https://pytorch.org/vision/main/_modules/torchvision/models/resnet.html#ResNet18_Weights.
[16] M. F. Tandia, "Some Tricks for Handling Imbalanced Dataset (Image Classification)," 2021.
[17] C. Geng, S. J. Huang, and S. Chen, "Recent advances in open set recognition: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 10, pp. 3614–3631, 2020.

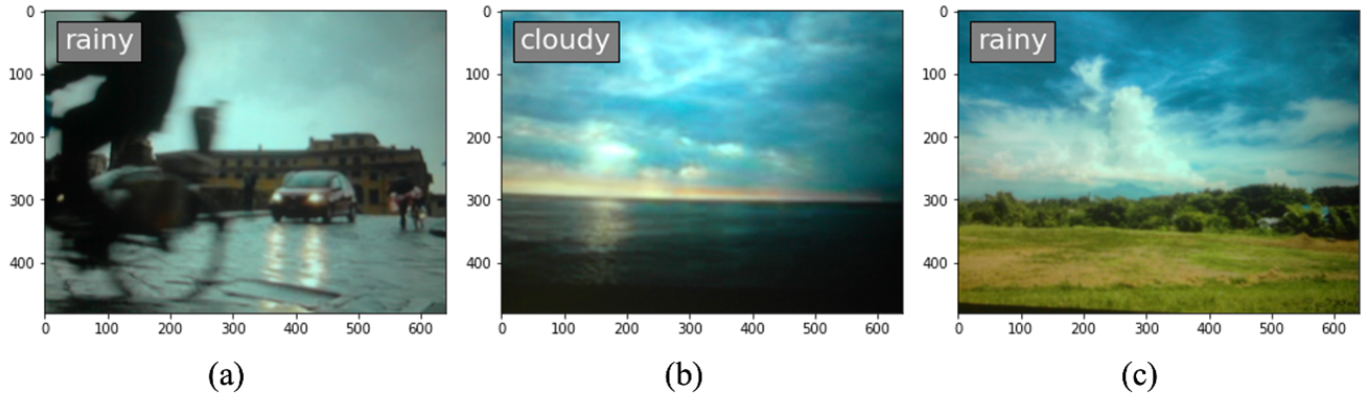

Fig. 12: Prediction using Image2Weather dataset. (a) Correct prediction of rainy condition. (b) Correct prediction of cloudy condition. (c) Incorrect prediction, which predict cloudy conditions to rainy.

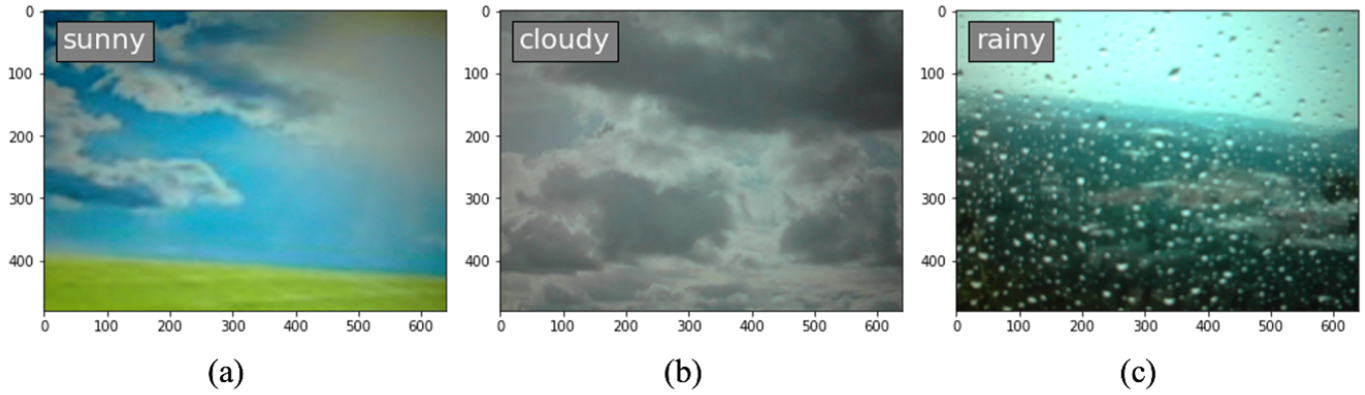

Fig. 13: Prediction using Multi-class weather dataset. (a) Correct prediction of sunny condition. (b) Correct prediction of cloudy condition. (c) Correct prediction of rainy condition.

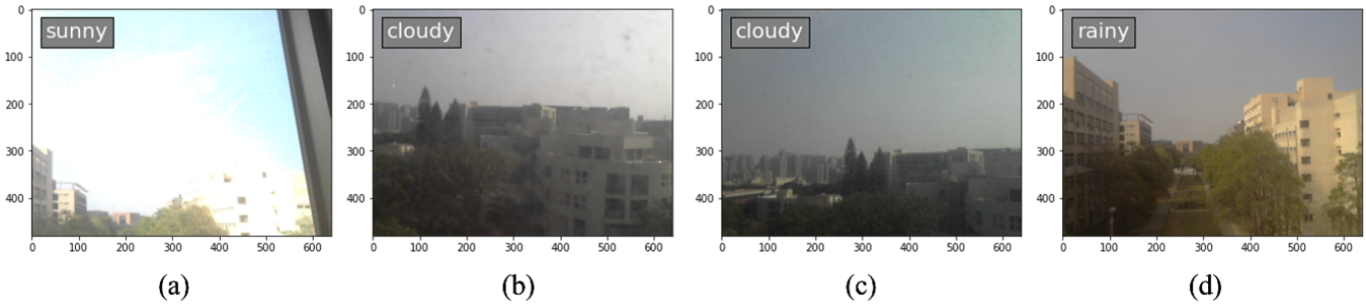

Fig. 14: Prediction using real time data captured in campus's library. It was sunny that day, so we got sunny data. (a) Correct prediction of sunny condition. (b) Incorrect prediction of sunny condition. (c) Incorrect prediction of sunny condition. (d) Incorrect prediction of sunny condition.

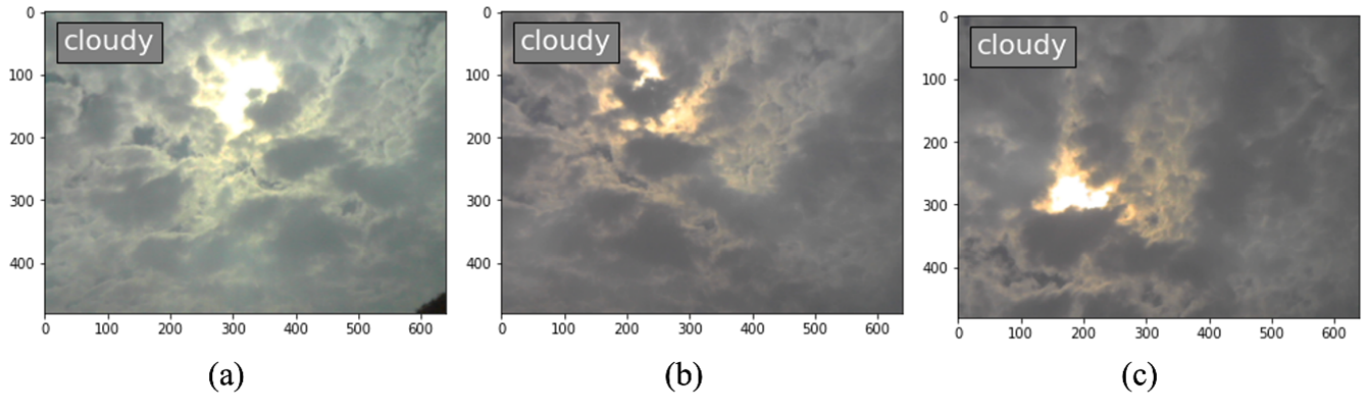

Fig. 15: Prediction using real time data captured at Girls Dorm 2 in the NYCU campus. It was cloudy that day, so we got cloudy data. For those pictures that clouds covered the most part, we got high accuracy of prediction. (a)(b)(c) Correct prediction of cloudy condition.

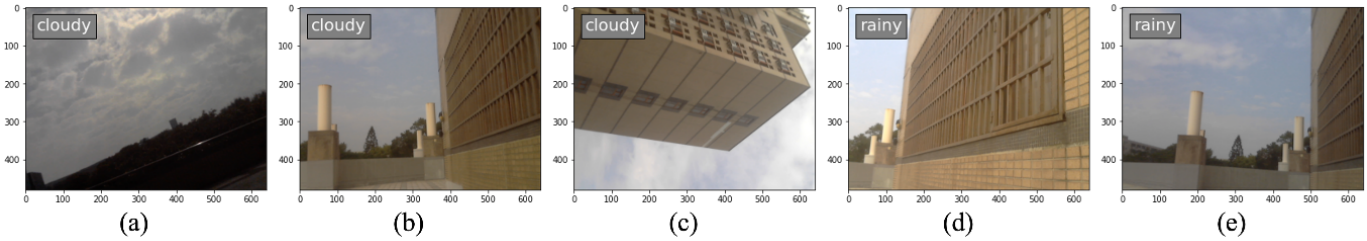

Fig. 16: Same as Fig. 15. Prediction using real time data captured at Girls Dorm 2 in the NYCU campus and all under cloudy condition. However the pictures above are all partially covered by buildings or other scene and are rotated with different angle. (a)(b)(c) Correct prediction of cloudy condition (d)(e) Incorrect prediction that cloudy condition is predicted to rainy.

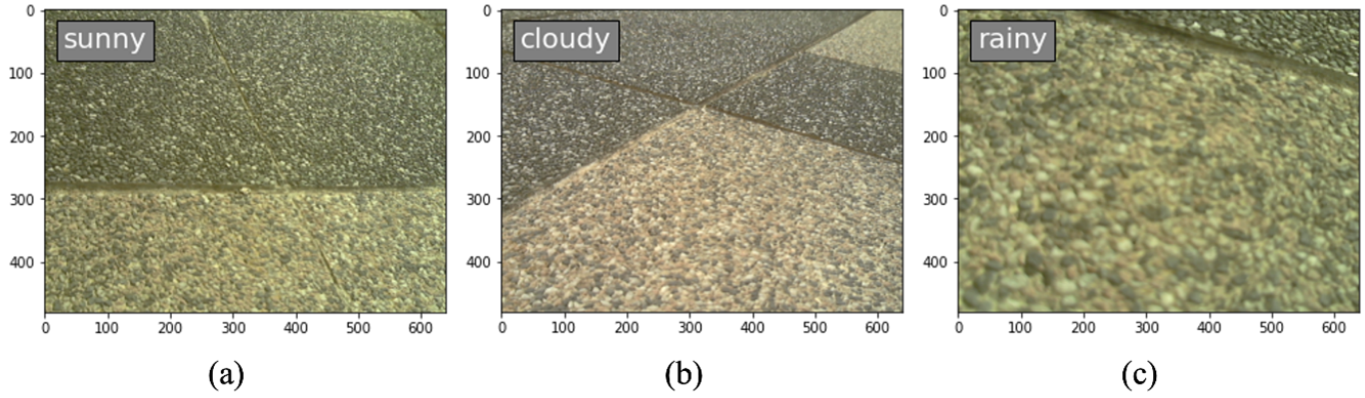

Fig. 17: Same as Fig. 15. Prediction using real time data captured at Girls Dorm 2 in the NYCU campus, and all under cloudy condition. However the pictures were focus on the floor. Without the cloud information, it seems that we got low accuracy. This may results from lack of such kind of data in our training dataset. (a)(c) Incorrect prediction of floor picture under cloudy condition. (b) Correct prediction of floor picture under cloudy condition.

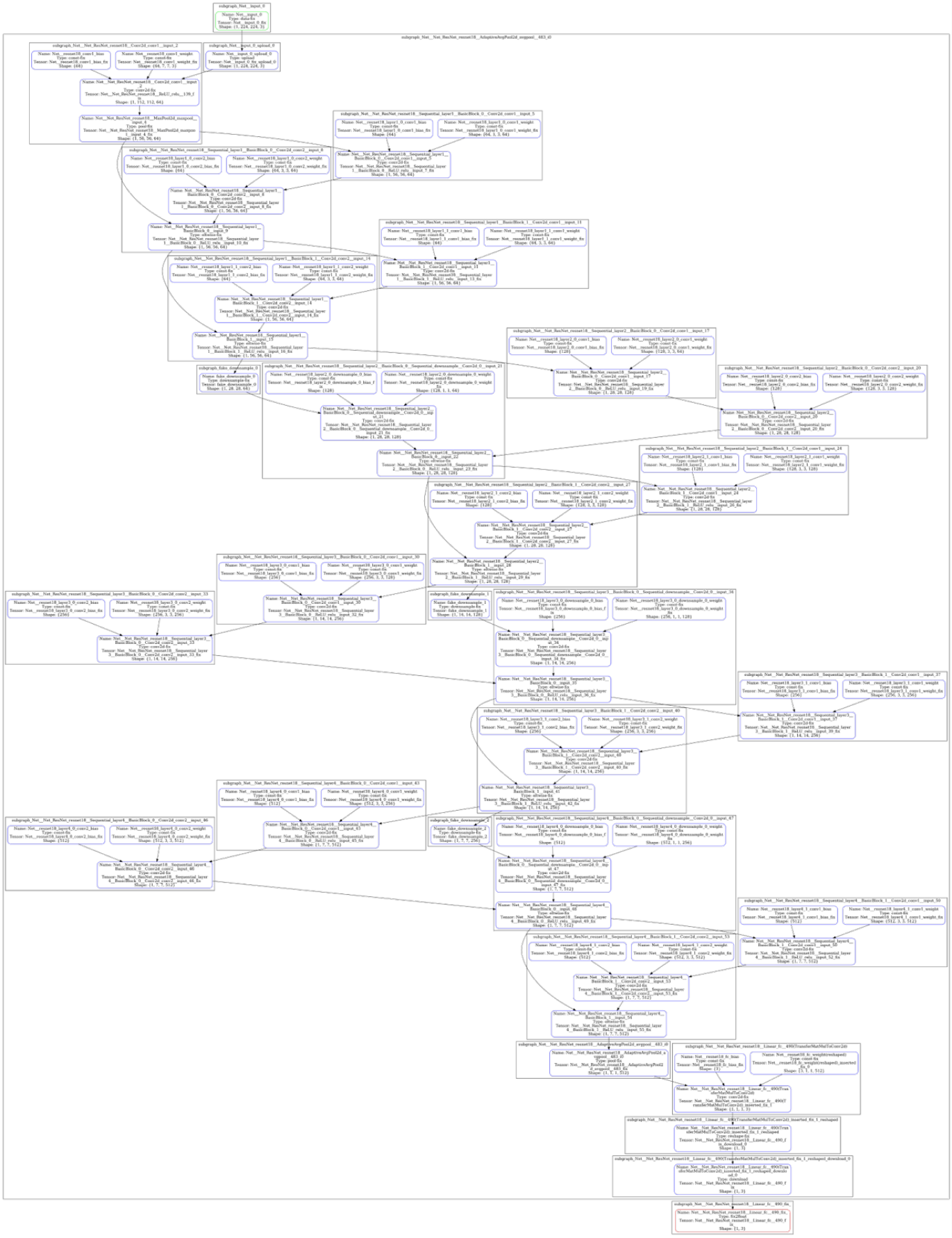

Fig. 18: The model graph of our model generated by Vitis AI Compiler.