Analyzing and Mitigating the Impact of Permanent Faults on a Systolic Array Based Neural Network Accelerator

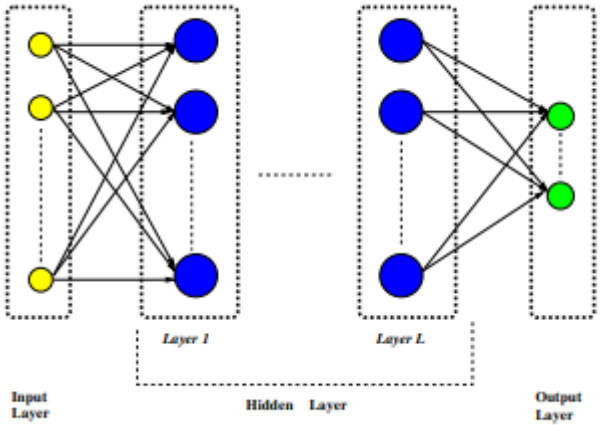
Ref: https://arxiv.org/pdf/1802.04657.pdf

Deep neural network accelerator

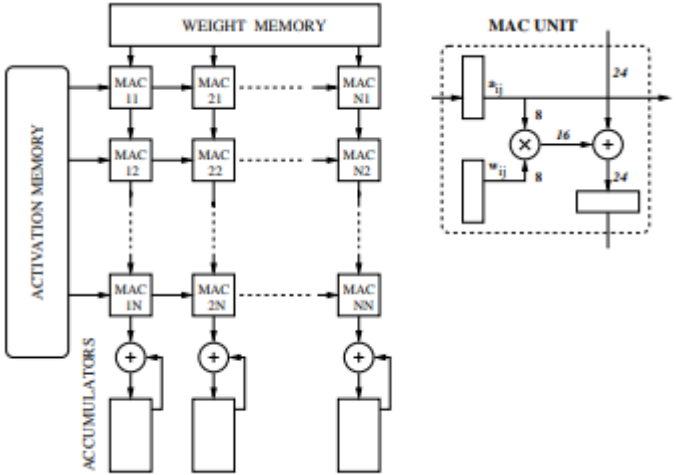
Due to the compute intensive nature of DNNs, several accelerator designs have been proposed in the literature to accelerate the inference process of DNNs.

These accelerators are mainly capable of performing efficient vector and matrix multiplication operations, which are the fundamental operations in the DNN inference.

This is achieved by using multiple computational units which operate in parallel and by enabling local data sharing/reuse in these units.



(a) DNN Architecture

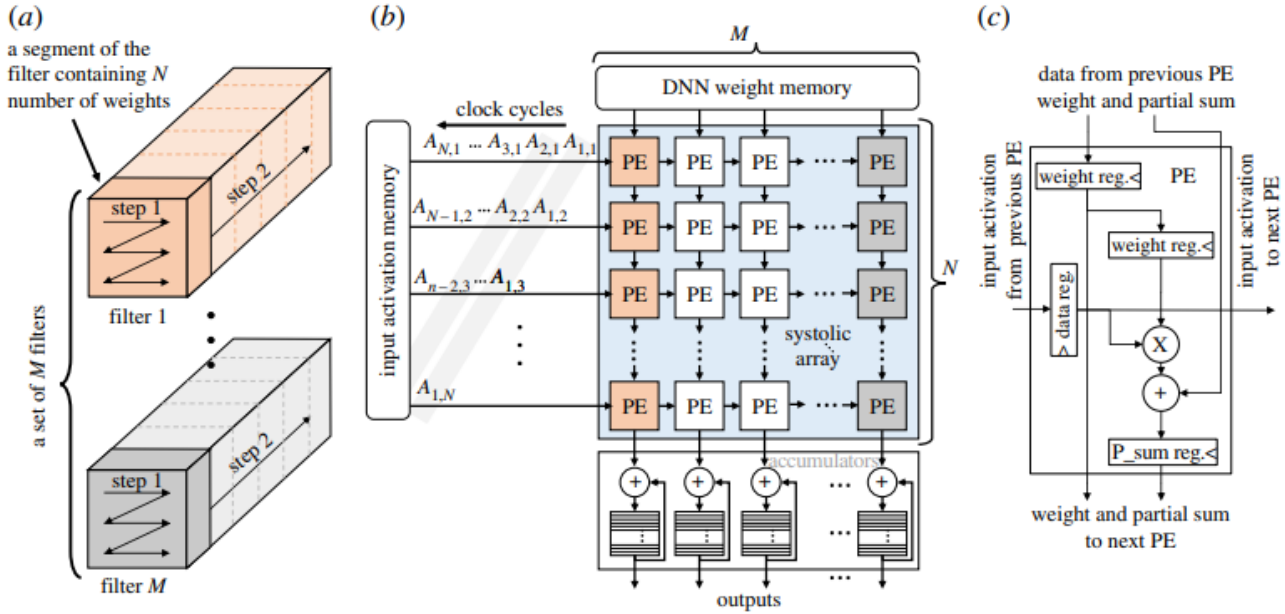


(b) TPU Architecure

The Tensor Processing Unit (TPU)

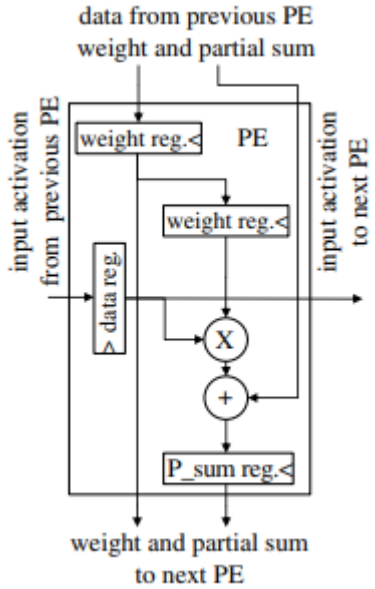
prominent DNN accelerators developed by Google for addressing the increased cloud-based DNN related workloads.

TPU is a systolic array composed of processing elements (PEs) that are connected in a 2D grid like manner.



. (a)The steps show the sequence of unrolling and mapping of the weights onto the array. (b) baseline systolic array-based DNN accelerator (similar to the design of well-know systolic arrays like Google TPU and Eyeriss, illustrating the mapping of a segment of filters highlighted in (a). (c) Detailed design of the processing element (PE).

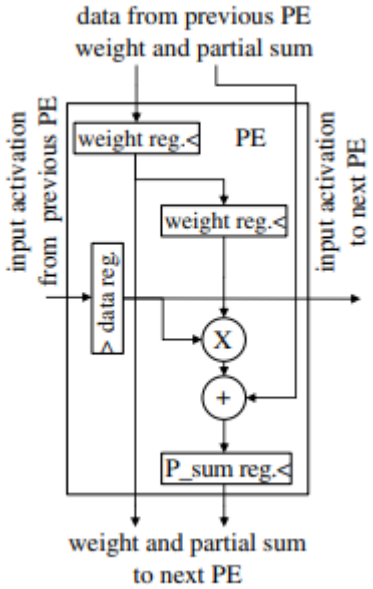
Processing element (PE)



The PEs work in lock steps with their neighbouring PEs to generate the output.

- For example, in the first clock cycle, the top left corner PE will multiply the first weight with the first input activation and then pass the partial output to its downstream neighbour and the input activation to the PE on its right side.
- The downstream PE (i.e. the second PE in the first column), in the next clock cycle, computes the product of the second weight with the second activation and adds the available partial product from the upstream PE to generate the partial product for its downstream PE.
- In the meantime, the first PE in the first column will generate the first partial product related to the second input activation vector and the first PE in the second column will generate the first partial product related to the first input activation vector.
- By continuing this procedure, the result of the first dot product from the systolic array will be available after N clock cycles and, at peak, the array can generate one result per clock cycle per column.

processing element (PE)



- If the number of weights in the filters/neurons is more than the number of rows in the array, the weights cannot be mapped to the array at the same time.
- In such cases, the results generated by the array are not complete, and the accumulation units connected below the array are responsible for storing the partial products and accumulating them with the rest of the corresponding partial products of the filters/neurons to compute the final outputs.

Scaling Problem

- An important challenge with future technology scaling is the increase in fault rates, including both permanent (hard errors) and temporary faults (soft errors).
- Temporary faults might occasionally impact the DNN’s classification results, but their overall impact on classification accuracy is small.
- Permanent faults can affect the result of every DNN execution and significantly reduce the classification accuracy.
- While permanent faults, at least those that are related to manufacturing defects, can be identified during post-fabrication testing, discarding every chip with a permanent fault reduces yield.

Yield

Although the goal of every fabrication process is to produce only working dies, in practice the process is never perfect and not all dies work or operate as desired within specs.

Yield is a quantitative measurement of the process quality in terms of working dies.

Where,

- N_{good} - number of working dies per wafer
- N_{total} - number of dies per wafer

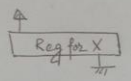
$$Y = \frac{N_{\text{good}}}{N_{\text{total}}}$$

Why fault models?

- Impossible to count and analyze all faults.
- abstract physical defects and define logical models.
- Functional - check faults in functional blocks (registers, adder, multipliers etc.)

Eg of faults

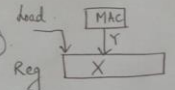
(1) Integer X : (W/I) → stored in reg



If V_{DD} is disconnected
⇒ all 0.

If GND is disconnected
⇒ all 1.

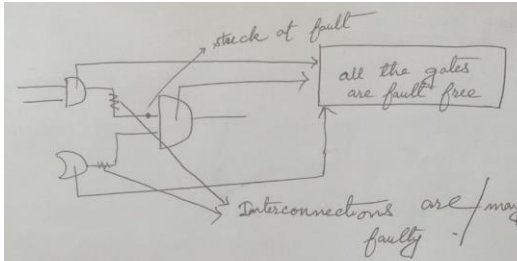
(2)



X = Y.

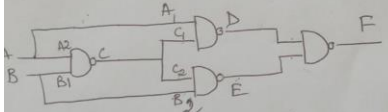
If load is high :
Y will be assigned to X.

If load is disconnected
⇒ fault.



stuck at fault :

- some of the circuit lines are permanently at logic 0 or logic 1.



Total number of lines = 12
Total faults = 24 (Stuck at faults)
[A/0, A/1, A1/0, A2/0, A1/1, A2/1 . . .]

Single stuck at fault :
Only one line of the circuit has a stuck-at fault at given time.

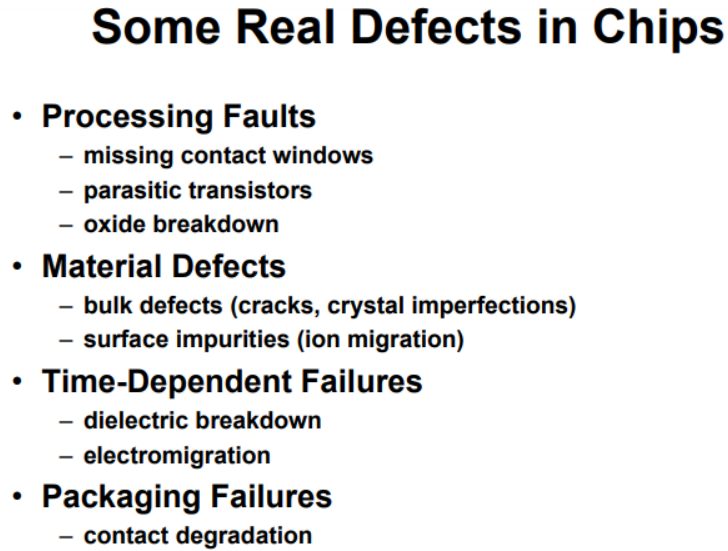
Total numbe of stuck at fault = ~~2^K~~ 2K [K lines]
= 24

Faults, Errors and Failures

- **Fault: A physical defect within a circuit or a system**
  - **May or may not cause a system failure**
- **Error: Manifestation of a fault that results in incorrect circuit (system) outputs or states**
  - **Caused by faults**
- **Failure: Deviation of a circuit or system from its specified behavior**
  - **Fails to do what it should do**
  - **Caused by an error**
- **Fault ---> Error ---> Failure**

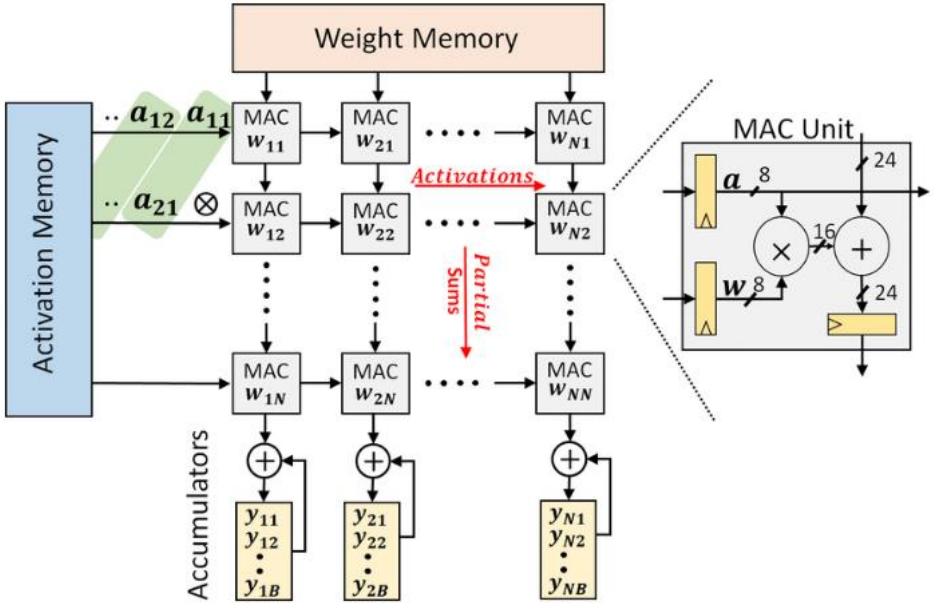
Imperfections in DNN accelerators

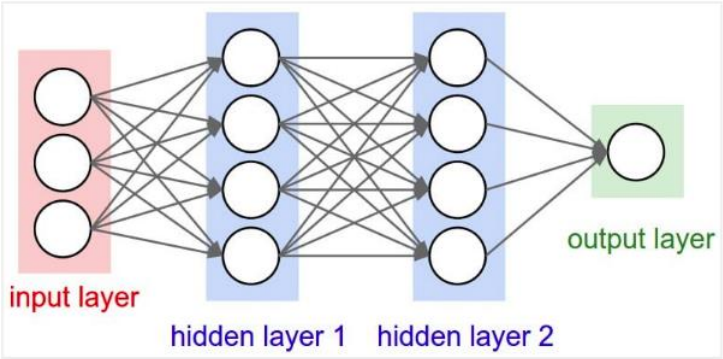
- DNN accelerators are fabricated using nanometre CMOS technologies, which require a highly sophisticated manufacturing process.
- The imperfections in the process result in defects in the fabricated chips.
- These defects can take a wide variety of forms, from permanent faults (e.g. stuck-at faults) that affect the functionality of the chips to variations that affect just the operating characteristics of the hardware (e.g. timing errors).



Permanent faults (e.g. stuck-at faults) in DNN accelerators

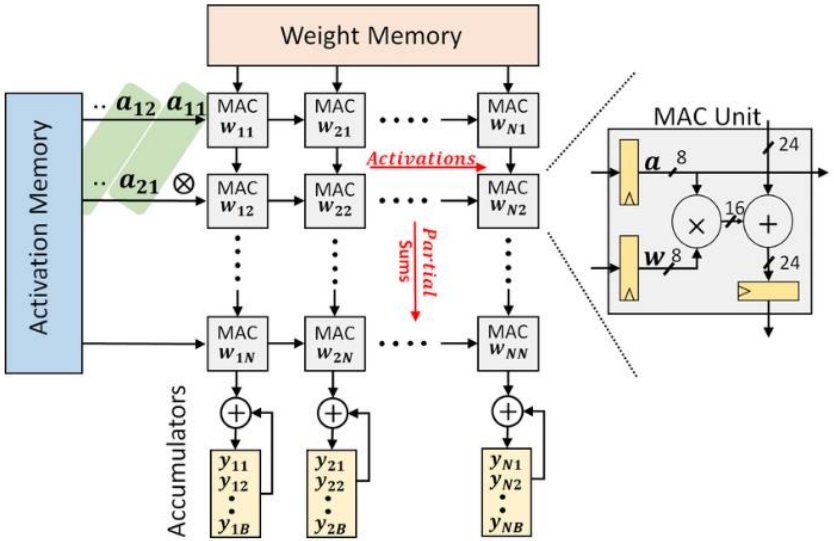
How TPU Works?





Two or more hidden layers comprise a Deep Neural Network

How TPU Works?



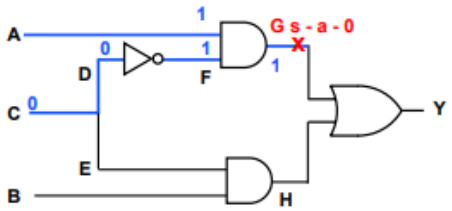
- MAC_11 computes w_11*a_11 in the first clock cycle,
- MAC_12 unit adds w_12*a_21 to MAC11's product in the next clock cycle, and so on.

- In clock cycle N, MAC_1N unit outputs:
  y_11 = ∑ w_1i*a_i1,
  [the first element of the output matrix]

- MAC_1N proceeds to output y_12, . . . ,y_1B in subsequent clock cycles.

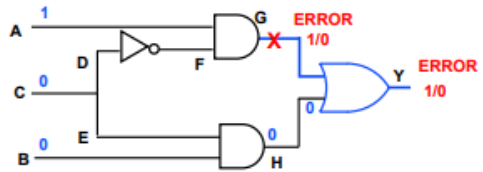
- The second column receives the same stream of inputs as the first column, but delayed by one clock cycle.
- This column outputs y_21,y_22, . . . ,y_2B.
- In this manner, a batch of B inputs is multiplied by an N × N weight matrix in 2N + B clock cycles.

Fault Excitation



Activates the fault s-a-0 on line G by applying a logic value 1 in line G

Propagate Error To Primary Output Y



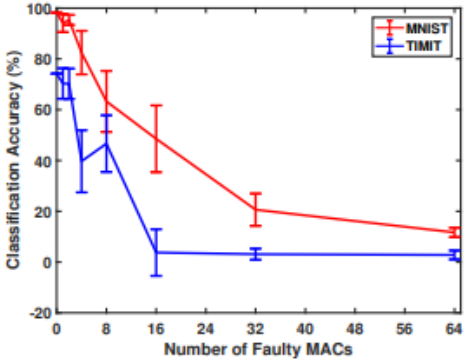
Test Vector A,B,C = 1,0,0 detects fault G s-a-0

Stuck-At Fault as a Logic Fault

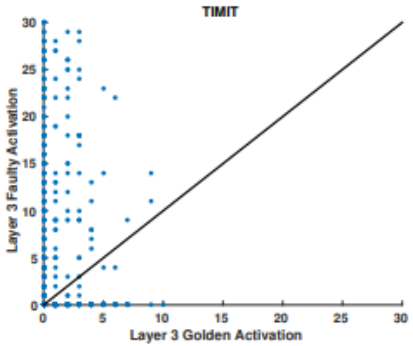
- **Stuck-at Fault is a *Functional Fault* on a Boolean (Logic) Function Implementation**
- **It is not a Physical Defect Model**
  - Stuck-at 1 does not mean line is shorted to V_{DD}
  - Stuck-at 0 does not mean line is grounded!
- **It is an abstract fault model**
  - A logic stuck-at 1 means when the line is applied a logic 0, it produces a logical error
  - A logic error means 0 becomes 1 or vice versa

Ref: https://web.stanford.edu/class/ee386/public/stuck_at_fault_6per_page

Impact of Permanent Faults on TPU



(a) Classification Accuracy Drop Due to Stuck-at-Fault MACs.



(b) TIMIT Output Regression for Layer 3 Activation with 8 Faulty MACs.

- For TIMIT, we observe that even with only four faulty MACs (i.e., with only ⇠ 0.005% MACs faulty), the classification accuracy drops from the 74.13% to 39.69%.
- Golden = Fault free

stuck-at faults frequently affect the higher order bits of the MAC output, resulting in large absolute errors in the matrix vector product.

Solution: Fault-Tolerant TPU Design

Motivation

TPUs with even a relatively small number of faulty MAC units cannot be used unless the fault impact is mitigated.

Observation:

Each weight in the DNN maps to exactly one MAC unit.

In other words, there is a static mapping between DNN weights and MAC units. We then exploit the static mapping to determine which weights to prune.

Mapping in Submatrices

- mapping functions r() and c() that take as input the indices of a DNN weight and output the row and column, respectively, of the MAC unit on which the weight is mapped.
- The mapping functions for weight wi,j in a fully-connected layer:

$$r(i, j) = j\%N \text{ and } c(i, j) = i\%N, \quad \% \text{ is the modulo operator.}$$

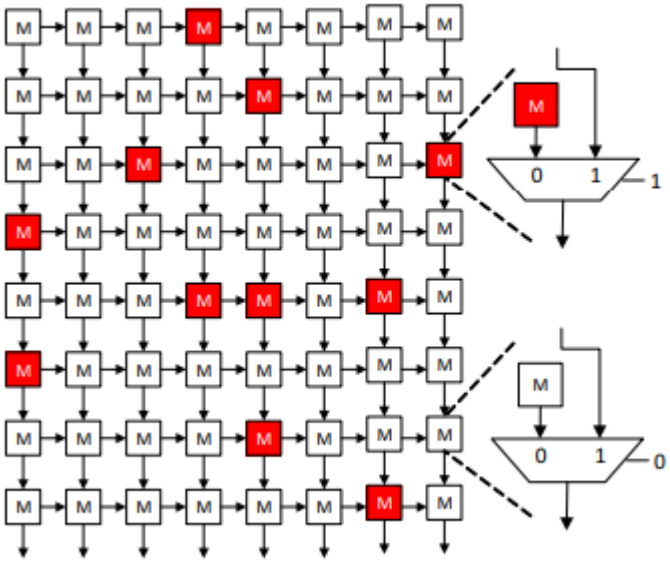
- weight matrices that do not fit fully in the systolic array are first blocked into smaller N ⇥ N sub-matrices

Two Solutions

fault-aware pruning (FAP)

fault-aware pruning plus retraining (FAP+T)

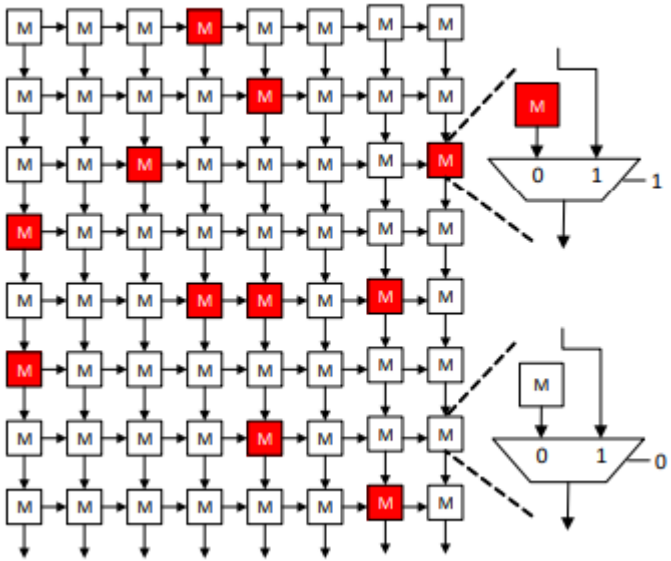
FAP



- The proposed FAP and FAP+T techniques both assume that standard post-fabrication tests are used on each TPU chip to determine the location of faulty MACs.
- Given this information, FAP is to set (or prune) any weight that maps to a faulty MAC to zero.
- That is, for all pairs of (i, j) values such that MACc(i,j),r(i,j) is faulty, we set the corresponding wi,j = 0
- this is for fully connected layers, a similar strategy is used for conv layers.
- multiple weights can map to one MAC unit; correspondingly, even a single faulty MAC can result in multiple weights being pruned

Pruning in hardware

- In hardware, pruning is achieved by introducing a separate bypass path for faulty MAC units.
- With the bypass path being enabled, the faulty MAC unit's contribution to the column sum is skipped, which is equivalent to setting the faulty MAC's weight to zero.
- The area overhead due to the new bypass path is only about 9%.



FAP + T

Algorithm 1: FAP+T Training Algorithm

```
1 Algorithm FAP+T()
2 Load the pre-trained DNN weights and TPU fault
    map;
3 Determine indices of pruned weights from TPU fault
    map;
4 Set all pruned weights to zero;
5 for Training epochs ≤ MAX_EPOCHS do
6 |   Update weights using back-prop.;
7 |   Set all pruned weights to zero;
8 end
9 return Retrained model;
```

- The FAP+T approach starts with FAP based on each TPU's fault map, but additionally retrains the unpruned weights in the DNN while forcing all pruned weights to zero during the re-training process.

- returns new, optimized values for the unpruned weights that improve the classification accuracy compared to the FAP solution.

Limitation

- re-training needs to be performed for each TPU chip based on its unique fault map;
- however, this needs to be done only once per TPU chip and the cost of doing so is amortized over the entire lifetime of the TPU chip.

MAX EPOCHS : determines the number of iterations of the re-training algorithm.

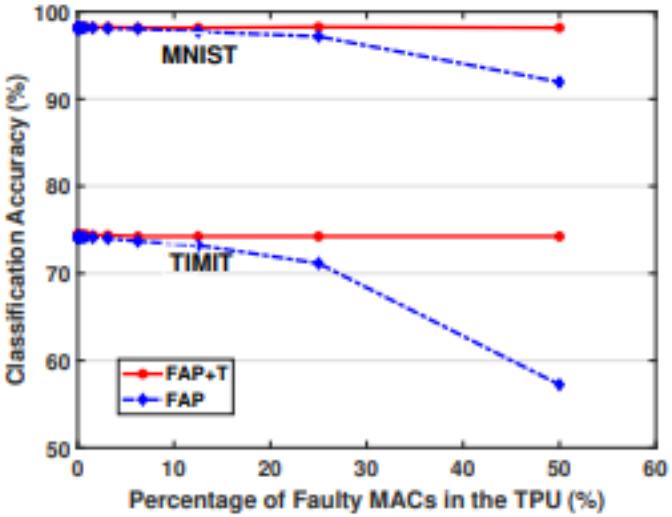
Setting this parameter to zero is equivalent to FAP.

As MAX EPOCHS is increased, the re-training time increases in return for increased classification accuracy

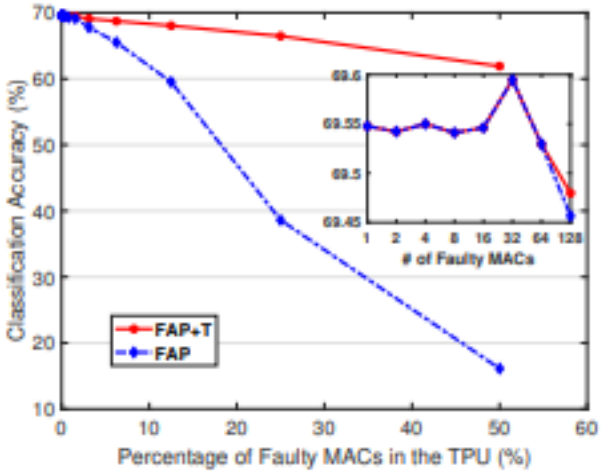
Algorithm 1: FAP+T Training Algorithm

```
1 Algorithm FAP+T()
2 Load the pre-trained DNN weights and TPU fault
    map;
3 Determine indices of pruned weights from TPU fault
    map;
4 Set all pruned weights to zero;
5 for Training epochs ≤ MAX_EPOCHS do
6 |   Update weights using back-prop.;
7 |   Set all pruned weights to zero;
8 end
9 return Retrained model;
```

Result



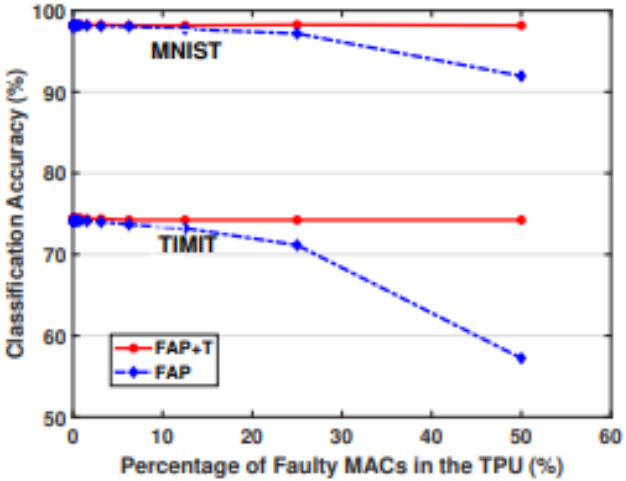
(a) MNIST and TIMIT.



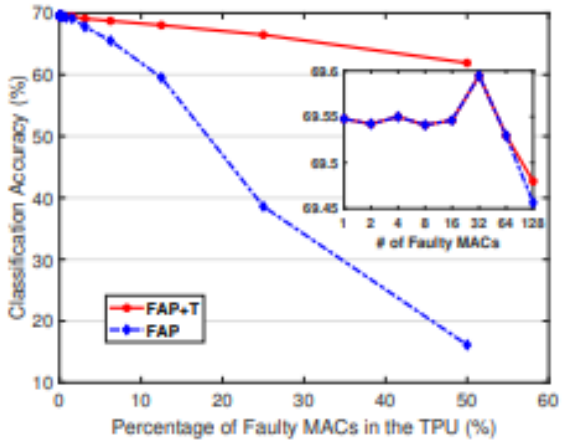
(b) AlexNet.

TPUs operate even with fault rates as high as 50%, with negligible to tolerable drops in classification accuracy (from 0.1% drop for TIMIT to 8% drop for AlexNet).

Result



(a) MNIST and TIMIT.



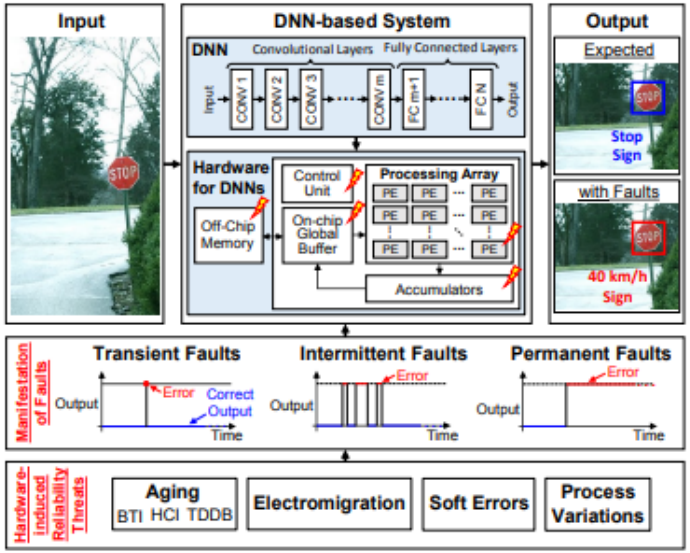
(b) AlexNet.

- **retraining overhead of about 12 minutes per TPU chip for AlexNet. However, this one-time cost is amortized over the entire TPU's lifetime**

Other relevant faults in NN

[Ref: Dependable Deep Learning: Towards Cost-Efficient Resilience of Deep Neural Network Accelerators against Soft Errors and Permanent Faults(https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9159734)]

Types of faults



- A transient fault is a **fault that is not long and can be restored.**
- An intermittent fault, often called simply an "intermittent", is **a malfunction of a device or system that occurs at intervals, usually irregular.**
- Permanent faults are **failures that manifest as stuck-at bits in the architecture**, that is, lines that always carry the logical signal “0” or “1” as the result of a short or open circuit.

Soft Errors

Soft Errors are **transient faults** that **manifest as bit flips** and result in data corruption. These are mainly caused by extrinsic sources like alpha particles emitted from the impurities in packaging materials or by neutrons from cosmic radiations when they strike the chip.

Temperature is another factor that can result in an increase in the Soft Error Rate (SER).

Aging

Aging of electronic circuits occurs due to various physical phenomena such as Bias Temperature Instability (BTI), Hot Carrier Injection (HCI), Time-Dependent Dielectric Breakdown (TDDB) and Electromigration (EM).

- It typically results in circuits becoming slower with time, e.g., by increasing the threshold voltage (VTH) of the circuits, or breakdown of dielectric and wires.
- The **aging faults manifest as timing errors** during the early stages, and later can transform into permanent faults. Alongside various other factors, the rate of aging usually increases with temperature.

Process Variations

Process Variations is an issue caused by the variations **introduced during the manufacturing process**.

This issue arises due to the fact that it is difficult to manufacture transistors with the same properties, such as channel length, oxide thickness and doping levels, at the nano-scale.

results in performance (e.g., in the form of reduced operating frequency) and power efficiency, as well as the yield of the manufacturing process (in case of permanent faults).

COST-EFFECTIVE RESILIENCE

Technique	Abstraction Layer	Related Work	Brief Description	Cost	Targeted Faults	Other Dependencies
Fault-Aware Training	SW	[22] [23]	It incorporates the information of hardware-level faults in the training process to enable the network to adapt accordingly and produce correct output using faulty hardware during inference.	Design-time: High	Permanent faults in memory	Fault map extraction
Fault-Aware Pruning	HW Architecture + SW	[24] [23] [25]	This technique leverages the inherent redundancy in DNNs and propose to drop the faulty computations to maintain the accuracy of the provided DNN. In case of higher number of faults, this technique is coupled with fault-aware training to offer optimal performance.	Run-time: Low	Permanent faults in computational array	Modifications in HW architecture + fault map extraction
Fault-Aware Mapping	HW Architecture + SW	[24]	Fault-aware mapping leverages the fact that different computations/parameters have different level of significance and pruning the least significant set of computations/parameters during fault-aware pruning can have the least impact on the accuracy of the DNN-based system. Therefore, it proposes to find the least significant set and, with the help of mapping, map them to the faulty components	Design-time: Low	Permanent faults in computational array	Fault map extraction
Range Restriction	SW	[26] [27]	In DNNs, faults in sensitive computations usually result in abnormal output at an intermediate stage in the network that can propagate to the output to produce incorrect result. Range restriction techniques propose to determine ranges of intermediate outputs and identify any output that does not fall in the defined range as faulty and map them to some in the range using some pre-defined mechanism.	Design-time: Low Run-time: Low	Transient Faults (i.e., Soft Errors)	Modifications in HW architecture in case of specialized HW
Algorithm-based Fault Tolerance	SW	[28]	Algorithm-based Fault Tolerance (ABFT) techniques that are based on checksum computation are commonly used to protect matrix-matrix multiplications. These techniques have been extended to offer cost-effective error detection and correction for convolution operations as well.	Run-time: Low	Transient faults in computational units and data corruption	No
Radiation Hardening	Circuit	[29] [30]	This approach enables us to make the hardware less prone to radiation-induced faults, using stronger cells or through designing the hardware components such that they are biased towards a specific type of faults towards which DNNs are more resilient. For example, designing SRAM cells such that, in case of radiation-induced faults, they result in a '0' more often then a '1'.	Run-time: Low	Transient Faults (i.e., Soft Errors)	No
Redundancy	HW Architecture + SW	[31] [32]	It introduces selective spatial and temporal redundancy to detect and rectify errors in critical computations during inference.	Run-time: Moderate to High	Transient, intermittent and permanent faults	No

SalvageDNN: salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping

https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2019.0164

Limitation of FAP+T

1. The **additional circuitry** used for bypassing the MAC units is faulty.

1. The number of chips manufactured is large and the number of computational units in the DNNaccelerator design is also significant. **With an increase in the size of the systolic array, the number of possible fault maps increases significantly.** Inthis case, fine-tuning/training a neural network for each faulty chip seems impractical,

Limitation of FAP+T

3. The training dataset is not available, e.g. in a case where a company has released a neural network model, but they have not made the training data available.

4. The above method will lack efficiency in case the number of permanent faults changes over the lifetime of the chip, e.g. due to ageing.

Saliency evaluation of neurons/filters of a DNN

- **The saliency of a network parameter defines its importance based on its contribution and/or the expected impact it can have on the output.**

Widely used methods:

1. **L1 and L2 norms, where the norms of the parameters define their saliency:**

   [L1 Norm is the sum of the magnitudes of the vectors in a space.

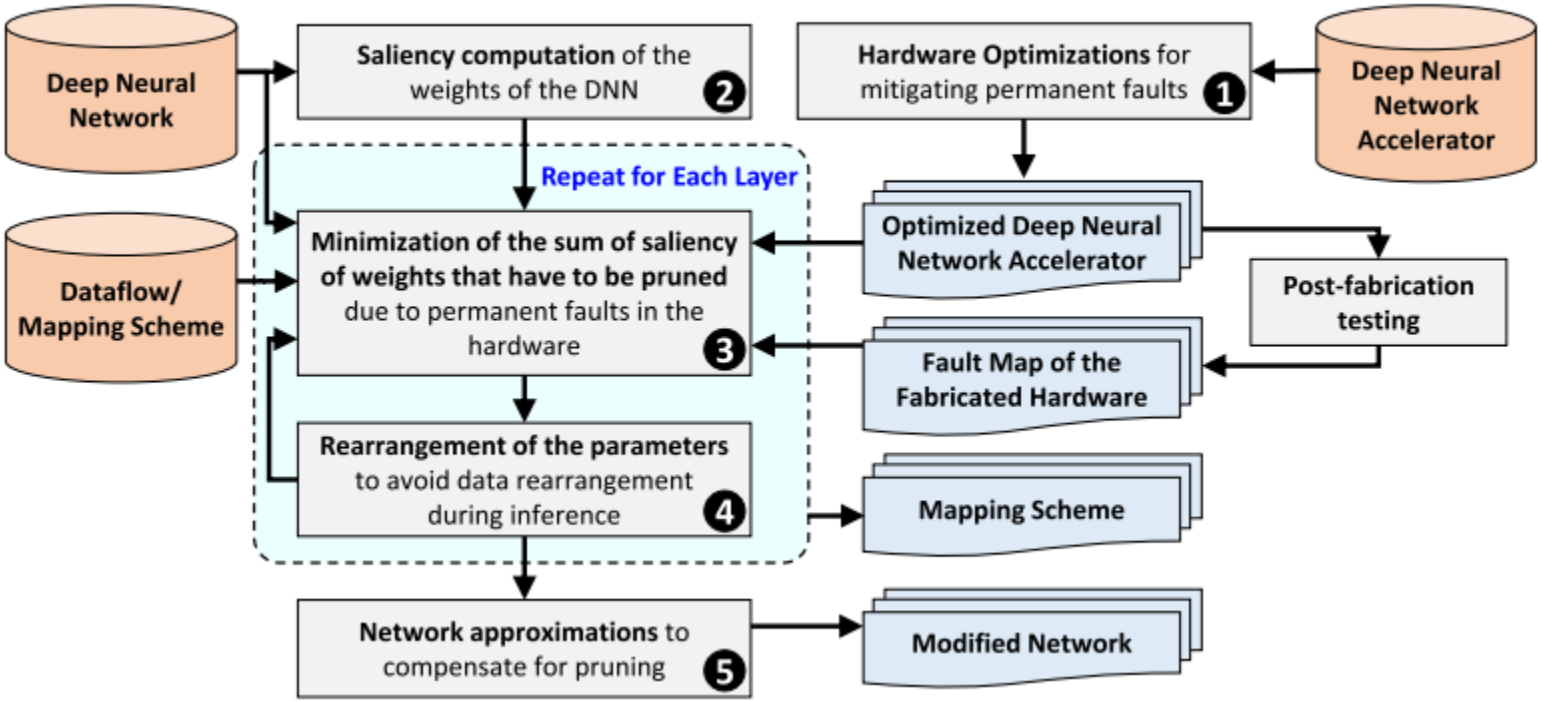
   L2 norm:Euclidean norm]

Saliency evaluation of neurons/filters of a DNN

2. Neuron importance score computation based on propagation .

[Such methods are more accurate but computationally more intensive than norm based methods, as they have to back propagate from the output to a particular neuron/filter to compute its saliency.]

$$s_i^{<l>} = \sum_k |W_{(k,i)}^{<l+1>}| \times s_k^{<l+1>}$$

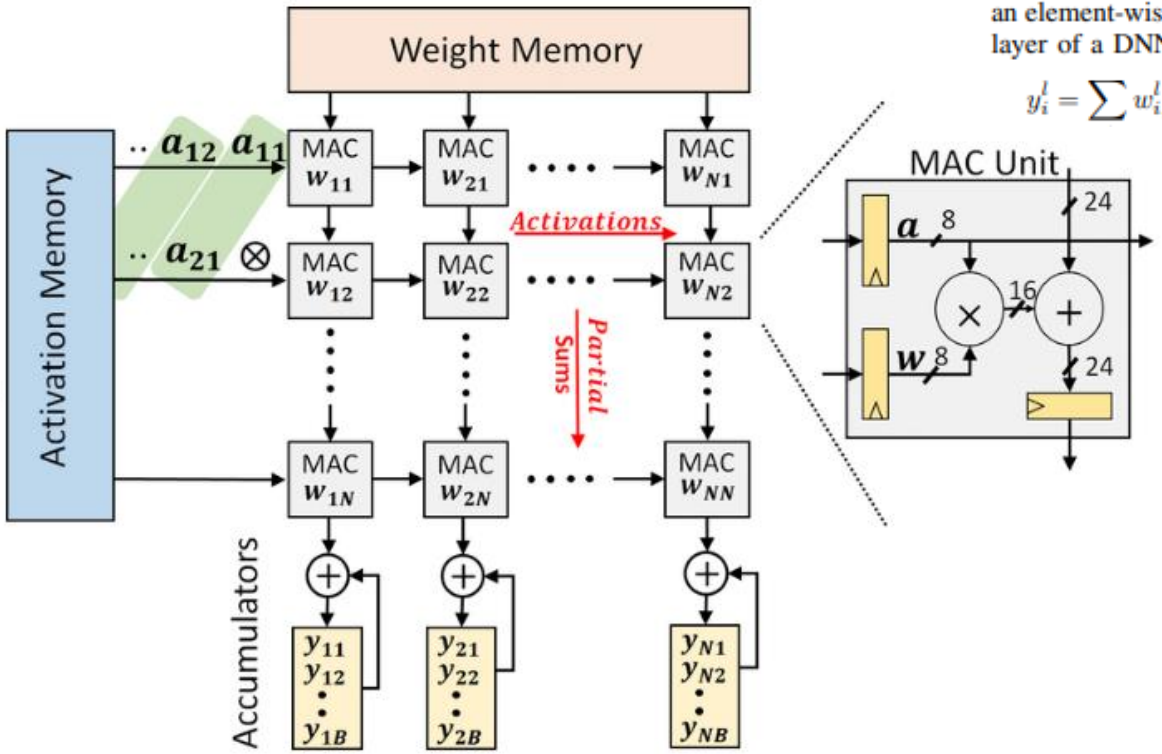


Pruning is **a process of removing weights which connect neurons from two adjacent layers in the network**.

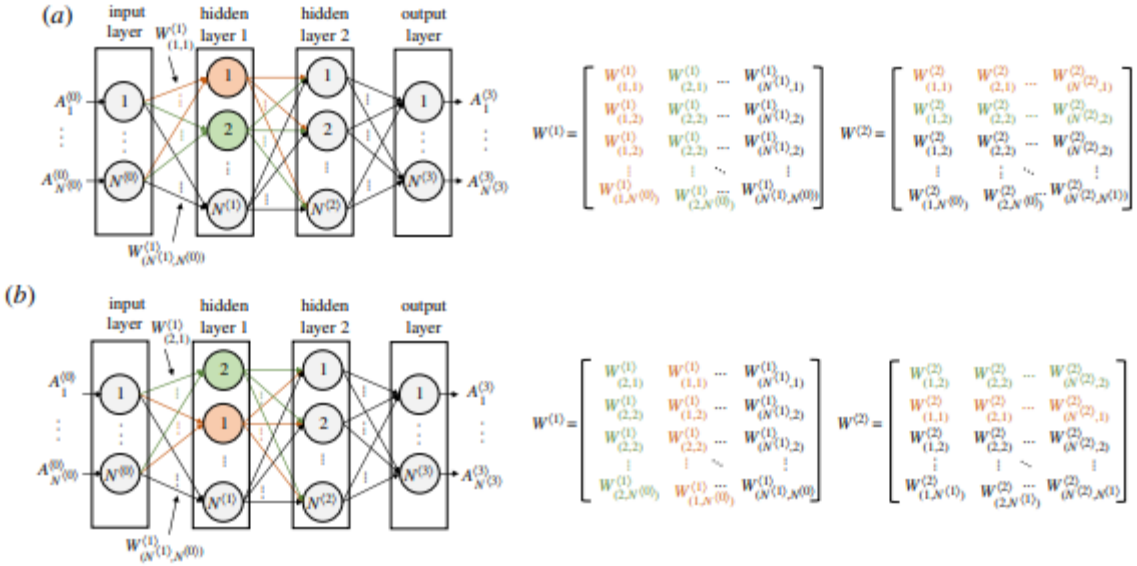
Why Rearrangement?

A DNN consists of L stacked layers of computation, Layer l has N^l *neurons* whose outputs are referred to as *activations*, represented by an N^l dimensional vector a^l. Each layer multiplies the vector of activations from the previous layer with a weight matrix w^l of dimensions $N^{l-1} \times N^l$ and adds constant biases represented by an N^l dimensional vector b^l, followed by an element-wise activation function ϕ. Mathematically, each layer of a DNN performs the following operation:

$$y_i^l = \sum w_{i,j}^l a_j^{l-1} + b_i^l, \quad a_i^l = \phi(y_i^l) \quad \forall l \in [1, L],$$

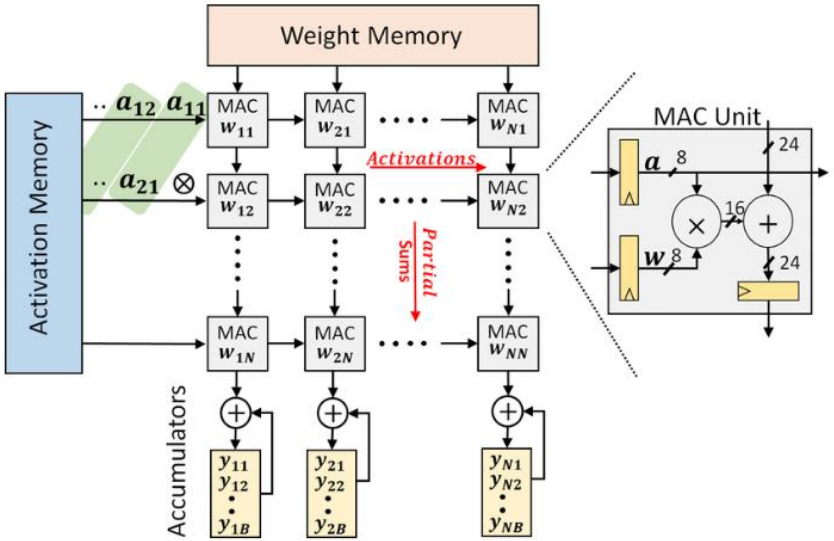


Why Rearrangement?



Impact of rearranging neurons in a layer of a fully-connected DNN on the arrangement of the weights to be mapped on the systolic array. (*a*) Shows the arrangement before swapping neurons 1 and 2 in the first hidden layer of a fully-connected DNN and (*b*) shows the arrangement after swapping the neurons. The left side of the figure illustrates the state of the neural network and the right side shows the weights of the first and second hidden layers in a manner in which they will be mapped

Why Rearrangement?

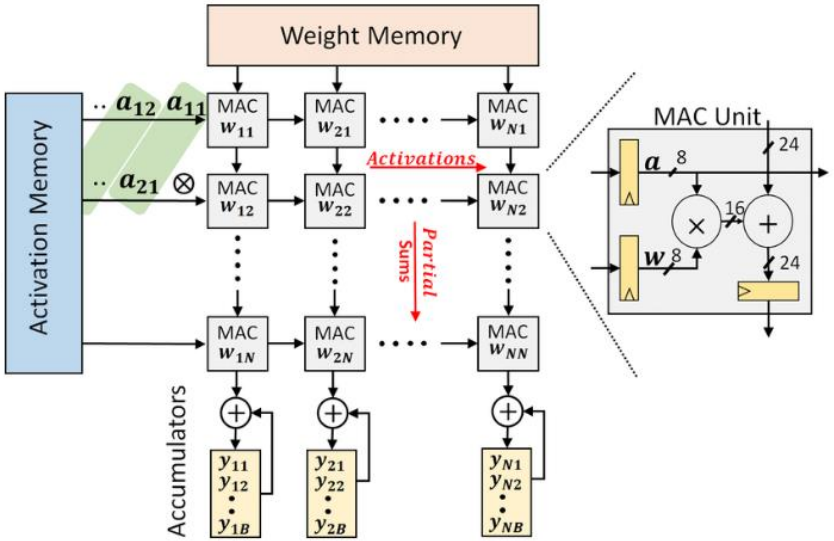


- Assume that w_ij (a_ij) is the element in the i-th row and j-th column of weight (activation) matrixW (A).

The operation of the systolic array can be explained as follows:

- First, weights are pre-loaded into the array and remain stationary throughout a block of computation.
- MAC unit that stores w_ij as MAC_ij.
- Next, activations stream in from the activation memory, one activation per row per clock cycle, and move from left to right.

Why Rearrangement?



Rearrangement due to change of w_ij locations.

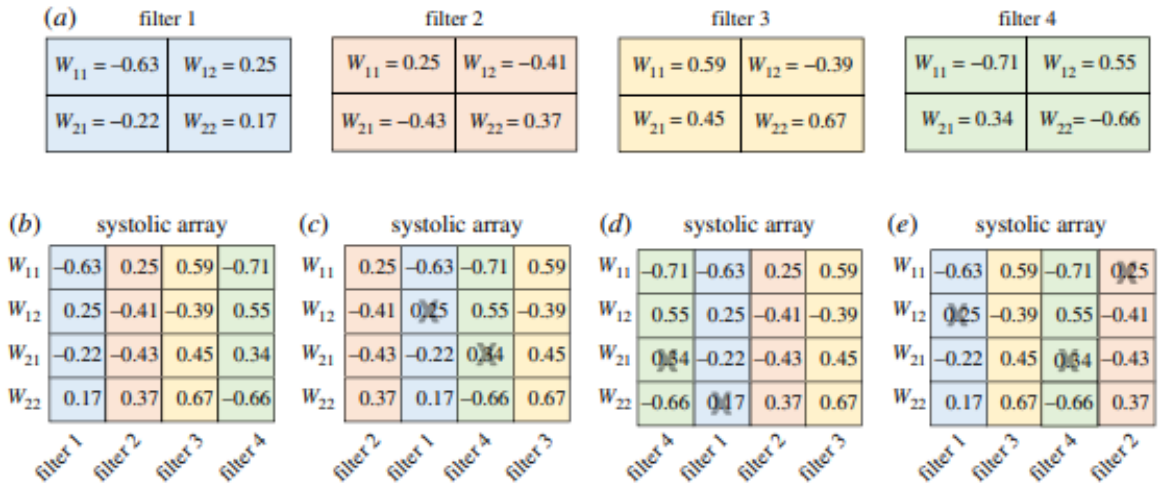
- MAC_11 computes w_11*a_11 in the first clock cycle,
- MAC_12 unit adds w_12*a_21 to MAC11's product in the next clock cycle, and so on.

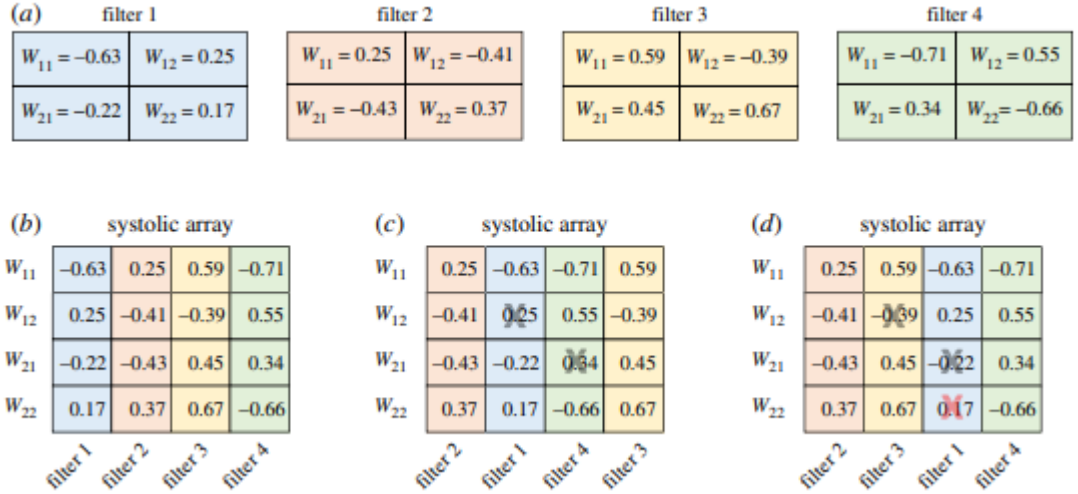
- In clock cycle N, MAC_1N unit outputs:
  y_11 = ∑ w_1i*a_i1,
  [the first element of the output matrix]

- MAC_1N proceeds to output y_12, . . . ,y_1B in subsequent clock cycles.

- The second column receives the same stream of inputs as the first column, but delayed by one clock cycle.
- This column outputs y_21,y_22, . . . ,y_2B.
- In this manner, a batch of B inputs is multiplied by an N × N weight matrix in 2N + B clock cycles.

— Case 1: None of the PEs in the systolic array is faulty (figure 9*b*).
— Case 2: The array has two faulty PEs (figure 9*c*).
— Case 3: The array has the same number of faulty PEs as in Case 2, but at different locations (figure 9*d*).
— Case 4: The array has different number of faulty PEs than Case 2 and Case 3, and also at different locations (figure 9*e*).





An example illustration of how the mapping would change if the fault map of the DNN hardware changes over time. The grid shown in *b*, *c* and *d* corresponds to a systolic array, where each small box represents a single processing element (PE). The PEs with black cross over them in (*c*) and (*d*) correspond to faulty PEs detected during post-fabrication testing and the PE with red cross over it in (*d*) corresponds to the PE which experienced fault over time due to wear-out. (*a*) Filters, (*b*) Case 1, (*c*) Case 2, and (*d*) Case 3.

Advantage

Saliency-driven fault-aware mapping, without requiring extensive retraining as typically done in the state of the art.

Steps in the Algorithm

1. The DNN accelerator design is optimized such that the components having permanent faults can be disconnected from the main datapath to mitigate the effects of the faults.

2 Saliency of the weights of the DNN is computed.

3. Given a dataflow, the architectural characteristics of the modified accelerator design and the fault map of the fabricated hardware, mapping of neurons/filters of a layer of the DNN on different segments of the hardware such that the sum of saliency of the weights that are pruned (mapped on the faulty/disconnected parts of the datapath) during inference is minimized.

4. makes the required rearrangements in the DNN such that the data rearrangements, which are highly memory intensive, are not required during the DNN processing.

Steps

- Steps 3 and 4 are repeated for each layer and the resultant network is forwarded to step 5 after setting all the weights which are to be mapped on the faulty/disconnected computational units to zero.
- The network is traversed in a sequential order from the first layer to the last.
- Step 5 , required adjustments are made to the network parameters to compensate for the pruned weights/computations.

Minimization of the sum of saliency

Algorithm 1 A fast method to reduce the sum of saliency of the weights of a layer that have to be pruned due to permanent faults.

Inputs: A matrix ***Costs*** containing the costs of mapping neurons/filters to faulty columns of the array and a vector ***Idx*** containing the indexes of faulty columns of the array
Outputs: A matrix ***Mapping*** which defines which neuron/filter has to be mapped to which faulty column and a variable *TCost* which defines the cost of this mapping scheme
Initialize: $TCost = 0$, ***Mapping*** = Zeros(Number of Columns in ***Costs***, 2) and ***NF_Idx*** = [1, 2, 3, . . . Number of neurons/filters in the layer]

1: **for** $i = 1$ to Number of columns in ***Costs*** at the start of the loop **do**
2: [*val*, *row_id*, *col_id*] = min(***Costs***)
3: ***Mapping***(*i*,1) = ***NF_Idx***(*row_id*)
4: ***Mapping***(*i*,2) = ***Idx***(*col_id*)
5: *TCost* = *TCost* + *val*
6: ***Costs*** ⇐ ***Costs*** after removing *row_id* row and *col_id* column
7: ***Idx*** ⇐ ***Idx*** after removing *col_id* column
8: ***NF_Idx*** ⇐ ***NF_Idx*** after removing *row_id* column
9: **end for**
10: **return** *TCost* and ***Mapping***

- Generation of the Disconnection Map (DM).

- The DM is a matrix that defines the MAC units which will be disconnected from the array during processing. This is done by setting the values corresponding to the MACs which will be disconnected in DM to 1.

- [For example,if the MAC unit in the first row and first column of the array will be disconnected during processing, the DM(1, 1) is set to 1; otherwise it will be 0.]

- The DM is constructed using the fault map of the array. The fault map is represented using two matrices FMMAC and FMMUX, where FMMAC keeps track of the faulty MAC units and FMMUXkeeps track ofthe faulty multiplexers in the array.

- **Unrolling of the Saliencies of Weights of a Layer in a Matrix S**. The matrixS represents the matrix containing the saliency of neurons/filters of a layer in flattened form.

- **Generation of the Pruning Matrix (PM):**
  - The DM matrix is replicated in x and y dimensions such that the number of columns and rows in the final matrix is at least equivalent to the number of columns and rows in S.
  - The additional columns (from the right) and rows(from the bottom) are then removed, and the matrix is stored in the Pruning Matrix (PM).
  - The columns containing all zeros are removed from the PM while keeping track of the original indexes of the non-zero columns in a vector Idx.

- **Minimization of the Sum of Saliency of the Weights to be Pruned**:

The objective of this step is to generate a mapping strategy which minimizes the sum of saliency of weights to be pruned.
It can mathematically be represented as:

$$\text{argmin}_{\text{mapping}} ||S^*. * \text{PM}||.$$

Here,S* represents a transformed version of S generated after applying rearrangement of network parameters based on the mapping strategy.

Algorithm 1 A fast method to reduce the sum of saliency of the weights of a layer that have to be pruned due to permanent faults.

Inputs: A matrix ***Costs*** containing the costs of mapping neurons/filters to faulty columns of the array and a vector ***Idx*** containing the indexes of faulty columns of the array
Outputs: A matrix ***Mapping*** which defines which neuron/filter has to be mapped to which faulty column and a variable *TCost* which defines the cost of this mapping scheme
Initialize: $TCost = 0$, ***Mapping*** = Zeros(Number of Columns in ***Costs***, 2) and ***NF_Idx*** = [1, 2, 3, . . . Number of neurons/filters in the layer]

1: **for** $i = 1$ to Number of columns in ***Costs*** at the start of the loop **do**
2: [*val*, *row_id*, *col_id*] = min(***Costs***)
3: ***Mapping***(*i*,1) = ***NF_Idx***(*row_id*)
4: ***Mapping***(*i*,2) = ***Idx***(*col_id*)
5: *TCost* = *TCost* + *val*
6: ***Costs*** ⇐ ***Costs*** after removing *row_id* row and *col_id* column
7: ***Idx*** ⇐ ***Idx*** after removing *col_id* column
8: ***NF_Idx*** ⇐ ***NF_Idx*** after removing *row_id* column
9: **end for**
10: **return** *TCost* and ***Mapping***

- sub-optimal mapping strategy.
- Input:
  - Costsmatrix: the costs of mapping each neuron/filter to each faultycolumn of the systolic array, and
  - Idxvector: contains the indexes of the faulty columns of the array, as inputs.

TheCost matrix is computed using matrix multiplication of transpose of S with PM.

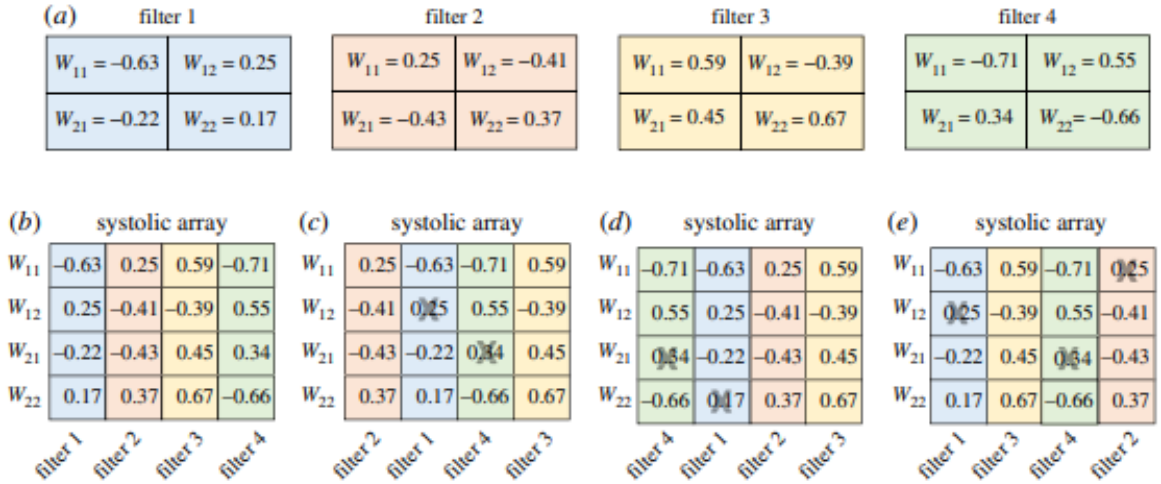
Algorithm 1 A fast method to reduce the sum of saliency of the weights of a layer that have to be pruned due to permanent faults.

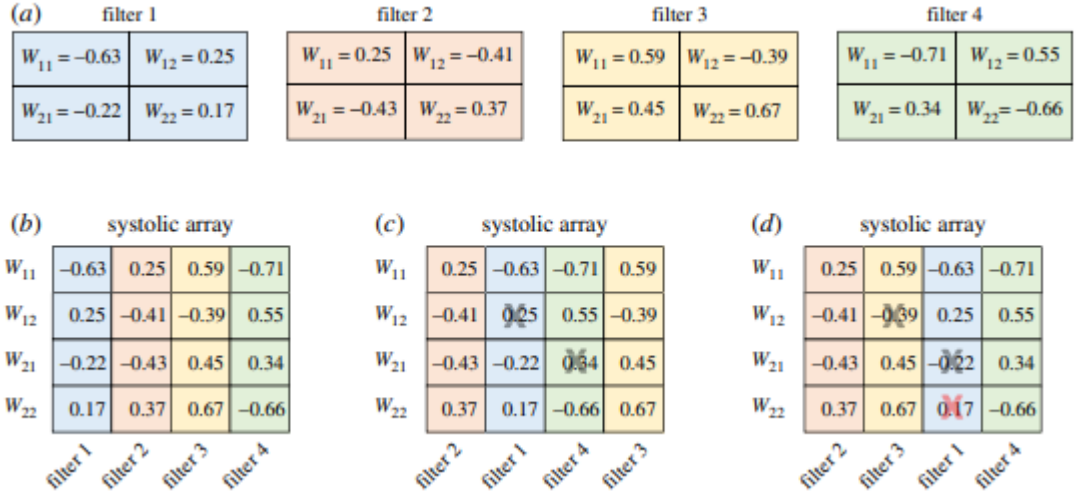
Inputs: A matrix ***Costs*** containing the costs of mapping neurons/filters to faulty columns of the array and a vector ***Idx*** containing the indexes of faulty columns of the array
Outputs: A matrix ***Mapping*** which defines which neuron/filter has to be mapped to which faulty column and a variable *TCost* which defines the cost of this mapping scheme
Initialize: $TCost = 0$, ***Mapping*** = Zeros(Number of Columns in ***Costs***, 2) and ***NF_Idx*** = [1, 2, 3, . . . Number of neurons/filters in the layer]

1: **for** i = 1 to Number of columns in ***Costs*** at the start of the loop **do**
2: [*val*, *row_id*, *col_id*] = min(***Costs***)
3: ***Mapping***(i,1) = ***NF_Idx***(*row_id*)
4: ***Mapping***(i,2) = ***Idx***(*col_id*)
5: *TCost* = *TCost* + *val*
6: ***Costs*** $\Leftarrow$ ***Costs*** after removing *row_id* row and *col_id* column
7: ***Idx*** $\Leftarrow$ ***Idx*** after removing *col_id* column
8: ***NF_Idx*** $\Leftarrow$ ***NF_Idx*** after removing *row_id* column
9: **end for**
10: **return** *TCost* and ***Mapping***

Revisit The Example

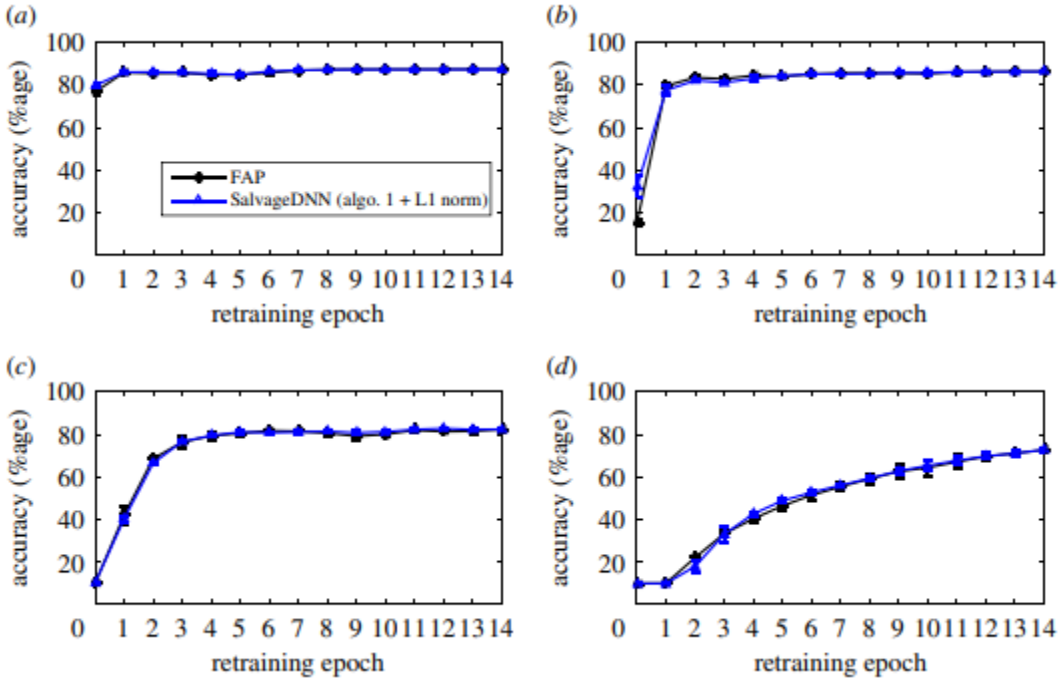
— Case 1: None of the PEs in the systolic array is faulty (figure 9*b*).
— Case 2: The array has two faulty PEs (figure 9*c*).
— Case 3: The array has the same number of faulty PEs as in Case 2, but at different locations (figure 9*d*).
— Case 4: The array has different number of faulty PEs than Case 2 and Case 3, and also at different locations (figure 9*e*).





An example illustration of how the mapping would change if the fault map of the DNN hardware changes over time. The grid shown in *b*, *c* and *d* corresponds to a systolic array, where each small box represents a single processing element (PE). The PEs with black cross over them in (*c*) and (*d*) correspond to faulty PEs detected during post-fabrication testing and the PE with red cross over it in (*d*) corresponds to the PE which experienced fault over time due to wear-out. (*a*) Filters, (*b*) Case 1, (*c*) Case 2, and (*d*) Case 3.

Result



Impact of using SalvageDNN before applying fault-aware training on the accuracy of the VGG11 network trained for the Cifar10 classification. The subfigures show the comparison between FAP + retraining and SalvageDNN when the underlying hardware has: (*a*) 10% faulty PEs, (*b*) 30% faulty PEs, (*c*) 50% faulty PEs, and (*d*) 70% faulty PEs.