Accelerating Video Analytic Processing on Edge Intelligence

Pedro Fernández, Jaime Jiménez and Armando Astarloa
Departamento de Tecnología Electrónica
University of the Basque Country (UPV/EHU)
Bilbao, Spain
armando.astarloa@ehu.eus

Mikel Idirin and Sergio Salas
System-on-Chip *engineering*
Ribera de Axpe 50 Erandio, Spain
mikel.idirin@soc-e.com

***Abstract*—The most demanding Artificial Video analytic applications require in-edge inference of the AI model to ensure low-latencies to obtain the result. In general, the training process of the model can be executed on the cloud taking benefit from the high-performance computing capabilities available on those premises.**

This work presents an AI Video analytic application implemented on an Edge-computing device. This device is capable of accelerating the inference of AI models and Video compression by dedicated hardware.

This paper presents the architecture designed to implement Image, Networking and Deep Learning Processing functionalities on a reconfigurable System-on-Chip. Additionally, the design tools and design flow followed to generate all software and hardware configuration is detailed.

This Edge Intelligence platform is currently in-service, providing the preliminary results for the targeted applications. The proposed solution can process 33 times more video data volume in real-time than the software GPU accelerated implementation for the testing conditions described in the paper.

***Index Terms*—AI, NN, DNN, CNN, , FCN, RNN, DPU, SOC**

I. Introduction

Edge Intelligence (EI) combines edge computing and Artificial Intelligence (AI). Real-time video analytics is a killer application for edge computing [1], [2], [3]. Critical systems like automotive or Aerospace&Defence demand low latency in the analysis of the videos. Additionally, the resolution of these video sources increases continuously. Thus, the traditional AI Cloud-based approach requires higher bandwidth and speed networking, making this approach inviable for many applications. One viable approach is edge computing with dedicated hardware acceleration for Video coding, decoding, and Deep Learning Processing units to accelerate AI computing.

The most demanding AI Video analytic applications require in-edge inference of the AI model to ensure low-latencies to obtain the result. In general, the training process of the model can be executed on the cloud taking benefit from the high-performance computing capabilities available in those premises.

This work was partially supported by ECSEL Joint Undertaking in H2020 project IMOCO4.E, grant agreement No.10100731 and by the Department of Education of the Basque Government within the fund for research groups of the Basque university system IT1440-22.

This work presents an AI Video analytic application implemented on an Edge-computing device that specifically targets real-time object detection, in this case face detection. The main point of interest is to demonstrate how this device is capable of accelerating the inference of AI models and Video compression by dedicated hardware.

In Section II an introduction to Deep Neural Networks (DNNs), Convolutional Neural Networks (CNNs) and Densely Connected Convolutional Networks (DenseNet) is presented. These models are the base for the AI application implemented. The hardware acceleration is achieved using FPGA technology embedded on a reconfigurable SoC. Section III-B introduces briefly the use of these technology for this purpose.

Section IV details the SoC architecture implemented to run the accelerated AI Video analytic application. In this section, the AI/ML Design Flow followed to implement the design, the High level Architecture and the Data Flow path inside the SoC are presented.

Section V summarizes the preliminary results obtained in the set-up and the paper ends with the conclusions and future work in Section VI.

II. State-of-the-art

A. Deep Neural Networks (DNNs)

Deep Neural Networks (DNNs) are currently the base for many modern Artificial Intelligence (AI) applications [4]–[6]. DNNs are employed in applications from selfdriving cars [7], tumor detection [8] to gaming [9]. In many of these domains, DNNs are now able to exceed human accuracy. DNNs are able to extract high-level features from raw sensory data after using statistical learning over a large amount of data to obtain an effective representation of an input space.

B. Convolutional Neural Networks (CNNs)

A weight-shared and windowed DNN layer implements the computation as a convolution. The weighted sum for each output activation is computed using a small neighborhood of input activations and the same set of weights are shared for every output. This is named as the 'receptive field'. These convolution-based layers are referred to as convolutional (CONV) layers.

Convolutional Neural Networks (CNNs) are composed of multiple CONV layers. Each layer generates a successively higher level abstraction of the input data, called a feature map (fmap), which preserves essential yet unique information.

CNNs are widely used in a variety of applications including image understanding [6], robotics [10] and speech recognition [11].

CNNs composes of the following layers:

- **CONV layers:** They implement high-dimensional convolutions. The input activations of a layer are structured as a set of 2-D input feature maps (ifmaps), each of which is called a channel.
- **Fully connected (FC) layers:** All output activations are composed of a weighted sum of all input activations.
- **Nonlinearity layers:** They implement a nonlinear activation. They are located typically after each CONV or FC layer. Typical nonlinear functions are sigmoid or hyperbolic tangent as well as rectified linear unit (ReLU) [12].
- **Pooling layers**: Implementation computations that reduce the dimensionality of a feature map pooling a set of values in its receptive field into a smaller number of values. They are applied to each channel separately enhancing the robustness of the network.
- **Normalization layers**: They control the input distribution across layers improving accuracy and speeding up training and improve accuracy. They are normalized such that it has a zero mean and a unit standard deviation.

CNNs are in continuos evolution. Lenet [13], Alexnet [6], Overfeat Fast [14], VGG-16 [15], Googlenet [15] and ResNet [16] are good examples of how they have evolved since 1989 till now. As an example, one of the latest models, ResNet [17] (Residual Net) was the first entry DNN in ImageNet Challenge that exceeded human-level accuracy with a top-5 error rate below 5% using residual connections to go even deeper (34 layers or more).

C. Fully Convolutional Networks (FCNs)

A well-known CNN standard is the Fully Convolutional Network (FCN). This variant of CNNs is taking advantage of the advances provided by convolutional layers in recognition, which induces improvements in tasks such as image classification [16] or object detection by bounding box [14], which is implemented for the purpose of this work.

The main characteristic of FCNs is that their architecture is composed entirely of convolutional layers, eliminating the fully connected layer of them. As the Figure 1 shows, the operation of these networks is based on making use of a convolutional network to convert the pixels that make up the image into classes of pixels. That is, they classify objects based on the shape of the pixels of a particular object class in an image, leaving aside the spatial position of the features to focus on the position of the object itself. As such, they are especially useful in object localization and detection.

The basic design of the fully convolutional network model consists of the following layers:

- **CNN network**:The characteristics of this type of networks are explained in Section II-B. Its function is to extract the characteristics of the different images.
- **1x1 convolutional layer**: The purpose of this layer is to convert the number of channels into the different classes to be detected.
- **Transposed convolution layer**: During the convolution and grouping operations performed by the CNNs layers, the spatial dimensions of the image are reduced. This layer transforms the height and width of the feature maps to those of the original image.
- **Output channel**: Contains the predicted classes for the input pixel at the same spatial position.

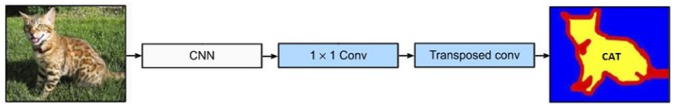


Figure 1. Basic operation of FCN networks.

FCNs represent a way to speed up the training process by reducing the parameters required by dense layers. Methods based on fully convolutional neural networks [18], [14], represent a revolution in the field of object detection and their parameter learning approach these convolutional neural networks significantly improve performance.

III. FCN based Network (DenseBox) on embedded systems

The large number of situations that require real-time detection capabilities has had an impact on the development of advances in neural networks that have this task as their main purpose.

Due to the improvements introduced by the CNNs [19], object detection methods [20] have evolved. Nowadays, the success of CNNs has led to methods which improve detection based on YOLO networks [21], R-CNN [22] or the FCN network highlighted in Section II-C, to gain importance in the field of object detection.

Methods such as R-CNN improve the efficiency in object detection, however, this is negatively influenced by small objects such as faces due to low resolution [23]. Moreover, object bounding box generation and object classification in these networks cannot be jointly optimized making end-to-end training of the network difficult.

An improvement of small object detection using a unified end-to-end detection pipeline is the FCN-based Densebox network. The approach of this network allows detecting objects under small scales and strong occlusion with results that place Densebox as a clear candidate for the improvement of efficiency in the detection applied to applications such as face or vehicle detection [24].

A. Densebox Layers

The layers that make up the Densebox network are based on the VGG-19 image classification model. Densebox is composed of 12 layers initialized by the VGG-19 model where three pooling processes are connected to the first layer of the next set of convolutional layers. Finally an up-sampling to resize the image to the original size is performed.

The convolutional layer connected before the third pooling process is fed into the up-sampling layer by concatenating it with the last convolutional layer based on VGG-19. The features learned by this layer are used to enhance the learning of the pre-up-sampling layer. Densebox concatenates these layers to improve detection performance.

The output of the last VGG-19 convolutional layers is fed into four other 1×1 convolution layers. The first pair of convolutional layers output a 1-channel map for the class score and the second two predict the position of the bounding box using a 4-channel map. At the output of the first of these layers a Dropout is applied to reduce the overfitting of the network and at the output of the second one negative mining is applied to search for the samples that give a negative result (The object is not in that pixel) to subsequently add them to the network learning so that gradient descent learning on those samples leads to a more robust prediction.

B. *FPGAs for Deep Learning Computing*

CNNs and Recurrent Neural Networks (RNNs) requires complex and intelligent information processing. The hardware required for this processing needs to address the following challenges [25]:

- High computational processing.
- Cost efficiency and/or low power consumption.
- Scalability techniques and architectures to accommodate different networks, sizes and topologies.

FPGA technology compared with CPU and GPU technology shows some advantages to address these challenges. The hardware implementation can be customized to meet the specific requirements of the algorithm to be implemented. For example, DNNs can be implemented in FPGA to only perform the required target logic. This leads to an efficient hardware implementation which is capable of higher computational processing and higher energy efficiency.

FPGAs give the flexibility of the platform to allow the researcher to develop different deep learning application in very short timeframe. The rich set of programmable logic cells and embedded components, such as DSP, allow it to perform arithmetic intensive operations.

IV. SoC Architecture

The SoC systems capabilities support different levels of efficiency. The FPGA shows several not inconsiderable advantages that make them a valuable option for machine learning applications.

With this in mind, to achieve this acceleration of real-time AI applications using embedded devices, specific tools are needed. In this respect, the Deep Learning Processing Unit (DPU) block is of great relevance. The DPU is a configurable hardware block to perform and accelerate convolutional neural network applications in SoC systems.

The solution proposed in this work involves many of the resources offered by these systems. For the development of the work, it is necessary to carry out different phases that include training the machine learning model and designing the appropriate hardware and software environment.

All these steps are relevant for implementing the application for real-time intelligent face detection mentioned in the introducction by using video recording through network cameras.

A. *AI/ML Design Flow*

To fulfil the main objective of the present research, it is necessary to carry out an adequate training of the neural network. The diagram shown in Figure 2 summarises the steps involved. For the application, `WIDER Face kaggle` dataset has been used to train the network.

All these steps are carried out by the Vitis AI compiler. An environment offered by Xilinx that includes the libraries of the main frameworks for the development of neural networks as well as additional tools for the generation of models implementable on Edge.

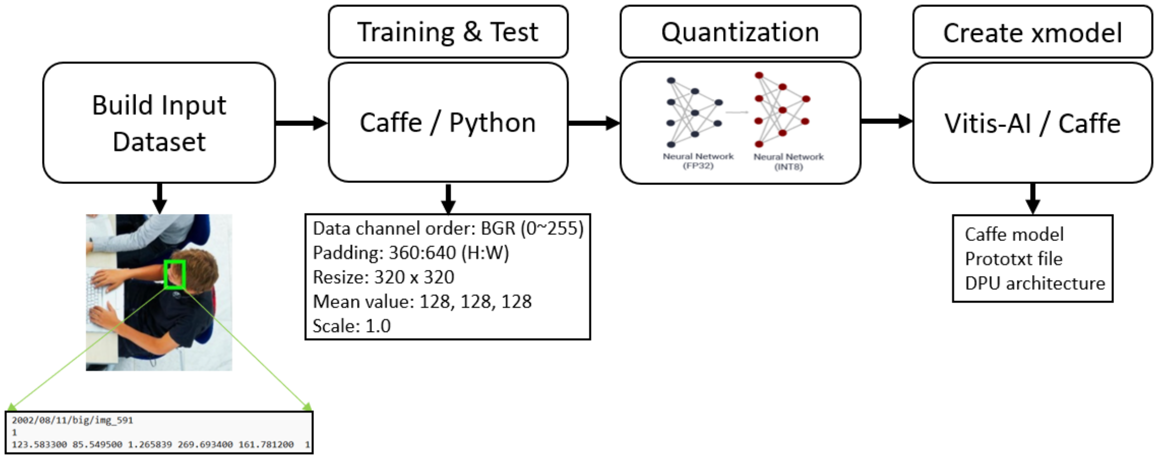
Firstly, it is necessary to adapt the input data to the network to train the model. For the face shape learning application, each image must provide the vector position of the learning target.

The Densebox network is trained and tested using the Caffe framework for machine learning. The input image size is 320x320 pixels, and the mean and scale value is 128 and 1, respectively. Once tested, it is necessary to quantize the model using integer computational units and the representation of weights and activations by lower bits.

Quantization is necessary as the inference process requires high computational power and bandwidth to achieve optimal latency and throughput values on the Edge. During this step, the 32-bit floating point weights and activation values used during network training are converted to 8-bit integers to reduce the complexity of the requirements while maintaining the accuracy of the model. For this process, the data layers file, called `prototxt`, and the file with the weights of the pre-trained model, defined by the extension `caffemodel`, are needed. The Vitis AI compiler uses the files to generate the quantized network description `prototxt` file and quantized Caffe model parameter `caffemodel` file

Finally, the quantified Caffe model, the `prototxt` file with the network specifications, and the architecture defined for the DPU used to generate the implementable model file through the Vitis environment. The generated output of the flow is a `xmodel` file that contains the required instructions and network information for the DPU to run the model on the embedded system.

It should be noted that the implemented model is only suitable for this specific application and neural network. For similar applications based on real-time object detection, it is not necessary to change the hardware design but the implementable model has to be regenerated so that these new weights and layers of the network are correctly interpreted by the IP block. In order to do this, it is necessary to repeat what



Data channel order: BGR (0~255)
Padding: 360:640 (H:W)
Resize: 320 x 320
Mean value: 128, 128, 128
Scale: 1.0

```
2002/08/11/big/img_591
1
123.583300 85.549500 1.265839 269.693400 161.781200  1
```

Figure 2. Model compilation process.

has been discussed above regardless of the AI framework used (Caffe, Tensorflow or PyTorch).

B. *High level Architecture*

Different types of tools are necessary for the proposed solution to develop the system properly. This system comprises several hardware blocks to implement each functionality. In the diagram illustrated in Figure 3, the communication between the processing system (PS) and the programmable logic (PL) can be seen.

- The input video pipeline that enters the images into the system using Gstreamer tools.
- The video converting pipeline involves encoding/decoding tasks for the data.
- The accelerator that comprises the PL hardware blocks for acceleration functions.
- The adaptation of the co-ordinates obtained by the Machine Learning model and the mapping to the original stream.
- The output pipeline refers to the output data capture with Gstreamer.

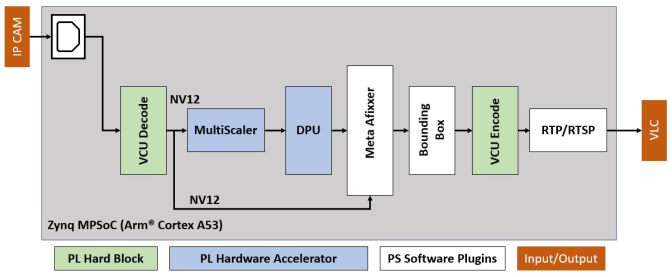


Figure 3. Hardware/Software architecture diagram.

This entire system architecture is composed of four different stages. Each stage is broken down into independent phases.

The Video Codec Unit (VCU) accelerates video processing. It is implemented in the PL region of the FPGA. This block encodes the video streams from and to H.264/H.265 video standards. Frames are treated differently depending on the targets. In this case, the data is decoded for further processing for machine learning.

Another step included in the architecture diagram is the video pre-processing. This task is performed by the Multi-scaler IP block located in the PL. This IP block function is to process the input data in order to modify it as required for the model specifications. Those are defined by a kernel that includes the values of the color code (BGR or RGB), the mean, and the scale of the concrete network to adapt the input data by:

- **Cvtcolor:** It reads and converts the NV12 video color format to BGR.
- **Resizing:** It scales down the original frame to at most 720x720 pixels.
- **Quantizing:** It performs linear transformation (scaling and shifting) to each pixels of BGR frame to satisfy DPU input requirement.

Once the data is processed, it can be interpretable by the neural network system.

As mentioned above, the most relevant block of the project is the DPU, used for the acceleration of Machine Learning applications. It is a block implemented in the SoC system that internally, it comprises a high-performance scheduler module, a hybrid computing array module, an instruction fetch unit module, and a global memory pool module.

The computational engine is governed by instructions that the unit fetches from off-chip memory. The instructions are generated by Vitis software to perform substantial optimizations. On the other hand, the on-chip memory buffers the data to achieve high efficiency and performance.

To carry out machine learning operations, a specific level of adaptation of the network is necessary to the hardware. The DPU IP block is developed and implemented under different specifications. These features provide the embedded systems with the ability to accelerate the applications. They include the possibility of implementing multiple cores to improve performance. Furthermore, it can be configured with a group of architectures related to the parallelism of the convolution unit. They require different programmable logic resources to achieve higher or lower performance. Other characteristics such as RAM usage, ReLU type or average pool are offered. All these characteristics determine the architecture used to build the DPU. The architecture needs to be included after the network training process for the network implementation over the PL. The DPU requires the model bitstream compiled using the Vitis environment for proper functioning.

Finally, there are three software blocks implemented in the processing system of the SoC. Refering to the first one, the scaled frames of video that are processed by the Multiscaler and the DPU might not be the best for the output display. The output of the VCU decoder block is fork into two streams, one for the accelerator blocks and the other for the display. The software converts the detection co-ordinates obtained by the model and maps them to the original video stream.

The second block finishes the application process by performing the task of drawing boxes around the objects identified by the network model. This software is launched by a kernel which includes the path to the bounding box application.

The last one is designed to compress the data payload into RTP packets to establish the RTSP communication and complete the last four pipelines.

C. Data Flow path

The responsible for generating the input/output information is the `GStreamer` framework. This framework allows for the implementation of multimedia solutions in different work scenarios.

To ensure data communication is necessary to define the correct network IPs for each device involved in the process. As it can be seen in the Figure 4, attending to the IP camera default IP direction, the FPGA is connected to the same network switch and its corresponding IP is assigned. The IP direction will be used to establish the VLC Client on the client-side. All the IPs must be in the same subnetwork.

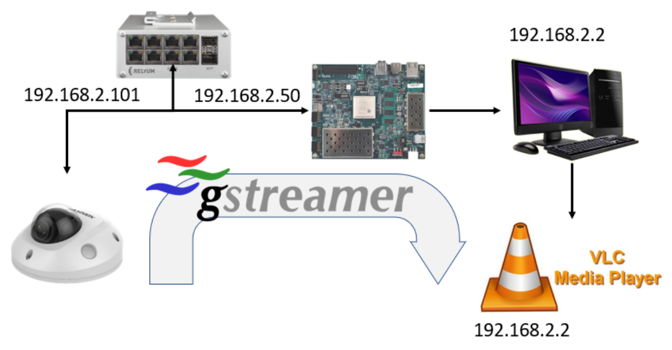


Figure 4. High level image dataflow representation.

Once the setup is ready, the `GStreamer` application is executed. Initially, the IP camera generates the data input, which uses the H.264 standard. Gstreamer creates an RTSP sink capturing directly from the camera.

The data is decoded in the VCU block, which sends it to the pre-processor software block to adapt this information into a valid DPU input. This software is launched by the kernel run into the `GStreamer` application. The format NV12 is converted to BGR, the images are resized, and input quantizing blocks are to meet the requirement of the DPU AI inference engine. These steps are done in the dedicated preprocessing IP to achieve the optimal framerate and latency.

The following kernel is referred to as the DPU block. At this point, the called application is executed together with the xmodel file generated during training. During the DPU process, the input frames that the pre-processor step has properly defined are analyzed and detected by the model implemented in the programmable logic through the DPU. The application gets the required tensors, including size and vectorial position to detect the objects. Then the results are sent to the Bounding Box block containing the software application and box results.

Finally, the VCU encoder encodes again in the H.264 standard, and the RTP sink is created via the `GStreamer` software. With the appropriate IP assigned to the client host, VLC is waiting to establish the communication with the server implemented in the FPGA to show the data output results.

V. Results

This Edge Intelligence platform is currently in-service, providing the preliminary results for the targeted applications.

For the face shape learning application tested in this set-up, the framerate of the IP cameras has been modified to stress the system. The maximum quality and the maximum framerate allowed by the IP camera (4K resolution and framerate 25 fps) have been selected. In these conditions and using a single thread to run the two DPU instances implemented in the design, there is no performance degradation in identifying shapes and video displaying compared to the lighter values tested (6 fps and 1240x720 pixels of resolution). This result is consistent with the theoretical values for this Neural Network and the DPU model used (2x4096). They are scored up to 440 fps for a single thread, more than 1500 fps for multithreading and a DPU processing latency of 2.26 ms [26]. Multithreading increases performance based on the FPGA's ability to split the DPU tasks by defining these software threads in the application running on the MPsoC that run in parallel in hardware, similar to a multicore processor. The number of processors that can be put into a specific FPGA is limited by the size of the processor, the available logic gates and memory in the FPGA, and the ability to connect it all and meet the timings.

The same Neural Network model and configuration were tested on RTX-2070 Mobile GPU. The face detection and box draw has been limited to an image size of 320x320 pixels, offering a performance of 64 fps.

Therefore, the proposed solution can process 33 times more video data volume in real-time than the software GPU accelerated implementation for the described testing conditions.

Resources	Used	Available	Percent
Slice LUT	100,058	230,400	43,43 %
LUT RAM	11,588	102,760	11,39 %
Flip-Flops	200,981	460,800	43,62 %
Block RAM	182	321	58,00 %
Ultra RAM	92	96	95,83 %
DSP Blocks	1,380	1,728	79,86 %

Table I
FPGA RESOURCES USED FOR VIDEO ANALYTIC SOC IMPLEMENTATION ON A XILINX ZYNQ ULTRASCALE+ XCZU7EV-2FFVC1156 DEVICE MPSOC DEVICE.

Table I summarizes the FPGA resources used in the implementation. The design comprises two 2x4096 DPU instances and one VCU unit in the reconfigurable section. The intensive use of Ultra-RAM memories resources (95%) and DSP blocks (1.380 blocks) is worth mentioning. The device Xilinx Zynq UltraScale+ XCZU7EV-2FFVC1156 device MPSoC used in this implementation is available in new generation COTS Edge Computing devices [27]. Thus, it is feasible to implement the techniques presented in this research in commercial equipment.

VI. CONCLUSION

This work has presented an AI Video analytic application implemented on an Edge-computing device. This device is capable of accelerating the inference of AI models and Video compression by dedicated hardware.

The architecture is designed to implement Video, Networking, and Deep Learning Processing functionalities on a reconfigurable System-on-Chip has been presented. An overview of the design tools and design flow followed to generate all software and hardware configuration has been covered.

This Edge Intelligence platform is currently in-service, providing the preliminary results for the targeted applications. The acceleration obtained offers latency times adequate to implement low-latency Video analytic applications required in critical sectors like Automotive or Aerospace&Defence.

Future work includes additional NN model testing, Time-Sensitive Networking communication capability, and sensor information enrichment integration on the SoC.

REFERENCES

[1] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, p. 58–67, jan 2017. [Online]. Available: https://doi.org/10.1109/MC.2017.3641638

[2] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and {Delay-Tolerance}," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 377–392.

[3] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 115–131.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[5] L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams *et al.*, "Recent advances in deep learning for speech research at microsoft," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 8604–8608.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[7] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2722–2730.

[8] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[9] M. L. Silva and J. C. Ferreira, "Support for partial run-time reconfiguration of platform FPGAs," *Journal of Systems Architecture*, no. 52, pp. 619–639, 2006.

[10] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[11] T. N. Sainath, B. Kingsbury, A.-r. Mohamed, G. E. Dahl, G. Saon, H. Soltau, T. Beran, A. Y. Aravkin, and B. Ramabhadran, "Improvements to deep convolutional neural networks for lvcsr," in *2013 IEEE workshop on automatic speech recognition and understanding*. IEEE, 2013, pp. 315–320.

[12] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[13] Y. Le Cun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard, "Handwritten digit recognition: Applications of neural network chips and automatic learning," *IEEE Communications Magazine*, vol. 27, no. 11, pp. 41–46, 1989.

[14] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[19] Y. Zhang, K. Sohn, R. Villegas, G. Pan, and H. Lee, "Improving object detection with deep convolutional networks via bayesian optimization and structured prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 249–258.

[20] J. Lee, J. Bang, and S.-I. Yang, "Object detection with sliding window in images including multiple similar objects," in *2017 international conference on information and communication technology convergence (ICTC)*. IEEE, 2017, pp. 803–806.

[21] C. Liu, Y. Tao, J. Liang, K. Li, and Y. Chen, "Object detection based on yolo network," in *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)*. IEEE, 2018, pp. 799–803.

[22] Y. Liu, "An improved faster r-cnn for object detection," in *2018 11th International Symposium on Computational Intelligence and Design (ISCID)*, vol. 2. IEEE, 2018, pp. 119–123.

[23] L. Zhang, L. Lin, X. Liang, and K. He, "Is faster r-cnn doing well for pedestrian detection?" in *European conference on computer vision*. Springer, 2016, pp. 443–457.

[24] L. Huang, Y. Yang, Y. Deng, and Y. Yu, "Densebox: Unifying landmark localization with end to end object detection," *arXiv preprint arXiv:1509.04874*, 2015.

[25] K. P. Seng, P. J. Lee, and L. M. Ang, "Embedded intelligence on fpga: Survey, applications and challenges," *Electronics*, vol. 10, no. 8, p. 895, 2021.

[26] More than 25 contributors, "Ai-model-zoo xilinx github," https://github.com/Xilinx/Vitis-AI/tree/master/models/AI-Model-Zoo.

[27] Relyum by SoCe, "RELY-MIL-SWITCH-ROUTER High-availability Military Switch Router," https://www.relyum.com/web/rely-mil-switch-router/, 2022.