Dandelion: Boosting DNN Usability Under Dataset Scarcity

Xiangru Chen, *Student Member, IEEE*,
Jiaqi Zhang, *Student Member, IEEE*, and Sandip Ray, *Senior Member, IEEE*

Abstract—The development of deep neural network (DNN) has provided transformative impacts on many fields, including computer vision and video recognition. However, the impact is limited by the need for large, labeled datasets to enable effective training. To address this fundamental problem, we propose a novel inter-network system (Dandelion), providing architecture support (Dandelion-architecture) for data augmentation that trains DNNs with rare images generated by the generative adversarial network (GAN) with orthogonal attributes modified (Dandelion-function. The approach can account for the latency requirement and resource limitation of target applications by exploiting data and computation reuses between the two networks; this amortizes the impact of bottleneck brought by GAN and facilitates design of inter-network accelerator. Moreover, we show how to implement two-network design on 3D architecture to further enhance the accelerator. Our results show that with the generated images, DNN yields 13.6% - 37.5% improvement on accuracy, depending on the data scarcity level. Our architecture achieves at least 30% speedup compared with the baseline while 40% of the overhead brought by the incorporation of GAN is reduced in our design compared with ScaleDeep, and 26.3% of performance improvement over TETRIS.

Index Terms—Neural network training, inter-network accelerator, data augmentation, edge device

1 Introduction

In the recent years DNN has established itself as an efficient and accurate tool for image and video recognition. For instance, the auto-driving feature of Tesla [1] is based on the classification for the road condition using neural network and Google's image engine trains its network to support image searching. There has been successful application of DNN in diverse and critical areas including medical [2], cyber security [3], film preproduction [4] and gaming [5], performing tasks like object locating, face recognition, target partition and action tracking.

However, a critical obstacle to usability of DNNs on a number of applications is the scarcity of data. The high accuracy of DNNs relies on training datasets that effectively reflects the variations and distributions encountered in field. Unfortunately, for many applications that can benefit from DNN, it is challenging to collect sufficient training data reach the desired accuracy within a short time or a cost budget. For instance, the symptom images required by DNN training for automated COVID-19 diagnosis [6] can only be collected as time elapses, on account of a long detection period. Enabling faster development of DNN's with sufficient accuracy could have significant impact on the response to the pandemic, with transformative impact to public health and social economy [7]. Another example is the recognition of an escaped prisoner or criminal, which requires a training dataset including pictures of the targeted criminal that are difficult to obtain quickly even in the big data era.

To enrich the training dataset, prior work relies on image augmentation methods, e.g., cropping and rotation [8], [9]. GAN (Generative Adversarial Network) has also been leveraged to generate new images from random noise [10], [11], [12]. However, all existing approaches to our knowledge only utilize information from the scarce target data domain. Although network training benefits from these attempts, the performance improvement is limited due to lack of additional real-world information. Recent research [8] developed StarGAN to perform multi-domain image translation which can add multiple desired features learned from orthogonal real-world dataset, tremendously enhancing the ability of image generation. Nevertheless, data scarcity remains the bottleneck.

The key result of this paper is *Dandelion*, an accelerator for the combination of two networks, to compensate for the scarce DNN training data by feeding GAN-generated data into it. We demonstrate how this can efficiently solve the training dilemma on degraded DNN accuracy caused by scarce training data. Dandelion is specifically developed exploit data augmentation ability of StarGAN [13].

A critical challenge in realizing this vision is the considerable hardware overhead added inevitably to the accelerator by the inclusion of GAN, which can lead to lower performance and energy efficiency of the entire system. This problem is getting particularly acute within the emergent paradigm of edge intelligence, where the networks must be deployed on resource constrained edge devices. Prior design [7], [8] on single-network accelerator support network training with advanced dataflow and architecture. However, the data dependency between networks may cause either considerable idleness or memory consumption.

• *The authors are with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611 USA. E-mail: {cxr1994816, jiaqizhang}@ufl.edu, sandip@ece.ufl.edu.*

Manuscript received 15 August 2021; revised 11 November 2021; accepted 14 November 2021. Date of publication 2 December 2021; date of current version 8 September 2022.
(Corresponding author: Xiangru Chen.)
Recommended for acceptance by C. Li.
Digital Object Identifier no. 10.1109/TC.2021.3132170

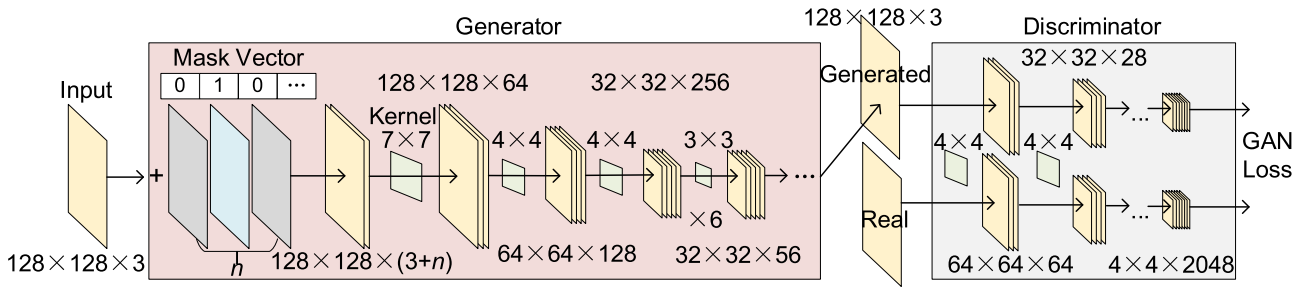


Fig. 1. StarGAN computation structure.

Furthermore, the various sizes of kernels and feature maps result in resources underutilization, bringing about less benefits when employing single-network accelerator to handle two networks altogether.

To reduce this overhead, we propose the inter-network accelerator, *Dandelion-arch*, with four techniques to maximally exploit the reuse opportunities between the two networks. First, we leverage a shared buffer to reuse the input images for the two networks, and the generated images are consumed on-chip to save bandwidth. In addition, the proposed architecture makes full utilization of the computing resources between networks by dynamic dataflow reconfiguration and processing element reallocation. Furthermore, we propose mapping policies that can adapt our architecture to any two neural networks. Finally, we extend our accelerator to 3D implementation for even higher efficiency.

In summary, the paper makes the following important contributions:

1. We propose a new *inter-network* system: *Dandelion*, which validate the accuracy improvement of DNN training by changing orthogonal attributes in light of dataset scarcity (*Dandelion-function*) with 30% performance improvement brought by inter-network accelerator (*Dandelion-arch*).
2. We utilize the novel *reuse opportunities lying on the combination of networks* in our first inter-network accelerator architecture design and further extend it to support the inclusion of various network as a principle of inter-network accelerator design.
3. We show how to implement the two-network system using 3D architecture with *optimized cross-vault partitioning and buffer-bypassing strategy*.

The rest of this paper is organized as follows: Section 2 provides the background on GAN and DNN training as well as our motivation for *Dandelion*. Section 3 introduces the detailed function design of *Dandelion*. Section 4 proposes the architecture support for *Dandelion* and 3D integration. Sections 5 and 6 elaborates our evaluation methodology and analyzes the results, respectively. The related work is discussed in Sections 7 and 8 concludes the paper.

2 Background

DNN Training: DNN typically consists of convolutional layers (i.e., hidden layer performing convolution operations), pooling layers and fully connected layers. As a supervised-learning algorithm, DNN is trained by labeled training data to adjust its parameters before deployed for inference on the unknown samples from real-world applications. In the training phase, there are three stages for each input sample organized by the gradient decent algorithm, i.e., forward propagation (FP), backward propagation (BP), and weight update (WU). In forward propagation, the inputs are processed by kernels to extract features as intermediate data of each layer. Once the output of one layer is generated, the intermediate data is stored for the next layer and the following training stages. The output of the last layer is the prediction vector which indicates the network's classification of input. The prediction vector is compared with the input label to calculate the loss. Then, the loss is used in backward propagation to generate the error of each layer, which contains the information to update the kernel parameters. Finally, in weight update stage, the errors are convolved with the intermediate data to produce the weight updates.

GAN: GAN is proposed as a feasible solution to unsupervised learning. As indicated in Fig. 1, it is composed of two components. The generator tries to learn the features of real images and embeds them into the input (a random vector or a real image) to produce new images. The discriminator aims to distinguish generated images from the real one. During the training phase, the outputs of the discriminator by processing fake images along with real images are computed in a subtraction manner to form the loss which indicates the difference between them. The generator and discriminator are trained adversarially against each other, i.e., the generator intends to reduce the loss while the discriminator attempts to enlarge it. When the training converges, the discriminator achieves the highest accuracy in classifying images into real and generated data while the generator can generate images that resemble the real ones to the most extent. During the inference phase, for each layer of the generator, the input is convolved with kernels. Simultaneously the image information already learnt by the kernels is reflected to the feature map to generate human-recognizable images. In terms of the computation, the GAN layers also perform convolution and fully connection operations on each input.

StarGAN: StarGAN [13] is a state-of-the-art GAN that provides unique ability for data augmentation compared to conventional GAN,. In particular, StarGAN can learn multiple attributes in its generator and selectively apply them to the input. The red area in Fig. 1 shows the inference of the generator of StarGAN. Specifically, StarGAN utilizes a mask vector to control the output attributes. For each element in the vector, there will be an additional input channel with the same 2D dimension as image. The value (true or false) of each vector element corresponds to different additional

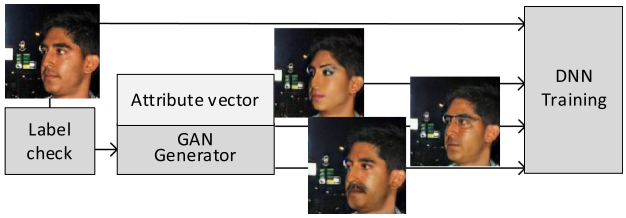
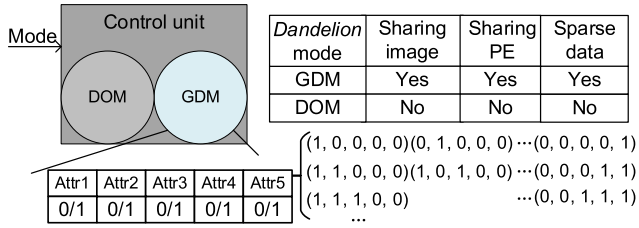


Fig. 2. Overview function of *Dandelion*. GAN changes the attributes of input image and provides DNN with populated dataset.



Dandelion mode	Sharing image	Sharing PE	Sparse data
GDM	Yes	Yes	Yes
DOM	No	No	No

Fig. 3. Two modes of *Dandelion*. The table indicates different characteristics of two modes.

feature maps. In the inference phase, the input with certain attribute information is convolved by a series of kernels to 32×32×256 and processed several times under this size to apply attribute modification. After this, the data passes through the following convolution layers similarly, to be restored to the size of the input and serves as generated image. Note that since *Dandelion* relies on the data augmentation function, we only deploy a generator which is readily trained on an orthogonal data domain and perform inference on it.

3 *DANDELION* FUNCTIONALITY

3.1 High-Level Overview

Dandelion connects the generator of the GAN with the DNN. The output of GAN generator is forwarded to DNN as inputs, as illustrated in Fig. 2. For our work, we found StarGAN [8] the best fit because of its ability to change multiple orthogonal attributes under the control of pre-defined state machine. In the following, we use a face recognition example to explain *Dandelion* functionality.

Consider training the DNN to recognize different faces but assume that the labeled images of a specific face are too scarce to be appropriately learned. To enrich this dataset, the GAN generator works as a populating tool to increase the number and diversity of the certain face.

The role of StarGAN in our design is to introduce additional real-world information to the input by only changing the subordinate attributes while preserving the original primary characteristics. This could simulate the condition of training with real dataset. The attributes can be combined and permutated in a customized manner in image generation. When training the GAN to learn different attributes, although the target dataset is scarce, it is still possible to use another dataset with abundant samples and orthogonal attributes. For example, in our experiments, we utilize CelebA as our abundant dataset to train StarGAN and let it learn inessential face attributes like makeup, moustache and glasses. Then, StarGAN can augment the Pubfig dataset that classifies people by their names by adding certain features to the basic face images, as shown in Fig. 2. This populating function can be applied to all the scarce classes. For those classes with plentiful samples, images are sent to the DNN directly. This way, the DNN can learn from an adequate and balanced dataset which alleviates the problem of overfitting.

3.2 Operation Modes of *Dandelion*

Since the GAN only generates data for scarce classes as mentioned in Section 3.1, *Dandelion* is compatible with two modes dealing with samples from the scarce and abundant classes respectively, which are shown in Fig. 3.

DNN-Only-Mode (DOM) is designed for images from the abundant class, where there are sufficient samples for training and GAN augmentation is unnecessary. The images are loaded from the memory into DNN directly for training.

GAN + DNN Mode (GDM) applies to images from the scarce class that need to be populated by GAN. The pre-trained StarGAN generates images with different features by setting the attribute mask vector introduced in 2.1. The attribute bits can be set one at a time for images with only one attribute modified or in combination to generate more complicated faces. In practice, the number of permutations is adjusted by the ratio of scarcity of the class to be populated. The max number of generated images is $\sum_{r=1}^{r=n} C_n^r$ when the length of the attribute mask vector is n.

During the first epoch of DNN training, the DOM and GDM modes are dynamically alternated depending on the class of the input image in run-time. If the input image belongs to scarce data, the mode is set to GDM, and GAN starts to generate images in the same speed of DNN training controlled by resources allocation. In another case, DNN takes all resources to reach highest speed. Then in the following epochs, all generated images have already been stored in off-chip memory, so the DNN iterates the populated dataset with DOM mode statically on.

3.3 Network Structure for *Dandelion*

As is explained in Section 3.1, we select StarGAN for the GAN component of *Dandelion*. For the DNN model, the choice can be more flexible. *Dandelion* can be utilized to facilitate the training of a variety of DNNs by adjusting architecture configuration. As an example, consider using SqueezeNet [18], a popular lightweight DNN with high accuracy, suitable to be deployed for the mobile and edge applications. SqueezeNet is a variation of AlexNet where most of the convolution layers are replaced with the special fire layers which also helps us explain the following architecture with this complex configuration. Fig. 4 shows the structure of the fire layer. It is composed of two operations, squeeze and extend. In the squeeze operation, the number of input channels is reduced by 1×1 kernel to cut down the parameters required by the following 3×3 convolution. The output of squeeze operation is then convolved by kernels with different sizes in the extend operation, the results of which are concatenated to form the total output of the fire layer.

4 ARCHITECTURE DESIGN OF *DANDELION*

Although GAN is an effective augmentation tool for DNN training on scarce dataset, it also aggravates the problem of limitation in hardware resource and energy consumption in

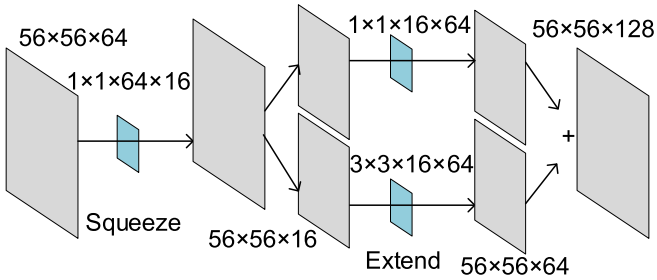


Fig. 4. One fire layer of SqueezeNet.

DNN accelerators, especially in mobile and edge ends. To reduce this overhead, we propose a novel architecture for *Dandelion* to maximally exploit the interplay between two networks and make full use of the resources. Note that our idea of inter-network acceleration works for any two networks. We take SqueezeNet and StarGAN to clarify the detailed configuration. The detailed multi-network mapping policy is analyzed later in 4.3.

4.1 Architecture Overview

Our architecture is a FPGA-based system depicted in Fig. 5, containing off-chip DRAM, on-chip BRAM, control unit, address management and processing elements (PE), which are connected through a bus. At training time, the input data is read from DRAM and the mode is signaled based on its label and the scarcity of the class. Then the control unit sets the corresponding state machine and registers including the weight loading signal, computation progress signal and PE allocation vector. Some of them are sent to address management module to calculate the addresses for weight buffer and data/error buffer in different training phase and the others are sent to PE channels to control their computation pattern. The detailed architecture which is designed for all 3 training phases and setup parameters will be discussed in the rest of this section and Section 5, respectively.

We notice that between the two networks, the inputs can be shared to avoid being sent to the off-chip memory. Besides, the computing resources are compatible with the two networks through special dataflow support and configuration. Therefore, we develop two kinds of reuses in our architecture, namely data reuse and PE reuse. We also discuss several important trade-offs in these two reuses that are important in determining the performance and energy efficiency. Although the resourse reuse has been discussed in previous works, none of them focus on that between two networks.

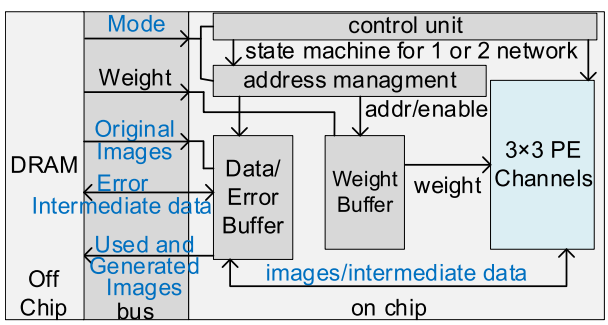


Fig. 5. An overview of *Dandelion* architecture.

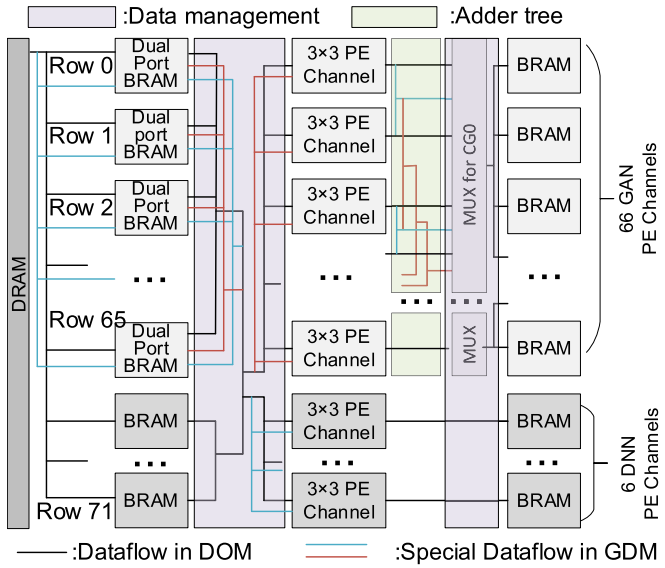


Fig. 6. Data management structure.

Considering the high bandwidth (3456 bits per clock cycle) and storage requirement of loading GAN during DNN training, we realize that 3D architecture is especially suitable to further promote the performance of our design. Therefore, we extend our design to explore the 3D opportunities and propose a novel partitioning method for 3D vaults.

4.2 Data Reuse

In Dandelion, to save the footprint and bandwidth of the memory resource, the DNN and GAN share the on-chip dual-port BRAM for the inputs as shown in the left half of Fig. 6. Besides, to reduce duplicate storage of the intermediate data and output muxings among PE channels, we use constrained output management within one channel group to be discussed in 4.3.

Under different operation modes, the pattern of data storage varies to support different dataflows with less memory access and energy consuming. In DOM, the data is separated into feature maps and each feature map is stored in one PE channel's buffer. During computation, one feature map is fetched and broadcast to all PE channels at a time. In GDM, the buffers are shared by the data of GAN and DNN. At this time, the input is separated into rows. Different rows are stored into buffers of different PE channels. Then, they are used for both DNN training and GAN inference simultaneously but read from different ports. The details will be explained in the PE combination of Section 4.3.

In GDM, the three stages of DNN training are pipelined with the generation of image of GAN, which is achieved by a precise resource allocation to be introduced in Section 4.3. In this pipeline, the DNN keeps consuming the generated image and the GAN keeps generating them at the same pace. This effectively avoids the time- and energy- consuming accesses to the off-chip memory. After processed by the DNN, the generated images are sent to off-chip memory for next epoch.

4.3 PE Reuse

Since our system switches between the DOM and GDM modes, it is wasteful to assign fixed computation resource

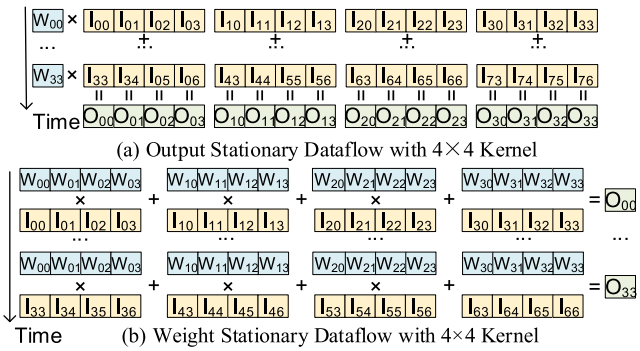


Fig. 7. Different dataflow types.

to each network, which will result in low utilization. Therefore, we propose a design to reconfigure the PEs so that they can be reused in different time slots by different networks.

PE channels and dataflow: We first discuss the possible dataflow types because it determines the basic setting of computation unit pattern which has great impact on the PE reuse efficiency. There are three kinds of dataflows focusing on different kinds of data reuse, weight stationary, output stationery and row stationary.

Output stationary (Fig. 7a) is to keep a certain output pixel in one PE while broadcasting one input in each cycle until the computation of the output completes. Weight stationary (Fig. 7b) means that all the weights in each kernel are unrolled and stationary reused until all the pixels of one feature map are calculated. Different from output stationary, the PE pattern in weight stationary is determined by the kernel size, e.g., if the kernel size is 3×3, there are 3×3 PEs in one PE channel. Row stationary [10] handles computation in row pattern. Input feature maps and kernels are separated into rows and fed into PE array, which reduces under-utilization by half.

Although output stationary is good at output reuse, it has limitation on the output size. For example, the outputs of StarGAN and SqueezeNet show significant difference in their sizes, so the output stationary dataflow will lead to a huge waste in the computation resource. Therefore, we consider weight stationary dataflow in our design for FP and BP (output stationary for WU because the output size equals to kernel size). The difference between FP and BP lies on the input data (images or errors). As most of the kernels share the same size in both two networks, the weight stationary dataflow where the kernels are unrolled manifest higher efficiency. As most of the kernels in these two networks are 3×3 and the rest of them can be easily mapped to 3×3, we choose 3×3 as the size of our PE channel. There is a total of 72 such PE channels, which is determined by the on-chip resources in our implementation.

PE partitioning: As mentioned in Section 4.2, when two networks operate together, the speed of image generation should be equal to the speed of DNN training so that the computation unit will not be idle. The total cycles for StarGAN generator inference and SqueezeNet training stages are analyzed to help decide the resource partition ratio between GAN and DNN to achieve the same computation latency. During training, the PEs are allocated at the channel level based on the mode. In GDM, 66 PE channels are allocated to GAN computation and 6 PE channels for DNN training (11:1). In DOM, with abundant data, no image generation is involved, so all the PE channels are allocated to DNN to maximize performance.

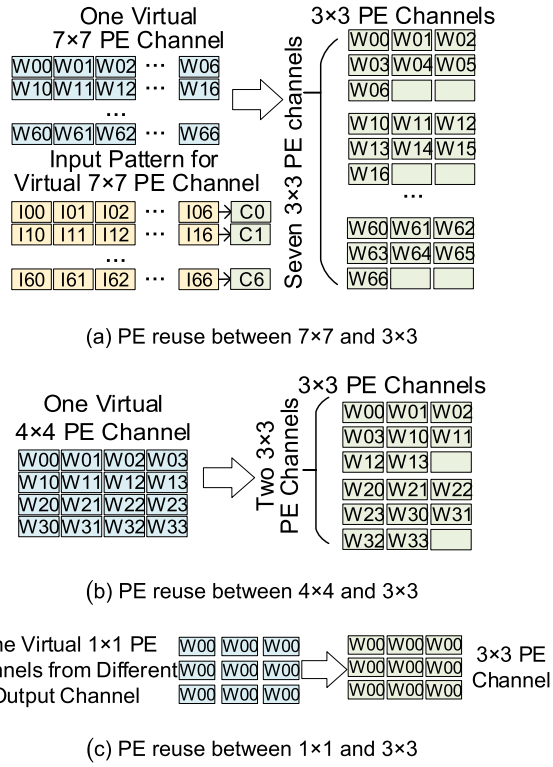
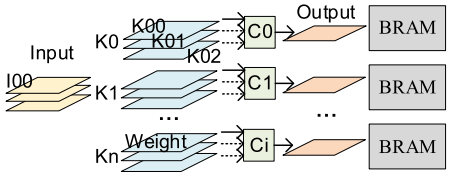


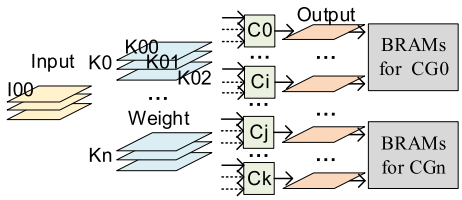
Fig. 8. PE reuse. C0 represents the first 3×3 PE channel in each PE group.

PE combination: Although most of the kernel size in StarGAN and SqueezeNet are 3×3, there still exist 7×7, 4×4, and 1×1 kernels. To enable full utilization of the computation resources when mapping all these kernels to 3×3 PE channels, we propose a novel PE combination technique, shown in Fig. 8. In PE combination, several PE channels are grouped together as a virtual channel. For example, one 7×7 kernel can be mapped to seven 3×3 channels as a channel group (Fig. 8a). Each 3×3 PE channel holds one row of the 7×7 kernel. Here, the partial results should be accumulated among these seven PE channels. This technique also applies when mapping 4×4 kernels by mapping two rows of kernels to each PE channel (Fig. 8b). Unlike Im2col [11] which is designed to reduce the memory latency of GPU but causes unpredictable storage pattern if applied to FPGA, our *Dandelion-arch* optimizes the dataflow to greatly improve the utilization. Compared with the simple kernel tiling, our proposed PE comination significantly raises the utilization (from 60% for 7×7 and 44% for 4×4 to 78% and 89% respectively).

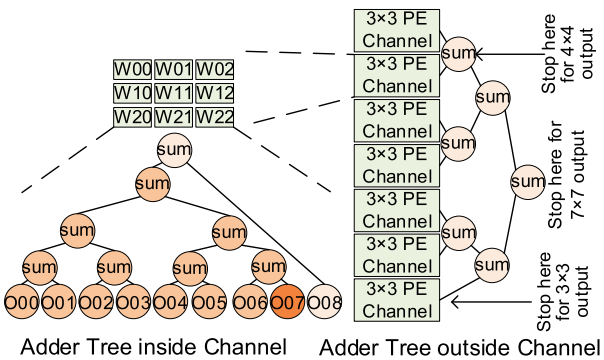
For 1×1 kernels, each PE channel can hold nine of them. Therefore, we need to load nine weights of different input channels to add them up as the result for one output channel. The 1×1 kernels exist in two kinds of operations, i.e., squeeze and extend (Fig. 4), in Squeezenet. In squeeze operation, all the kernels are 1×1 and the inputs are loaded in input channel order. In extend operation, there are two kinds of kernels. The 1×1 convolution works together with 3×3 convolution, or one by one and the results are combine when finish. Either way will result in the difference in input



(a) Kernel Allocation Pattern with size of 3×3



(b) Kernel Allocation Pattern and Output Storage Pattern with Size bigger than 3×3



(c) Output Accumulation Pattern

Fig. 9. Normal 3×3 PE channel design: (a), (b) K00 is first kernel in the first kernel group K0. (c)The outputs inside one channel are accumulated through adder tree. Between channels, there is also an adder tree. Selecting which sum is determined by reuse and controlled by control unit.

loading pattern. Therefore, we also need to read them from the dual-port buffer, as shown in Fig. 6. When processing layers with larger kernel size, the input loading is seperated into rows and n BRAMs are enabled to broadcast n rows of one input feature map to each channel group.

Based on the PE combination, we propose the overall PE channel design in Figs. 6 and 9. In Figs. 10a and 10b, the kernel K_{00} of kernel group K_0 is loaded into the first channel C_0 or CG_0. After C_0 or CG_0 completes the computation with input I_{00}, K_{01} is loaded. During this time, all channels share the input I_{00}. To keep the same input loading pattern for the next layer, the outputs or rows from one channel group should be stored in one BRAM. However, storing outputs in a round robin order causes huge muxing overhead. Therefore, we constrain the output in one channel group which eliminates the output muxing overhead with little control overhead in the right half of Fig. 6. What's more, to collect the partial results of the PE groups, a flexible adder tree is implemented as in Figs. 6 and 9c. The adder tree inside the channel is marked with three kinds of color. All of the sum points inside will be used during the 3×3 setup, among which, the sum points in orange are for 7×7 reuse setup and the deep orange point is for 4×4 reuse setup. Compared with the standard weight-stationary design, our

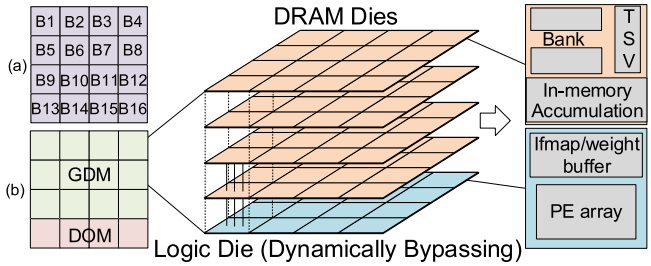


Fig. 10. 3D architecture design.

design use 3750 (9%) more LUTs (adder tree) and 116(10%) more control LUTs.

Multi-network mapping: To broadly adapt our design to other neural networks, we define a series of formulas to compute the most appropriate configuration, referred as kernel size mapping formulas and resource mapping formulas, separately.

The size of PE array should satisfy one of the following formulas, depends on the range of the size and corresponding utilization,

$$\begin{cases} \mathrm{K}_d \geq a \times \mathrm{K}_{min} \ and \ \mathrm{K}_d \geq b \times \mathrm{K}_{mid} \ and \ \mathrm{K}_d \geq \mathrm{K}_{max} \\ \mathrm{K}_d \geq c \times \mathrm{K}_{min} \ and \ \mathrm{K}_d \geq d \times \mathrm{K}_{mid} \ and \ (K_d)^2 \geq e \times \mathrm{K}_{max} \\ \mathrm{K}_d \geq \mathrm{K}_{min} \ and \ (K_d)^2 \geq f \times \mathrm{K}_{mid} \ and \ \ (K_d)^2 \geq g \times \mathrm{K}_{max} \end{cases}$$

where K_d is the size of *Dandelion*'s PE array. K_{min}, K_{mid}, K_{max} are the kernel size of two neural networks. a, b, c, d, e, f, and g are integer. $(K_d)^2 \geq \mathrm{e} \times \mathrm{K}_{max}$ means that $\mathrm{K}_d \leq \mathrm{K}_{max}$.

To finally determine the size of PE array, we should calculate the utilization first. The formula of utilization is,

when $\mathrm{K}_d \geq \mathrm{K}_{max}$:

$$U_d = \frac{\sum_{i=1}^{L} I_i \times O_i \times (Os_i)^2 \times (K_i)^2}{\sum_{i=1}^{L} I_i \times O_i \times (Os_i)^2 \div (t)^2 \times (K_d)^2}$$

when $\mathrm{K}_{mid} \leq \mathrm{K}_d \leq \mathrm{K}_{max}$:

$$U_d = \frac{\sum_{i=1}^{L} \begin{matrix} I_i \times O_i \times (Os_i)^2 \times (K_i \div e + 1) \times e \times K_i, K_i = K_{max} \\ I_i \times O_i \times (Os_i)^2 \times (K_i)^2, K_i = K_{mid} or K_{min} \end{matrix}}{\sum_{i=1}^{L} \begin{matrix} I_i \times O_i \times (Os_i)^2 \times (K_i \div e + 1) \times (K_d)^2, K_i = K_{max} \\ I_i \times O_i \times (Os_i)^2 \div (t)^2 \times (K_d)^2, K_i = K_{mid} or K_{min} \end{matrix}}$$

when $\mathrm{K}_{min} \leq \mathrm{K}_d \leq \mathrm{K}_{mid}$:

$$U_d = \frac{\begin{matrix} I_i \times O_i \times (Os_i)^2 \times (K_i \div g + 1) \times g \times K_i, K_i = K_{max} \\ \sum_{i=1}^{L} I_i \times O_i \times (Os_i)^2 \times (K_i \div f + 1) \times f \times K_i, K_i = K_{mid} \\ I_i \times O_i \times (Os_i)^2 \times (K_i)^2, K_i = K_{min} \end{matrix}}{\begin{matrix} I_i \times O_i \times (Os_i)^2 \times (K_i \div g + 1) \times (K_d)^2, K_i = K_{max} \\ \sum_{i=1}^{L} I_i \times O_i \times (Os_i)^2 \times (K_i \div f + 1) \times (K_d)^2, K_i = K_{max} \\ I_i \times O_i \times (Os_i)^2 \div (t)^2 \times (K_d)^2, K_i = K_{min} \end{matrix}}$$

where L is the total number of layers. I_i and O_i are the number of input and output feature map, respectively. Os_i is the output size for layer i. K_i is the size of kernel for layer i. This formula calculate the ratio between used PEs and available PEs during all the computation stages.

To efficiently allocate resource to two neural networks, we define the method to acquire the computation ratio,

$$R_d = \frac{\sum_{i=1}^{L_D} I_i \times O_i \times (K_i)^2 \div U_G}{\sum_{j=1}^{L_D} I_j \times O_j \times (K_j)^2 \div U_D}$$

where U_G and U_D are the utilization of GAN and DNN in a certain K_d, respectively. The ratio of resource determines whether the two-network system can be balanced to achieve best performance.

The next step for resource allocation is to determine C_d, the number of PE arrays (channels) in one channel group. Assume that the number of PE channels equals to N_c, $\mathrm{K}_d \leq \mathrm{K}_{max}$, and R_d equals to a: 1 or 1: a, C_d should be equal to $K_{max} \times Ceil((K_d)^2 \div K_{max})$ to reuse input data. If $N_c \neq \mathrm{b} \times \mathrm{C}_d$, the remaining part should be allocated to DNN channels which slightly affects the ideal ratio.

With these mapping formulas and slight adjustment, one can easily connect any two neural networks with optimized configuration.

4.4 3D Implementation

Besides the normal 2D architecture design, the emerging and promising 3D architecture can further benefit our design. There are three significant advantages of 3D memory: high bandwidth support, low access latency and low dynamic energy. The bandwidth limitation of AXI bus (256 bits per clock cycle) is weakened in 3D architecture, and the hardware could support larger PE channel on logic die using weight stationary dataflow. Besides, the large capacity and high speed save the time to write and read the data from the off-chip memory. Lower memory access latency could get rid of idleness. With less energy consumption, network on 3D architecture obtains portability on edge device.

TETRIS [12] develops in-memory accumulation, by-pass buffer and across-vault partition to explore the neural network computation on 3D architecture. The partition of jobs among vaults is proposed to minimize the across-vault memory access.

Unlike TETRIS which applies row stationary dataflow of Eyeriss on its logic die, we place *Dandelion-function* to it and combine separate BRAMs into smaller global buffer in total size which requires slightly more registers for buffer writing than 2D design. Based on that, our design further focuses on the two-network acceleration and discovers novel mode-partitioning and buffer-bypassing methods which is suitable for our training mode. In our 3D implementation shown in Fig. 10, 16 vaults are deployed to perform network training. We evaluate batch-partitioning (Fig. 10a) and propose mode-partitioning (Fig. 10b) with corresponding bypassing strategies.

Batch partitioning indicates that each vault is in charge of the computation of one sample in a batch. Although it cannot benefit the inference latency and cause duplication of networks, our *Dandelion* take advantage of its flexibility by using small network and distributing computation to vaults with adaptive configuration thanks to the different samples of one batch(scarce data or abundant data). The ofmap buffer is bypassed for vaults dealing with scarce data. Ofmap and weight buffers are bypassed for abundant data.

Mode partitioning means that all vaults are seperated into two types, two-mode vaults (TMV) and one-mode vaults (OMV). TMV will operate on both DOM and GDM which requires different buffer size and dynamic bypassing strategy caused by switching network weights. Instead of buffering input feature maps as normal, weights for another network is loaded into global buffer to reach less latency.

4.5 Tradeoff Between Computation and Memory Resource

Apart from the opportunities mentioned above, we also notice that the GAN generator provides an opportunity to trade the computation for the memory resource. In our implementation, after the first epoch, all of the generated images are sent to the off-chip memory for the computation of later epochs. There is another choice that the images are generated every epoch without buffering so that the memory accesses can be avoided. This is especially beneficial to the mobile ends where the memory is expensive, and the computation is small. Generating random training data over epochs also reduces the chance of overfitting. With GDM only in the first epoch, *Dandelion* has 23% less overall benefits (1.53x->1.30x) and brings 90% more storage cost (depends on scarcity level).

5 Evaluation Methodology

5.1 Function Validation

Note that, the effectiveness of *Dandelion* on scarce dataset training accuracy is the key for its function and the architecture design. We modified the Pytorch code of StarGAN and SqueezeNet to conduct the following two experiments. In both experiments, we use three methods to enrich the dataset, normal image processing method (NIP), DAGAN (representing normal GAN methods) and *Dandelion-function* (representing orthogonal GAN methods).

The first training experiment is performed with the famous face dataset CelebA [13], which contains 202599 face images with 40 binary attributes and Pubfig [14], which labels 2399 images by 60 names and each class contains at least 10 images. 10% images are randomly selected as the test set (Pubfig-Test), and the rest are the training set (Pubfig-Train) which trains DNN to recognize one person in Pubfig.

For our original baseline, the SqueezeNet is trained on full Pubfig-Train with different scarcity levels by randomly sampling it at a certain ratio. For NIP, we preprocess the target images with normal image processing operations (resize, crop, flip, rotation, and grayscale). For DAGAN, we choose 5 classes with equal number of samples (60) from the scarce dataset (Pubfig-Train) to train it to enrich the samples for the target person. For our design, we first train the StarGAN on CelebA to allow it to learn 7 attributes (i.e., face with smiling, glasses, hat, heavy makeup, and mustache). Then the sampled target images of Pubfig-Train as in the baseline are fed into the StarGAN and restore to the maximal number of target images. After that, the generated images are inserted back into the scarce Pubfig-Train and SqueezeNet is trained on it to obtain the accuracy with *Dandelion*. The batch size is 16. We train 36 epochs for original dataset and 10 epochs for other methods.

The second experiment is designed to show *Dandelion*'s potential improvement in real-world applications. CheXpert [15], a dataset of 191028 chest X-ray gray images with pneumonia-related symptom labels, and COVID-19 dataset containing 484 COVID images and 342 other images [16] are leveraged to train GAN and DNN respectively. Five attributes of the CheXpert dataset that are related to COVID-19, i.e., cardiomegaly, fracture, support devices, pleural other

and lung opacity, are learnt by the GAN to enrich the scarce COVID-19 dataset.

5.2 Architecture Evaluation

We evaluate the architecture support for *Dandelion*. We prototype the first baseline and our 2D *Dandelion*-arch design with all the operation modes and reuse structures on a Xilinx VCU118 Evaluation board with 20Gb off-chip DDR4 and 76 Mb on-chip BRAM connected by AXI bus on 200MHz. The data width in our design is 16-bit for both the kernel weights and intermediate data. Each data buffer is 0.49Mb and supports 96-bit access. The size of one weight buffer is 0.28Kb and supports 144-bit access.

Baseline: In the first baseline, to demonstrate the effectiveness of our proposed inter-network optimization, we simply run StarGAN followed by the DNN in GDM as a general accelerator without PE or data reuse. Since there is no reuse and reconfiguration of the PE channels designed for inter-network training, we choose the output stationary dataflow which results in higher utilization in the baseline. Each PE channel has 4×4 PEs to best fit the output sizes. To conduct fair evaluation, the baseline employs the same number of PEs as *Dandelion* (i.e., 41 PE channels).

The second baseline is a representative accelerator for single-DNN training, i.e., ScaleDeep [9]. ScaleDeep uses two kinds of tiles to separate the computation-consuming works like most of the convolution layers, and memory-consuming works like fully connection layers, respectively. In computation-consuming tile, there is a 2D PE array which can be divided into two parts when necessary. In each 2D array, the inputs and weights are casted in rows and columns respectively to accumulate the partial results in diagonal. In memory consuming tile, the special function units are designed for various computation. Two kinds of tiles are connected in a delicate pattern depending on the chip type. In our experiment, we implement 7 computation-consuming tiles, each of which has 8×3 2D PE arrays. We adjust ScaleDeep to use 4 lanes of each PE for normal convolution and 1 lane for convolution with kernel size of 1×1. The performance is estimated using our in-house simulator with necessary scale-down of the design size.

Dandelion: We compare the performance and utilization of different accelerators under different modes of *Dandelion* and under different dataset scarcity levels to demonstrate the efficiency of our proposed accelerator. Different PE ratios are applied to evaluate the efficiency of PE reuse and our mapping formula. Furthermore, we evaluate our proposed multi-network mapping method across five DNNs (SqueezeNet, Alexnet, Resnet50, Resnet152 and VGG16).

We also break down the energy consumption of *Dandelion* to show the overhead brought by the incorporation of GAN, based on which, we analyze the overall benefit brought by *Dandelion*.

3D: The 3D extension of Dandelion utilizes HMC stack defined in gem-5 [17] with 16 vaults each of which supports a bandwidth of 8 GBps. 225 PEs per vault are deployed on logic die to take advantage of the high bandwidth, which follows the limitation discussed in TETRIS. The power estimation is based on [18] and the performance is evaluated through the combination between 2D logic die and 3D memory on gem5, using batch partitioning and mode partitioning with different vault number, network computation ratio and buffer ratio.

TABLE 1
Accuracy Improvement

SR	*Dandelion*	NIP	DAGAN	Original
100	99.1 | 94.8	91.3 | 88.5	96.4 | 91.5	85.2 | 81.0
50	95.2 | 91.5	85.5 | 83.2	91.8 | 83.2	76.5 | 73.6
30	91.3 | 90.0	77.3 | 69.8	82.7 | 82.5	62.5 | 64.7
10	78.8 | 72.5	64.1 | 57.5	70.4 | 68.7	52.5 | 51.5
5	62.3 | 50.5	51.5 | 50.3	58.5 | 51.3	50 | 50

5.3 System Evaluation

To better understand the overall benefit of our *Dandelion*, we combine the functional and 2D architecture results to estimate the actual timing impact in two steps.

Step1: We compute the total runtime and energy efficiency for different architectures to get usable accuracy (90%) with or without data augmentation, which takes the training time and data collection time into account. Based on the detection time for COVID-19 in early 2020 [19], [20], We assume that the data collection time is 3 days, also as, and the power for data collection is the power of medical diagnostic X-ray set (800W) multiplying its average working time (8 hours/day).

Step2: From step1, we could figure out the time gap to reach the usable accuracy between different configurations which further impacts the detection of COVID-19. We propose a matrix to quantify the real-world influence. The real-world evaluation explores the impact of COVID-19 detection on health and economy.

6 Results and Analysis

6.1 Function Evaluation Results

Table 1 compares the training accuracies of SqueezeNet with different augmentation methods on Pubfig and COVID-19 dataset under different scarcity levels. As shown, *Dandelion* can significantly improve the accuracy of both datasets under all the scarcity levels. However, with extremely scarce dataset (e.g., sampling rate of 5%), the test accuracy is around 50% meaning that the network fails to learn anything even with the assistance of *Dandelion*. When the scarcity level goes lower, the accuracy of *Dandelion* increases rapidly. With the scarcity of 30%, *Dandelion* improves the accuracy from 62.5%|64.7% to 91.25%|90%, which is acceptably usable in our daily life. However, the accuracy of the other methods is still far from usable even when the scarcity is 50%. Impressively, *Dandelion* can further improve the accuracy from 85.15%|81.0% to near optimal (99.07%|94.8%) when the dataset is considered abundant (i.e., the scarcity level is 100%). For other applications where the accuracy requirement can be slightly loosened, *Dandelion* can even make the training results usable with dataset scarcity at 30%. Although DAGAN also increases the accuracy to some extent on relatively high scarcity level, its performance is restricted because of the lack of learning information learnt from the scarse dataset itself.

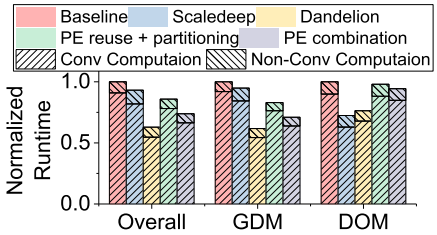


Fig. 11. Performance of different modes between architectures.

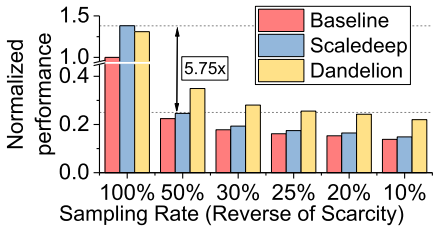


Fig. 12. Performance with different dataset scarcity levels.

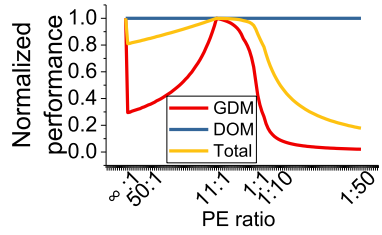


Fig. 13. Performance with different PE ratio of GAN and DNN.

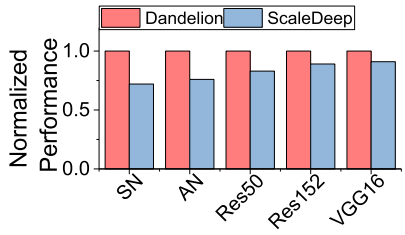


Fig. 14. Multi-network performance of *Dandelion*.

6.2 Architecture Evaluation Results

In Fig. 11, we compare the runtime of two baselines and our *Dandelion* when equipped with different techniques in different modes (DOM, GDM and overall training time with necessary mode activated). The scarcity level is set to 10%. In GDM, *Dandelion* gains the performance improvement up to 52% compared with ScaleDeep, which is because that: (1) The large sizes of StarGAN's kernel results in great waste which the dataflow of ScaleDeep suffers from. (2) Our baseline uses output -stationary dataflow. It can produce 4×4 outputs in each computation round which is suitable for 256×256 outputs and processes the weights in one kernel by loading them in each clock cycle to handle different kernel size. (3) *Dandelion*'s PE combination and partition reduce the runtime of convolutional computation. In DOM, the SqueezeNet only computes with smaller kernels and thus can benefit from the dataflow of ScaleDeep. The baseline architecture has limitation on the bandwidth of weights and performs the worst when computing 1×1 kernel in 4×4 PE channels. To summarize, although ScaleDeep performs better than *Dandelion* in DOM, our design has strength on the overall running time, thanks to the advantages on GDM, which occupies the dominant part of computation.

Fig. 12 measures the overall performance difference when we change the scarcity of dataset. In Section 6.1, we discuss the influence of dataset scarcity on the training accuracy. Now, the scarcity can also affect the hardware performance since it changes the ratio of computation time in two modes. As the scarcity level becomes higher, GDM computation tends to dominate the latency. Therefore, the disadvantage of inter-network training along with the advantage of our acceleration becomes more substantial. With additional computation, ScaleDeep becomes 5.75 times slower, and *Dandelion* reduces at least 40% of the gap with accuracy improvement according to the data scarcity level.

As is defined in Section 4.3, the multi-network mapping method enables the deployment of any two networks on our architecture. Fig. 13 validates the PE allocations formula. We set several kinds of PE ratio to search for the best allocation strategy. When the number of GAN PE channels is equal to the DNN PE channels (1:1), the computing resources are the same, but DNN still dominate the computation. In this situation, the DNN has to wait until GAN completes its work which reduces the performance or process next image which requires further scheduling and control, to be discussed in future work. When the ratio between GAN and DNN channels increases, the idleness is relieved until the point 11:1, the most efficient ratio obtained through the formula in Section 4.3. At this point, GAN and DNN work at the same pace which reduces data movement and achieves the best performance. Then, the ratio keeps increasing until all the PE channels are allocated to GAN. In this case, the GAN works first by itself and generates all the images and stores them back to off-chip, followed by the training of DNN. Although there is a sudden rise in the performance, it brings extra latency and energy to access the off-chip memory. If the ratio becomes smaller, the performance becomes worse, and GAN need to wait for DNN. The generated image either waits on-chip wasting time and storage or goes back to off-chip causing extra data movement and energy consumption. Finally, the extreme small ratio means that there is no GAN in *Dandelion*. *Dandelion* becomes the training of baseline SqueezeNet with unusable accuracy under scarce dataset as shown in Table 1.

Because of the computation varience of network, we evaluate the performance improvement with different networks on *Dandelion* in Fig. 14, using the mapping method in 4.3. We keep the the sample rate to be 10% and the training epoch is 10. VGG16 gains the best utilization resulting from its unchanged 3×3 kernels. The total speedup suffers from the increasement of computation ratio. In addition, different network architectures are corresponding to various training epochs. With the epochs increasing, images generated in the first epoch take smaller ratio in total training images and the benefit of data reuse decrease slowly. However, networks which requires more training epochs are designed for abundant datasets and face less scarcity problems.

Table 2 elaborates the energy breakdown in *Dandelion*. Note that, the energy consumed by BRAM during GDM takes the dominant position due to frequent on-chip data movement. To clearly explain the power condition, we utilize the data from the first epoch, and the entire power performs much better for the whole training process.

TABLE 2
Energy Breakdown

PE(DOM)	6.24%
PE(GDM-DNN)	2.50%
PE(GDM-GAN)	19.85%
BRAM(DOM)	18.34%
BRAM(GDM-DNN)	26.29%
BRAM(GDM-GAN)	18.96%
Attribute Control	1.31%
Adder Tree	5%
Other	1%

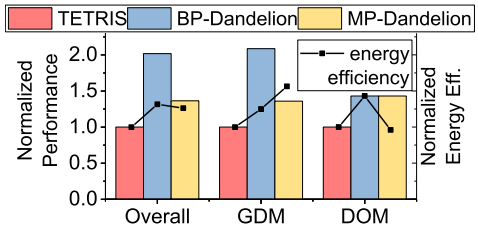


Fig. 15. Performance on 3D architecture.

Fig. 15 indicates *Dandelion*'s performance when equipped with batch-partitioning (BP) and mode-partitioning (MP) on 3D architecture. As is shown, even though it cannot benefit the inference latency, batch-partitioning performs 2 times better than TETRIS and wastes less cross-vault access energy, resulting from the fully parallelism. However, batch partition may suffer from the synchronization problem brought by the imbalance between different processing speed of scarce data and abundant data to be addressed by dynamic batch size and larger vault number. Mode-partitioning utilize dynamic bypass buffer to reduce memory access times and latency, achieving higher energy efficiency in GDM while it acts worse in DOM. Moreover, it processes 26% faster than TETRIS and survives from the synchronization problem becoming more practical than batch partitioning.

6.3 System Evaluation Results

Combining the function and architecture results, in Fig. 16 we compare the performance and energy efficiency between different architectures (2D) to reach the same accuracy on Covid-19 dataset. As shown, without data augmentation, it takes 625x more time to get the usable accuracy and the energy efficiency is even lower because of the high power of the medical X-ray device. Although the data collection may be pipelined, there will be considerable latency for data collection and training. With data augmentation, the time gap is reduced to 0.3x which is still unignorable.

Fig. 17 shows how the cumulative cases and economy condition are influenced by quarantine policy and COVID-19 detection. Data from WHO [21] is utilized to build the extended SEIR (Susceptible-Exposed-Infectious-Recovered) compartmental mathematical model which indicates how the early detection and response could have rescued lives. To evaluate the real condition from reported case, we adjust the parameters of SEIR model, namely the initial cases and the transmission rate in each stage. According to Wikipedia [22], the U.S. government recommended to keep social distance on 03/16/2020 while most states placed stay-at-home

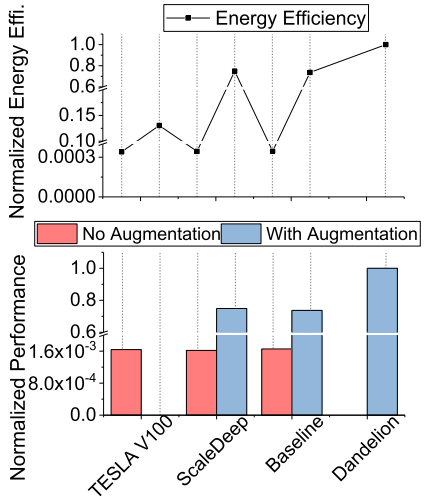


Fig. 16. Performance and energy efficiency with the same accuracy.

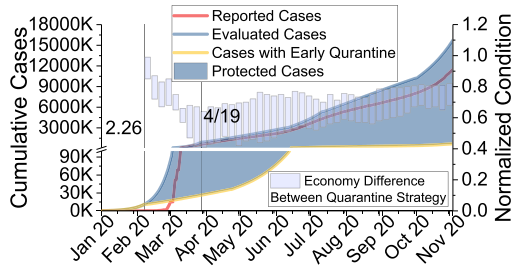


Fig. 17. Real world benefits of *Dandelion*.

order around 03/19/2020. Therefore, the impact of quarantine on epidemic control appears about 20 days later, obtained from two times of the incubation period. Assume that the government decides to begin quarantine based on the number of detected cases, or the severity of COVID-19. If our early detection were available to use, the quarantine would have been assigned previously on 02/26/2020 (yellow line), which would result in a case number of 770k by Nov 20, representing 0.6% of reported cases. Based on the SEIR model and the report from Whitehouse [7], we further evaluate the average economy impact of COVID-19. The economy effect shown by purple stack indicate that although the on-time quarantine results in early economy decrease and longer period, the range and degree of the negative impact is lower. 20% of the economy loss can be avoided.

7 Related Work

Some image processing methods are used to get simply changed picture [2], [23]. But the neural network is hard to learn brand new information from them. Augmenter[24] combines different image processing methods in a random order to generate new images. Generative adversarial network (GAN) has also emerged as a promising technique to generate new images. Some researchers randomly combine two images of dataset in a special manner to produce new samples [23]. CovidGAN [25] uses ACGAN [26] to enrich the CXR dataset for CNN training. DAGAN [27] generates images in the same class with different looks. Synthetic images are fed to SimGAN [28] and realism is added to

refined outputs. PGGAN [29], [30] improves generator and discriminator progressively to get high-resolution images. BAGAN [31] converts images to latent vector as the input data for pre-initialized GAN to get better performance. These methods are constrained by the limited information of the scarce dataset. In contrast, Dandelion can enrich the scarce dataset to enable better analysis using information from abundant dataset. CycleGAN [32] learns to translate characteristics into another domain with unpaired data which gets rid of the dependency on dataset. In low-shot learning area, an hallucinator [12] is proposed to change animals' pose and surroundings, which is a three-layer MLP. However, with the ability of changing one characteristic at a time, CycleGAN and hallucinator provide relatively less augmented data.

Besides of function challenges, neural networks suffer from the large computation, memory, and energy consumption. Most of the accelerators are designed for the inference phase as this is more time sensitive. PREMA [33] designed for multi-task DNN inference concentrates the preemption mechanisms on TPU, which could be applied to our work with some adjustment. The evolvement of neural network's function urges people to develop training accelerators. Eager pruning [34] reduces computation of the scarce weights in training to save time. GANAX [35] and [36] avoid zero computation by two different methods. These are orthogonal to our design which also benefit the GAN computation of *Dandelion*. ScaleDeep [9] designs specific architecture for different function unit and use diagonal dataflow to speed up training which improves the performance of single neural network training. TPU [37] proposes special matrix processing unit to be suitable for convolution, designed for general computation. Multi-network accelerators like [38] are proposed to support efficient cloud computing which processes various requests from clients and focus on supporting large number of different networks as well. None of them exploit the opportunity of processing the same dataset with selected two neural networks.

8 Conclusion

In this paper, we build a function-architecture co-design to address the problem of data scarcity in DNN training and extend it to 3D area. Urged by DNN's demand of labeled images, we explore the opportunities on connecting GAN to DNN to boost DNN usability. We also propose the first dual-network accelerator aiming at speeding up inter-network computation. The results indicate that our design improves up to 37.5% training accuracy with 30% performance improvement, 40% reduction of latency overhead and 26.3% overall performance on 3D architecture. In the future, we will explore the opportunities when connecting various types of neural networks.

References

[1] "Autopilot AI | tesla," Accessed: Sep. 17, 2020. [Online]. Available: https://www.tesla.com/autopilotAI

[2] A. A. A. Setio *et al.*, "Pulmonary nodule detection in CT images: False positive reduction using multi-view convolutional networks," *IEEE Trans. Med. Imag.*, vol. 35, no. 5, pp. 1160–1169, May 2016, doi: 10.1109/TMI.2016.2536809.

[3] K. Tan, "Application of neural networks to UNIX computer security," in *Proc. IEEE Int. Conf. Neural Netw.*, 1995, pp. 476–481, doi: 10.1109/icnn.1995.488223.

[4] M. Ghiassi, D. Lio, and B. Moon, "Pre-production forecasting of movie revenues with a dynamic artificial neural network," *Expert Syst. Appl.*, vol. 42, no. 6, pp. 3176–3193, Apr. 2015, doi: 10.1016/j.eswa.2014.11.022.

[5] S. Ouellet, "Real-time emotion recognition for gaming using deep convolutional network features," Aug. 2014.

[6] T. Ozturk, M. Talo, E. A. Yildirim, U. B. Baloglu, O. Yildirim, and U. Rajendra Acharya, "Automated detection of COVID-19 cases using deep neural networks with X-ray images," *Comput. Biol. Med.*, vol. 121, Jun. 2020, Art. no. 103792, doi: 10.1016/j.compbiomed.2020.103792.

[7] "The council of economic advisers," Accessed: Nov. 24, 2020. [Online]. Available: https://www.whitehouse.gov/wp-content/uploads/2020/08/Evaluating-the-Effects-of-the-Economic-Response-to-COVID-19.pdf

[8] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "StarGAN: Unified generative adversarial networks for multi-domain image-to-image translation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8789–8797, doi: 10.1109/CVPR.2018.00916.

[9] S. Venkataramani *et al.*, "Scaledeep: A scalable compute architecture for learning and evaluating deep networks," in *Proc. Int. Symp. Comput. Archit.*, 2017, pp. 13–26, doi: 10.1145/3079856.3080244.

[10] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017, doi: 10.1109/JSSC.2016.2616357.

[11] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for arm cortex-M CPUs," Jan. 2018.

[12] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and efficient neural network acceleration with 3D memory," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 751–764, Apr. 2017, doi: 10.1145/3037697.3037702.

[13] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 3730–3738.

[14] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar, "Attribute and simile classifiers for face verification," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2009, pp. 365–372, doi: 10.1109/ICCV.2009.5459250.

[15] A. Bustos, A. Pertusa, J.-M. Salinas, and M. de la Iglesia-Vayá, "PadChest: A large chest X-ray image dataset with multi-label annotated reports," *Med. Image Anal.*, vol. 66, Dec. 2020, Art. no. 101797, doi: 10.1016/j.media.2020.101797.

[16] J. P. Cohen, P. Morrison, and L. Dao, "COVID-19 image data collection," Mar. 2020, *arXiv*.

[17] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, May 2011, doi: 10.1145/2024716.2024718.

[18] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 380–392, Oct. 2016, doi: 10.1145/3007787.3001178.

[19] V. M. Corman *et al.*, "Detection of 2019 novel coronavirus (2019-nCoV) by real-time RT-PCR," *Eurosurveillance*, vol. 25, no. 3, Jan. 2020, Art. no. 2000045, doi: 10.2807/1560-7917.ES.2020.25.3.2000045.

[20] "CityMD | CityMD," Accessed: Apr. 16, 2021. [Online]. Available: https://www.citymd.com/news/covid-19-testing-update

[21] World Health Organization, "United States of America: WHO coronavirus disease (COVID-19) dashboard | WHO coronavirus disease (COVID-19) dashboard," *WHO Coronavirus Dis. (COVID-19) Dashboard*, 2020. Accessed: Nov. 23, 2020. [Online]. Available: https://covid19.who.int/region/amro/country/us

[22] "Timeline of the COVID-19 pandemic in the United States - Wikipedia," Accessed: Nov. 24, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Timeline_of_the_COVID-19_pandemic_in_the_United_States#cite_note-509

[23] H. Inoue, "Data augmentation by pairing samples for images classification," Jan. 2018.

[24] M. D. Bloice, C. Stocker, and A. Holzinger, "Augmentor: An image augmentation library for machine learning," Aug. 2017, *arXiv*.

[25] A. Waheed, M. Goyal, D. Gupta, A. Khanna, F. Al-Turjman, and P. R. Pinheiro, "CovidGAN: Data augmentation using auxiliary classifier GAN for improved covid-19 detection," *IEEE Access*, vol. 8, pp. 91916–91923, 2020, doi: 10.1109/ACCESS.2020.2994762.

[26] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier GANs," in *Proc. Mach. Learn. Res.*, 2017, pp. 2642–2651.
[27] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255, doi: 10.1109/CVPR.2009.5206848.
[28] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2242–2251, doi: 10.1109/CVPR.2017.241.
[29] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "ProgressiveGAN," in *Proc. 6th Int. Conf. Learn. Represent.*, 2018, pp. 1–26.
[30] C. Bowles *et al.*, "GAN augmentation: Augmenting training data using generative adversarial networks," Oct. 2018, *arXiv*.
[31] G. Mariani, F. Scheidegger, R. Istrate, C. Bekas, and C. Malossi, "BAGAN: Data augmentation with balancing GAN," Mar. 2018.
[32] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using Cycle-consistent adversarial networks," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2242–2251, doi: 10.1109/ICCV.2017.244.
[33] Y. Choi and M. Rhu, "PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2020, pp. 220–233, doi: 10.1109/HPCA47549.2020.00027.
[34] J. Zhang, X. Chen, M. Song, and T. Li, "Eager pruning," in *Proc. 46th Int. Symp. Comput. Archit.*, 2019, pp. 292–303, doi: 10.1145/3307650.3322263.
[35] A. Yazdanbakhsh, K. Samadi, N. S. Kim, and H. Esmaeilzadeh, "GANAX: A unified MIMD-SIMD acceleration for generative adversarial networks," in *Proc. Int. Symp. Comput. Archit.*, 2018, pp. 650–661, doi: 10.1109/ISCA.2018.00060.
[36] M. Song, J. Zhang, H. Chen, and T. Li, "Towards efficient micro-architectural design for accelerating unsupervised GAN-based deep learning," in *Proc. Int. Symp. High-Perform. Comput.*, 2018, pp. 66–77, doi: 10.1109/HPCA.2018.00016.
[37] N. P. Jouppi *et al.*, "In-Datacenter performance analysis of a tensor processing unit," in *Proc. Int. Symp. Comput. Archit.*, 2017, pp. 1–12, doi: 10.1145/3079856.3080246.
[38] E. Baek, D. Kwon, and J. Kim, "A multi-neural network acceleration architecture," in *Proc. Int. Symp. Comput. Archit.*, 2020, pp. 940–963, doi: 10.1109/ISCA45697.2020.00081.

Xiangru Chen (Student Member, IEEE) received the BS degree in electronic information engineering from Shandong University, in 2016, and the MS degree in electrical and computer engineering from the University of Florida, in 2018. He is currently working toward the PhD degree with the Department of Electrical and Computer Engineering, University of Florida. His research interest include the architecture support for ML applications.

Jiaqi Zhang (Student Member, IEEE) received the BS degree in communication engineering from Beijing Jiaotong University, in 2016. She is currently working toward the PhD degree at the Department of Electrical and Computer Engineering, University of Florida. Her research interests include the software and hardware acceleration of emerging algorithms and applications including machine learning and IoT.

Sandip Ray (Senior Member, IEEE) received the PhD degree from the University of Texas at Austin. He is currently an endowed IoT Term professor with the Department of Electrical and Computer Engineering, University of Florida. His research involves developing correct, dependable, secure, and trustworthy computing through cooperation of specification, synthesis, architecture and validation technologies. He focuses on next generation computing applications, including IoT, autonomous automotive systems, etc. Before joining University of Florida, he was a senior principal engineer with NXP Semiconductors, where he led the R&D on security architecture and validation of hardware platforms for automotive and IoT applications. Prior to that, he was a research scientist with Intel Strategic CAD Labs, where he led research on validation technologies for security and functional correctness of SoC designs. He is the author of three books and more than 90 publications in international journals and conferences.