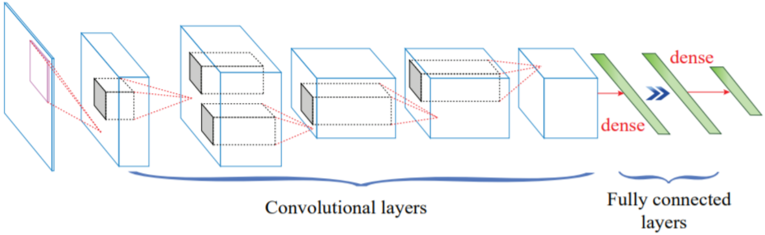
# CirCNN: Accelerating and Compressing Deep Neural Networks Using Block-Circulant Weight Matrices

Vladislav Lukoshkin
Temirlan Seilov
Mariya Zharikova

Convolution Neural Networks structure:

- The fully-connected (FC) layer
- The convolutional (CONV) layer
- The pooling (POOL) layer



# Advantages over conventional neural networks

- storage size reduction complexity is reduced from $O(n^2)$ to $O(n)$
- computational complexity reduction from $O(n)$ to $O(n \log n)$

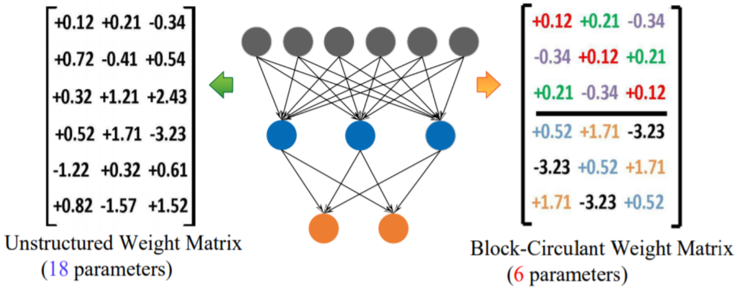


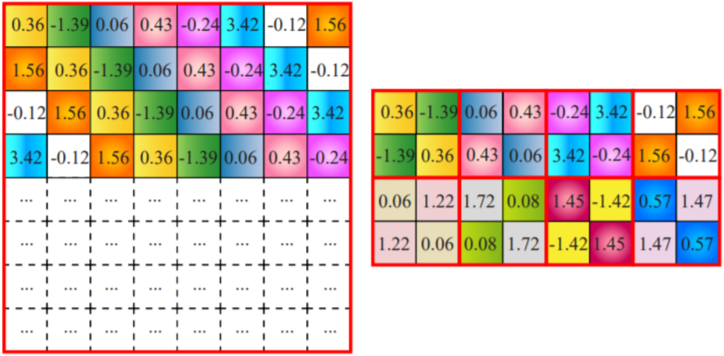

**Figure 1: Block-circulant Matrices for weight representation.**

# Benefits CircNN.

- current approaches apply various compression techniques (e.g., pruning) on the unstructured weight matrices and then retrain the network; while CirCNN directly trains the network assuming block-circulant structure.
- CirCNN provides the adjustable but fixed reduction ratio, while prior work can only reduce the model size by a heuristic factor,depending on the network.
- with the same FFT-based fast, multiplication, the computational complexity of training is also reduced from $O(n^2)$ to $O(n \log n)$,

# Benefits of block-circulant matrices

- applied both to FC layers and CONV layer
- no storage waste – do not need to be square and padd zeros
- a fine-grained tradeoff between accuracy and compression/acceleration




(a) Baseline (b) Proposed

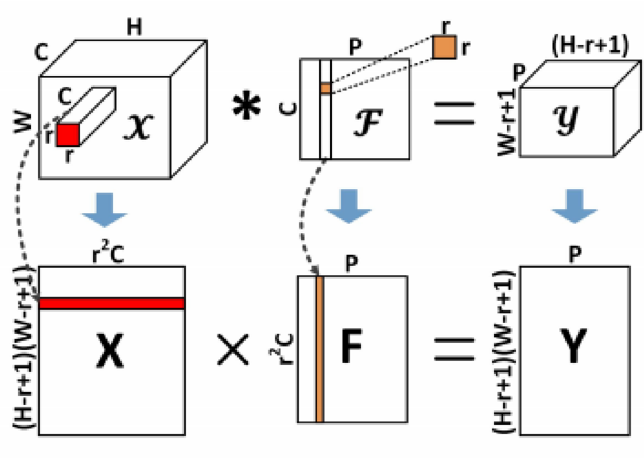
**Algorithm 1:** Forward propagation process in the FC layer of CirCNN

**Input:** $\mathbf{w}_{ij}$'s, $\mathbf{x}$, $p$, $q$, $k$
**Output:** $\mathbf{a}$
Initialize $\mathbf{a}$ with zeros.
**for** $i \leftarrow 1$ *until* $p$ **do**
    **for** $j \leftarrow 1$ *until* $q$ **do**
        $\mathbf{a}_i \leftarrow \mathbf{a}_i + \mathrm{IFFT}(\mathrm{FFT}(\mathbf{w}_{ij}) \circ \mathrm{FFT}(\mathbf{x}_j))$
    **end**
**end**
**return** $\mathbf{a}$

# Convolutional Layer in CirCNN

$\mathcal{Y}(\mathrm{x},\mathrm{y},\mathrm{p}) = \sum_{\mathrm{i}=1}^{\mathrm{r}} \sum_{\mathrm{j}=1}^{\mathrm{r}} \sum_{\mathrm{c}=1}^{\mathrm{C}} \mathcal{F}(\mathrm{i},\mathrm{j},\mathrm{c},\mathrm{p}) \mathcal{X}(\mathrm{x}+\mathrm{i}-1,\mathrm{y}+\mathrm{j}-1,\mathrm{c})$
where $\mathcal{F} \in \mathbb{R}^{\mathrm{r}\times\mathrm{r}\times\mathrm{c}\times\mathrm{p}}, \mathcal{Y} \in \mathbb{R}^{(\mathrm{W}-\mathrm{r}+1)\times(\mathrm{H}-\mathrm{r}+1)}, \mathcal{X} \in \mathbb{R}^{\mathrm{W}\times\mathrm{H}\times\mathrm{C}}$
Assume that $\mathcal{F}(\cdot,\cdot,\mathrm{c},\mathrm{p})$

# Convolutional Layer in CirCNN



# Results on CIFAR10

```
[1,  2000] loss: 2.011
[1,  4000] loss: 1.813
[1,  6000] loss: 1.734
[1,  8000] loss: 1.695
[1, 10000] loss: 1.649
[1, 12000] loss: 1.642
[2,  2000] loss: 1.594
[2,  4000] loss: 1.587
[2,  6000] loss: 1.588
[2,  8000] loss: 1.565
[2, 10000] loss: 1.548
[2, 12000] loss: 1.565
Finished Training
```

```
[1,  2000] loss: 2.249
[1,  4000] loss: 1.929
[1,  6000] loss: 1.703
[1,  8000] loss: 1.597
[1, 10000] loss: 1.511
[1, 12000] loss: 1.453
[2,  2000] loss: 1.392
[2,  4000] loss: 1.358
[2,  6000] loss: 1.331
[2,  8000] loss: 1.303
[2, 10000] loss: 1.293
[2, 12000] loss: 1.252
Finished Training
```

# Accuracy

```
Accuracy of plane : 38 %
Accuracy of   car : 28 %
Accuracy of  bird : 25 %
Accuracy of   cat : 32 %
Accuracy of  deer : 32 %
Accuracy of   dog : 40 %
Accuracy of  frog : 56 %
Accuracy of horse : 57 %
Accuracy of  ship : 70 %
Accuracy of truck : 52 %
```