Expanding Tail Layer Training Scope on FPGA with Data Augmentation

Yuki Takashima
Tokyo Institute of Technology, Japan
takashima.y.ad@m.titech.ac.jp

Akira Jinguji
Tokyo Institute of Technology, Japan

Ryota Kayanoma
Tokyo Institute of Technology, Japan

Hiroki Nakahara
Tohoku University, Japan

Abstract— The demand for deep learning has increased, and many accelerators have been proposed. Although they perform inference at high speed, many of them have problems in training. We present the "tail layer training" for a convolutional neural network (CNN). In this method, only the tail layer of the model is trained. Since the number of neurons in the output and classes must be the same for image classification, it is effective for retraining to count the number of classes. Accuracy loss is negligible when training only the tail layer with two added categories in CIFAR10. Even on large datasets such as ImageNet, underfitting of additional classes can be avoided by using data augmentation. And since only the tail layer is trained, fast computation using a CNN accelerator is possible. Therefore, lightweight learning on FPGAs is achieved. Our scheme can be applied to the all existing SoC-FPGA-based CNN accelerator.

I. Introduction

Deep learning models are widely used in a variety of fields, with image classification at the top of the list, for example Classification[1], Object Detection[2], Face Detection[3], Segmentation[4], Pose Estimation[5], Molecular Depth Estimation[6], Super Resolution[7], GAN: Generative Advisal Network[8], etc. Most of these applications are based on Convolutional Neural Network (CNN)[9].

CNNs need a massive number of parameters and computational complexity compared to existing machine learning models. Therefore, while they are suitable for training complex tasks with large amounts of data, they are also computationally demanding. Thus, CNN-specific accelerators have been developed and used in embedded systems that require high speed and low power consumption. However, many accelerators cannot train on the accelerator because they require pre-optimization (quantization and sparsification) and compilation of the trained model. In FPGAs, DPU (Deep learning Processor Unit) developed by Xilinx can be used for fast inference with learned models, but it does not support training. In contrast, embedded systems may require training on edge after design with additional functionality or updates. For example, face recognition has recently been developed as a seamless authentication method. Face recognition systems are expected to be used in various locations, so there may be some situations where the system is not connected to a network. In addition, the facial photographs needed to learn facial recognition are personal information. There is a risk in communicating such information over a network. Therefore, there is an advantage in completing the learning within the edge terminal. In this case, it is difficult to train a new face for authentication using only an edge terminal.

In this paper, we propose Tail Layer Training as a lightweight learning method on FPGA. In this method, additional hardware and training algorithms in a DPU, one of the CNN accelerators, and perform other training on the CPU. Typically, CNNs are trained using backpropagation. It calculates the error between the training data and the correct data for all layers and updates the parameters while controlling overfitting by the learning rate. Therefore, it requires many computer resources and time and is unsuitable for embedded systems. In contrast, adding or updating functionality in embedded systems is often limited, and the tasks are often the same with limited additional data. Fine-tuning is a method for fast convergence of CNN training. It replicates some of the parameters of the trained model to a part of the CNN model to be trained. Thus, making the trend of the target model closer to that of the trained model. For example, fine-tuning on ImageNet can converge in a short training epoch in image recognition tasks. In this paper, we target a classification task and propose to add a training function to the accelerator by making only the tail layer of the CNN independent. Only the tail layer is trained in software using an ARM processor on an FPGA, and the trained parameters are reflected in the hardware dedicated to the tail layer. In the classification task, the first part of the CNN is considered the feature extraction of the image. The last part is considered to be the encoding of

the extracted features into a class index that humans can understand. If this assumption is correct, the classification task can be re-trained by implementing a pre-trained model in the same domain with a DPU and training only the encoding part at the tail layer. This method can be applied in other models where labeling is based on features. It can also be applied to CNN accelerators other than DPU.

It was found that this tail layer learning tended to underfitting for additional classes. This effect is sufficiently small when small data sets are used and the proportion of additional classes is large. However, when using a large data set, the problem is that the ratio of additional classes to existing classes is so small that they are not adequately trained. Therefore, we solved this issue by using data augmentation.

The structure of this paper is as follows. Chapter 2 describes related research and tools. Chapter 3 describes the proposed method of tail layer training and data augmentation. Chapter 4 presents experiments and a discussion of the proposed method, and Chapter 5 concludes the paper.

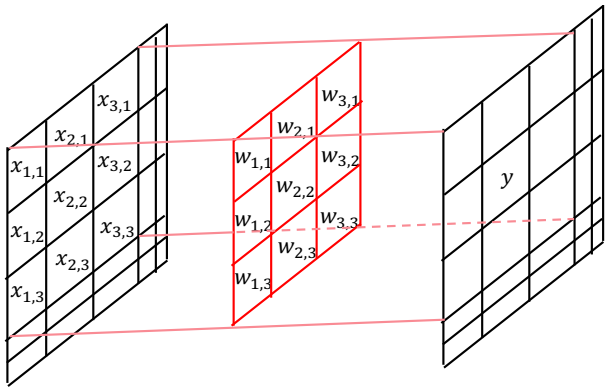

Fig. 1. Convolutional Layer Overview.

II. Related Work

A. CNN (Convolutional Neural Network)

A CNN (Convolutional Neural Network) is a deep learning model used for tasks such as image classification, which automatically extracts features from images by performing two-dimensional convolution. Typically, models used for image classification require that the number of neurons in the output layer matches the number of classification classes. Therefore, the CNN must be re-trained with all the data when adding a class.

CNN models such as VGG[10], ResNet[11], and MobileNetV2[12] have already been proposed, which use a convolutional layer, a pooling layer, and a fully connected layer. The convolutional layer (conv layer) has a structure like Fig. 1. The output is determined by the sum of the product of the weights assigned to the convolution window and the input, and this process is applied by shifting the window. It has the effect of extracting a variety of features. The pooling layer has the effect of compressing the size of the input by obtaining representative values. There are two methods for selecting expected values: max pooling (selecting the maximum value within a specific range) and average pooling (setting the average value). These are weightless because they are determined from the input values only. In the fully connected layer (fc layer), the output is determined from the sum of the product of the inputs and weights. Still, there are as many weights as the product of the number of input dimensions and the number of output dimensions. In VGG, ResNet and MobileNetV2, the all-connected layer is used at the end of the model.

B. Data Augmentation

Data augmentation is a technique that can virtually increase the amount of data by adding preprocessing to the data set. Typical methods include center crop and resize. In general, data augmentation is expected to be effective in learning and preventing overfitting even with a small amount of data.

III. Proposed Method

Fig. 2. Tail Layer (fc3) Training in VGG16.

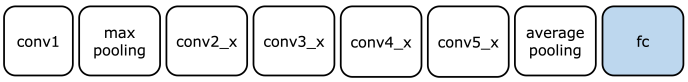

Fig. 3. Tail Layer (fc) Training in ResNet50.

A. Tail Layer Training

The number of neurons in the output layer of the CNN must match the number of classes to be classified. Therefore, when additional classes are desired, the model must be changed to one with an increased number of output neurons and retrained.

Assuming that features can be extracted while the input passes through many layers of the CNN, it is possible to achieve sufficient accuracy by simply retraining the mapping between features and labels in the tail layer. This training is defined as tail layer training. In this paper, we propose two methods for training the tail layer: one is to use a randomly determined initial value, and the other is to train only the additional classes without updating the weights of the existing classes.

Fig. 2 shows an example of tail layer training when VGG16 is used. Fig. 3 shows an example of tail layer training when ResNet50 is used. The layers shown in white in Fig. 2 and Fig. 3 are the layers whose weights are not updated in the tail layer training. Therefore, only

inference is performed on the input images. In contrast, the tail layer shown in blue is the fully connected layer (fc layer) for both VGG16 and ResNet50.

The inference-only layer reuses the same weights. Therefore, using pre-trained weights and compiling only the portions of VGG and ResNet without the tail layer can be computed faster using DPUs. Since this part of the model represents a large portion of the model, it can significantly reduce the computational cost compared to training the entire model. For example, VGG16 has 134M parameters. Of these, the input for the tail layer has 4,096 dimensions and output has 1000 dimentions if we use imagenet. Therefore, the number of parameters in the tail layer $param_{\text{fin}}$ can be said to be

$$\begin{aligned} param_{\text{fin}} &= 4096 \times 1000 + 1000 && (1) \\ &\simeq 0.41\text{M}. && (2) \end{aligned}$$

Only about 0.3% of the total parameters need to be trained. In addition, the output of the layers whose weights do not change as the training progresses. The training process can be made even faster in the second and subsequent epochs by saving the output at the DPU when training with the same image.

B. Back Propagation for Tail Layer Training

Neural networks, including CNNs, cannot make appropriate inferences about the input unless the weights have appropriate values, and the expected output and the model output will not match. Therefore, it is necessary to update the weights, one of which is the back propagation method. Let $\boldsymbol{y}$ be the expected output and $\hat{\boldsymbol{y}}$ be the model output, and let E be the error function $E = |\boldsymbol{y} - \hat{\boldsymbol{y}}|^2$.

The weights are changed to minimize E because E should be adjusted to be zero and $E \geq 0$ is always true in order to make the model output the expected output.To minimize E, the difference Δw for modifying each weight w is updated according to

$$\Delta w = -\eta \frac{\partial E}{\partial w} \tag{3}$$

using an appropriate learning rate η.

Let z_i^{l-1} be the input of the i-th neuron in the $l-1$ layer and x_i^{l-1} be its output. Also, let $w_{i,j}^{l-1}$ be the weight from the i-th neuron in layer $l-1$ to the j-th neuron in layer l. When the activation function σ is

$$x_j^l = \sigma(\sum_i z_i^{l-1} w_{i,j}^{l-1}) \tag{4}$$

, then the right side of Eq.(3) is

$$-\eta \frac{\partial E}{\partial w_{i,j}^l} = -\eta \frac{\partial E}{\partial x_j^{l+1}} \sigma'(z_j^{l+1}) x_j^l \tag{5}$$

.

In the tail layer, for $\frac{\partial E}{\partial x_j^{l+1}}$ in Eq.(5), when the number of layers of the CNN is n, $x_j^n = \hat{y}_j$. By $E = |\boldsymbol{y} - \hat{\boldsymbol{y}}|^2$, it can calculated as

$$\begin{aligned} \Delta w_{i,j}^l &= -\eta \frac{\partial E}{\partial w_{i,j}^l} && (6) \\ &= -\eta \frac{\partial E}{\partial x_j^{l+1}} \sigma'(z_j^{l+1}) x_j^l && (7) \\ &= -\eta \frac{\partial |\boldsymbol{y} - \hat{\boldsymbol{y}}|^2}{\partial \hat{y}_j} \sigma'(z_j^{l+1}) x_j^l && (8) \\ &= 2\eta (y_j - \hat{y}_j) \sigma'(z_j^{l+1}) x_j^l && (9) \end{aligned}$$

. This shows that only the tail layer can be trained.

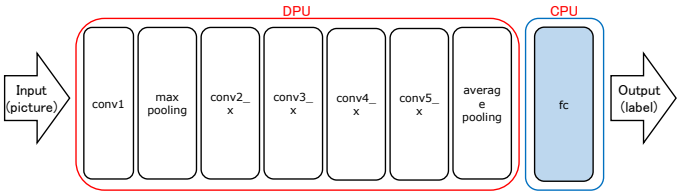

Fig. 4. Hardware Configuration of Tail Layer Training.

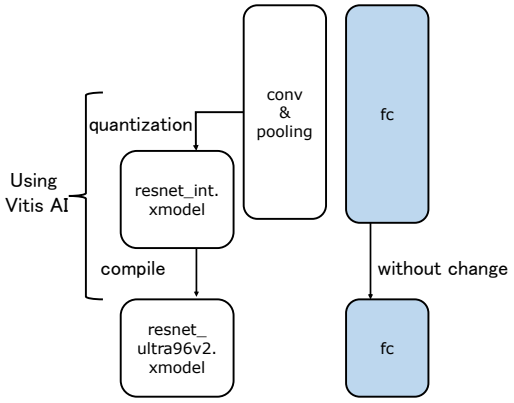

Fig. 5. Model Creation Flow.

C. Implemented Platforms

The hardware configuration for tail-layer training is shown in Fig. 4, where models compiled from ResNet except for the tail layer are computed on a DPU. On the other hand, the tail layer for training is computed on the CPU.

The model creation method is shown in Fig. 5. Except for the tail layer of ResNet, it needs to be converted into a format executable by DPU. xmodel files are generated by compiling after quantization using Vitis AI and then transferred to the board to be read. The system can be used for the following purposes. The tail layer can be loaded with weights trained in advance.

D. Data Augmentation for large dataset

For small datasets such as CIFAR10, sufficient accuracy has been confirmed even with Tail Layer Training.

On the other hand, when using a large dataset such as ImageNet, the proportion of training data for additional classes is extremely small, resulting in underfitting and loss of accuracy. Therefore, we propose a method to improve the ratio by applying data augmentation only to the additional classes. Some well-known augmentation methods are center cropping, resizing, and flipping. Preliminary experiments were conducted on center cropping and resizing, where the scale factor can be set continuously, and it was found that resizing in particular works effectively. Consequently, we decided to perform resizing of various magnifications only on the additional classes without augmentation on the existing classes to change their proportions in the data set.

IV. Experiments and Discussions

A. Experimental Environment

Python 3.8.8, PyTorch 1.7.1, and Torchvision 0.8.2 were used to train the model. The DPU was created using Vitis-AI 2021.1 and implemented on a ZCU104 board.

TABLE I
Classification accuracy of CIFAR10 using VGG16.

domain	8 classifications	Tail layer training	existing method
ALL	91%	86%	89%
plane	94%	71%	92%
car	98%	84%	93%
bird	86%	87%	85%
cat	84%	83%	88%
deer	93%	91%	91%
dog	83%	86%	88%
frog	98%	94%	92%
horse	91%	92%	93%
ship	−−	98%	97%
truck	−−	83%	92%

TABLE II
Classification accuracy of CIFAR10 using ResNet18.

domain	8 classifications	Tail layer training	existing method
ALL	95%	94%	95%
plane	98%	96%	96%
car	99%	93%	98%
bird	94%	93%	93%
cat	90%	88%	98%
deer	95%	94%	96%
dog	92%	92%	91%
frog	98%	98%	97%
horse	99%	96%	97%
ship	−−	96%	96%
truck	−−	94%	97%

B. Accuracy of Tail Layer Training Using CIFAR10

The results of the tail layer training using VGG16 are shown in Table I. Eight classes of VGG16, excluding ship and truck, were trained using the weights learned in ImageNet as initial values, resulting in a 91% correct response rate. Using this 8-class classification model as a base, we trained only the tail layer using all CIFAR10 images and obtained an overall accuracy rate of 86%. On the other hand, VGG16, in which the entire model was trained with CIFAR10, had a correct response rate of 89%. Tail layer training shows only three-point accuracy degradation compared to existing methods.

Similarly, the results of tail layer training using ResNet18, ResNet50, and MobileNetV2 are shown in Table II, Table III, and Table IV, respectively. These models achieve recognition accuracies of 94% or better. The results with Tail Layer Training showed only a 1% degradation in accuracy compared to existing methods. This sug-

TABLE III
Classification accuracy of CIFAR10 using ResNet50.

domain	8 classifications	Tail layer training	existing method
ALL	96%	96%	96%
plane	99%	95%	97%
car	99%	97%	97%
bird	94%	96%	95%
cat	93%	93%	91%
deer	97%	98%	97%
dog	94%	94%	95%
frog	97%	97%	99%
horse	98%	98%	97%
ship	−−	96%	98%
truck	−−	95%	97%

TABLE IV
Classification Accuracy of CIFAR10 Using MobileNetV2.

domain	8 classifications	Tail layer training	existing method
ALL	95%	94%	95%
plane	98%	93%	97%
car	99%	94%	97%
bird	94%	93%	96%
cat	90%	89%	89%
deer	96%	96%	96%
dog	90%	92%	90%
frog	98%	97%	98%
horse	97%	96%	96%
ship	−−	95%	96%
truck	−−	91%	97%

TABLE V
Classification accuracy of CIFAR10 when pretrains with ResNet50 are used as is.

domain	8 classifications	Tail layer training	existing method
ALL	85%	83%	96%
plane	91%	79%	97%
car	96%	89%	97%
bird	76%	77%	95%
cat	75%	75%	91%
deer	83%	75%	97%
dog	85%	84%	95%
frog	90%	88%	99%
horse	88%	86%	97%
ship	−−	91%	98%
truck	−−	89%	97%

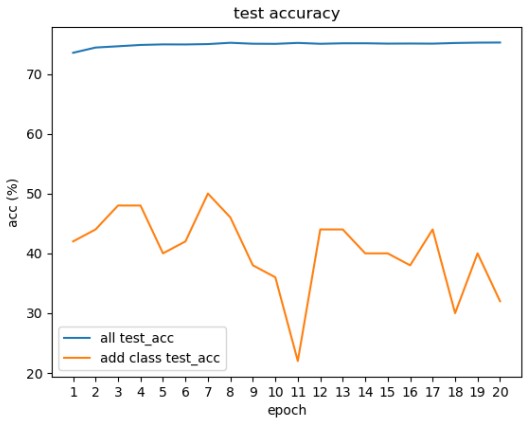

Fig. 6. The Process of Tail Layer Training

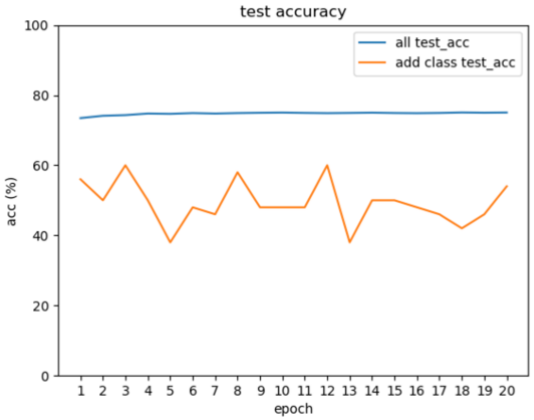

Fig. 7. Learning curve with double augmentation of data

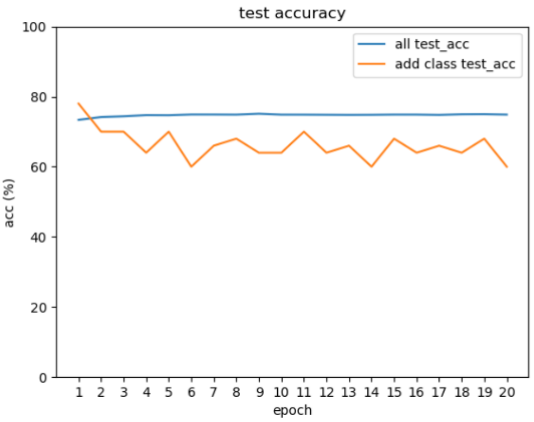

Fig. 8. Learning curve with 10x augmentation of data

gests that the higher the accuracy of the original model, the less the accuracy degradation may be when Tail Layer Training is performed.

The results of the same tail layer training experiment are shown in Table V. It decreases in accuracy for all items compared to Table III, which was tested using the same model and data set. It may suggest that when extracting features in layers other than the tail layer, the training results from the existing domain are effective even in the new domain in the additional training.

C. *Accuracy of Tail Layer Training Using ImageNet*

ResNet50 trained with 999 classes, excluding n15075141 (toilet tissue), which corresponds to the last label in ImageNet, yielded a correct response rate of 74%. Using this model as a base, we trained only the tail layer from initial values given by random numbers using all images in ImageNet. We obtained a correct overall rate of 75% (Fig. 6). On the other hand, the added classes only resulted in a 32% correct response rate. In this experiment, the class with the highest error rate was n03887697 (paper towel), which classified 24%. This suggests that the distinction between similar images remains ambiguous and underfitting is occurring.

D. *Data Augmentation for Tail Layer Training*

To improve underfitting, data augmentation was performed on the additional data. While the train data for the existing classes remained unchanged, the number of images for the additional classes could be increased to any number by adding images resized to various magnifications. Specifically, the following settings were used, we experimented with the following settings.

- Add an image with one side multiplied by $\sqrt{2}$ to double the total amount
- Add an image with one side multiplied by 1.1, 1.2, ..., 1.9 to increase the total amount by 10
- Add an image with 1.1, 1.2, ..., 2.9 times one side to increase the total amount by 20

The results are shown in Fig. 7, 8, and 9, respectively. The overall accuracy in all figures was a finally 75%. The final accuracy for the additional classes was 54% in Fig. 7, 60% in Fig. 8, and 74% in Fig. 9. Compared to Fig. 6, the accuracy of the additional class was improved in all cases, and the overall accuracy was not significantly affected. In the case of Fig. 7, the accuracy of the additional class is clearly lower than the overall accuracy. However, as the total amount of additional data is increased by augmentation, the accuracy approaches the overall accuracy. In particular, in Fig. 9, the two results show almost the same learning curve, which is considered to be the ideal learning state in which neither underfitting nor overfitting occurs.

E. *DPU Modeling and Inference*

The ResNet50 model trained with PyTorch using Vitis-AI can be quantized and compiled for ZCU104 to create xmodel files. The tutorial provided on DPU on

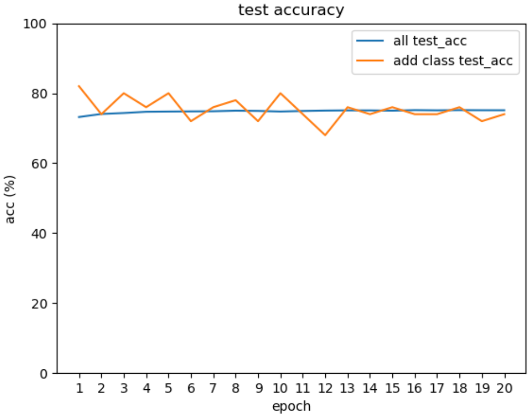

Fig. 9. Learning curve with 20x augmentation of data

PYNQ[13] includes inference using ResNet50, with a performance of 26.02 FPS. On the other hand, the remainder of ResNet50, excluding the tail fully connected layer, was inferred using DPUs. Only the tail layer was inferred using a model that can be trained with PyTorch, resulting in a performance of 18.27 FPS. Although the tail fully connected layer is implemented with PyTorch on a CPU, the inference speed is sufficient.

F. *Increased Learning Costs When Using Data Augmentation*

To find out how much learning cost is required by data augmentation, we performed the following experiment: For training on ZCU104, we prepared a dataset that is 1/10 of ImageNet and trained with a batch size of 1, which took 12139 seconds per epoch. For the same dataset, training with augmentation to increase the amount of additional classes by a factor of 20 took 12260 seconds per epoch. Thus, the augmentation is expected to increase the training cost by 1%.

V. Conclusion

We trained only the tail layer on the CPU/DPU hybrid system. Using a CNN accelerator allowed faster inference. However, the almost weights were not updated. It separates the tail layer from the CNN accelerator while the CPU is used for training, such as adding classes. Although tail layer training on large datasets tends to underfitting for additional classes, this is resolved by data augmentation. We showed that our CPU/DPU hybrid system only had a few performance degradation with a training functionality. Our scheme can be applied to the existing SoC-FPGA-based CNN accelerator.

VI. Acknowledgments

This research is partly supported by the Grants in Aid for Scientistic Research of JSPS.

References

[1] PyTorch Research Models. `https://pytorch.org/hub/research-models`.

[2] Yolo. `https://pjreddie.com/darknet/yolo/`.

[3] OpenFace. `https://github.com/cmusatyalab/openface`.

[4] DeepLab. `https://github.com/tensorflow/models/tree/master/research/deeplab`.

[5] OpenPose. `https://github.com/CMU-Perceptual-Computing-Lab/openpose`.

[6] PackNet-SfM. `https://github.com/TRI-ML/packnet-sfm`.

[7] Y. Zhang et al. "Image Super-Resolution Using Very Deep Residual Channel Attention Networks". *ECCV*, 2018.

[8] I. Goodfellow et al. "Generative adversarial nets". *in Proc. Int. Conf. Neural Inf. Process.Syst.Syst.*, 2014.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". *Proceedings of the IEEE*, november 1998.

[10] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". `https://arxiv.org/abs/1409.1556v1`.

[11] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". `https://arxiv.org/abs/1512.03385v1`.

[12] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". `https://arxiv.org/abs/1801.04381v3`.

[13] *"DPU on PYNQ"*. `https://github.com/Xilinx/DPU-PYNQ`.