Axiado CNN Accelerator

Sindhuja Yadiki, Salome Devkule, Raviteja Godugu, Arasch U Lagies

Rev.1: 7/22/2020

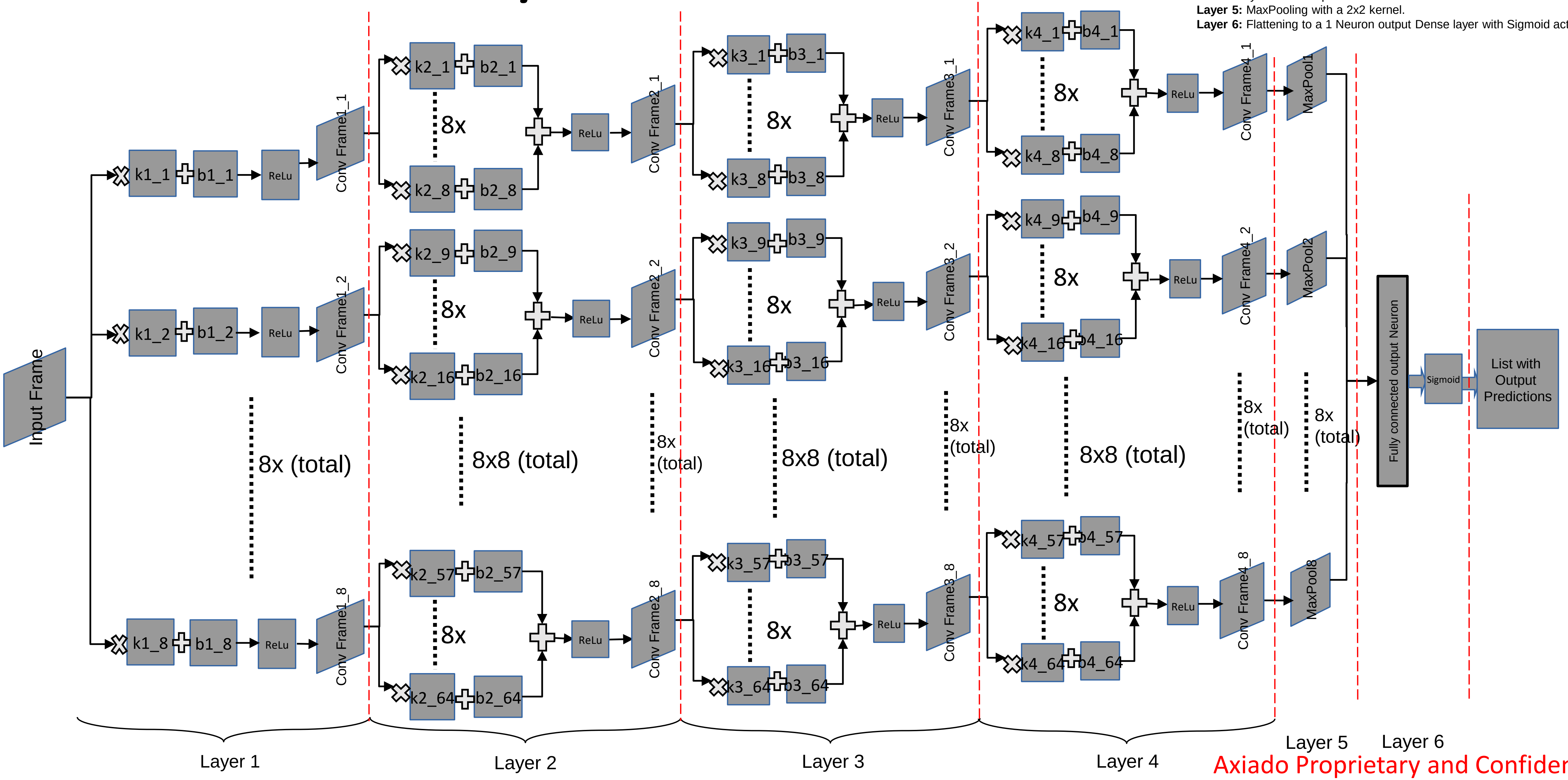
Update on: 9/1/2020

Four Convolution Layer CNN Accelerator

- The Axiado CNN Accelerator has following specifications (see block diagram on next slide):
  - Software-based configuration of **up to 4 Convolution layers**.
  - Software-based configuration of **up to 8 output frames per Convolution layer**.
  - **3x3** kernels per Convolution layer. In later revision planned choice between 3x3 and 5x5 kernels.
  - One MaxPooling layer with (2x2) filter size after all of the Convolution layers.
  - One fully connected output layer with **1 neuron** for classification between **2** classes (e.g. malicious or benign).
  - Predefined **ReLu** (rectified linear unit) activation functions for each Convolution layers.
  - Predefined **Sigmoid** activation function for the output fully connected layer.

Four Conv. Layer CNN Accelerator

Model:
Layer 1: Conv2D, max 8 filter with (3x3) filter-size, stride=1, no padding, followed by ReLu.
Layer 2: Conv2D, max 8x8 filters with (3x3) filter-size, stride=1, no padding, followed by ReLu. 8 output frames.
Layer 3: Conv2D, max 8x8 filters with (3x3) filter-size, stride=1, no padding, followed by ReLu. 8 output frames.
Layer 4: Conv2D, max 8x8 filters with (3x3) filter-size, stride=1, no padding, followed by ReLu. 8 output frames.
Layer 5: MaxPooling with a 2x2 kernel.
Layer 6: Flattening to a 1 Neuron output Dense layer with Sigmoid activation.

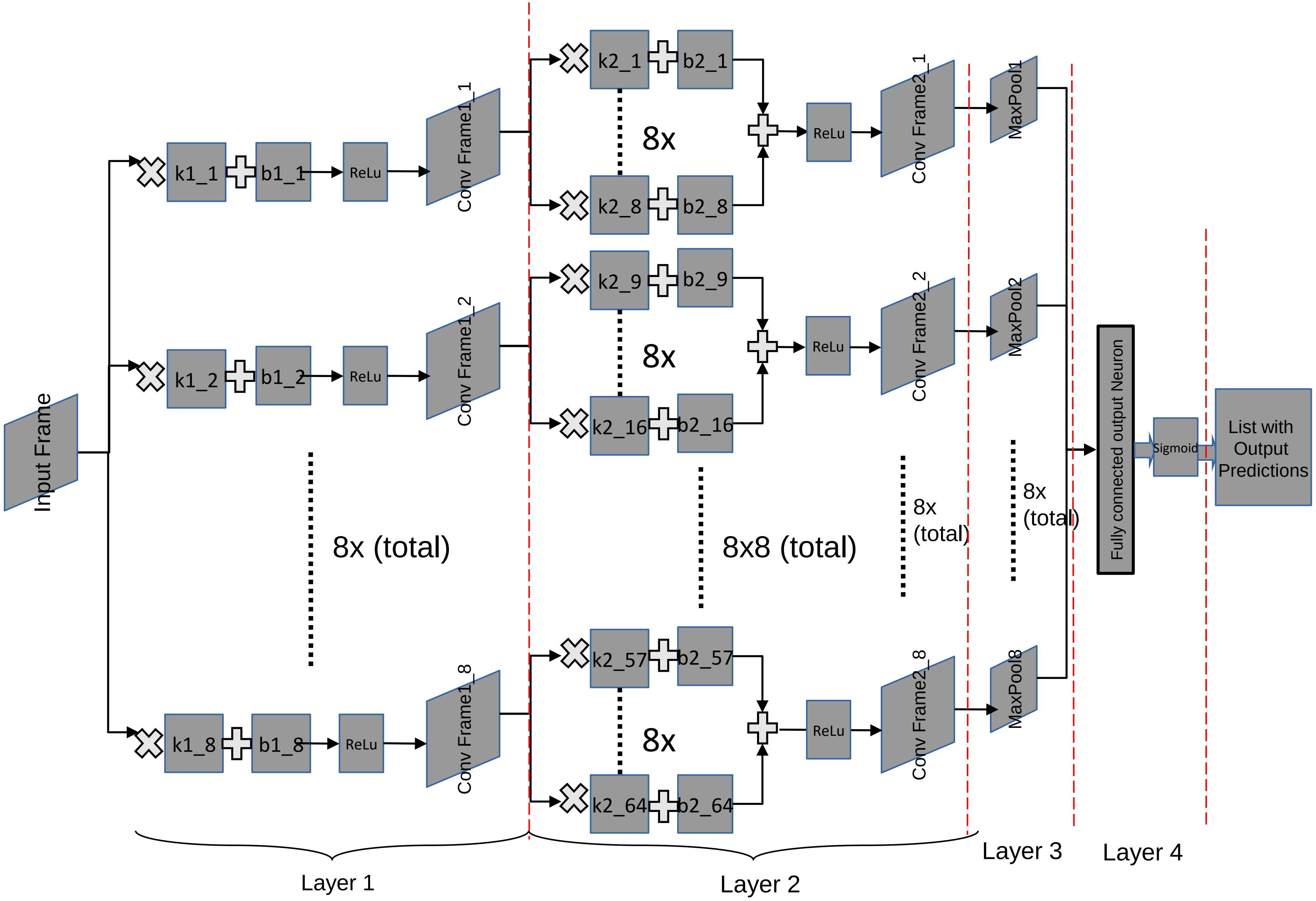


Two Convolution Layer CNN Accelerator Example

- The 4 x Conv2D layer accelerator block diagram is reduced on the following slides to 2 layers for simplified calculations of the required weight and parameter scaling.

Two Conv Layer CNN Model

Model:
Layer 1: Conv2D, max 8 filter with (3x3) filter-size, stride=1, no padding, followed by ReLu.
Layer 2: Conv2D, max 8x8 filters with (3x3) filter-size, stride=1, no padding, followed by ReLu. 8 output frames.
Layer 3: MaxPooling with a 2x2 kernel.
Layer 4: Flattening to a 1 Neuron output Dense layer with Sigmoid activation.



Two Layer CNN

- A floating point convolution between a frame *f* and a kernel k_1 (k_2) of dimensions (*R, C)* can be expressed as
  - Convolution Layer #1:

$$z_1[m,n] = \sigma\big[(k_1 * f)[m,n]\big] = \sigma\left[\sum_{\mathrm{r}=0}^{\mathrm{R}-1}\sum_{\mathrm{c}=0}^{\mathrm{C}-1} k_1\,[\mathrm{r},c]\cdot f\left[m+\mathrm{r}-int\left(\frac{R}{2}\right), n+\mathrm{c}-int\left(\frac{C}{2}\right)\right] + b_1[m,n]\right]$$

  - Convolution Layer #2:

$$z_2[p,q] = \sigma\big[(k_2 * z_1)[p,q]\big] = \sigma\left[\sum_{\mathrm{r}=0}^{\mathrm{R}-1}\sum_{\mathrm{c}=0}^{\mathrm{C}-1} k_2\,[\mathrm{r},c]\cdot z_1\left[p+\mathrm{r}-int\left(\frac{R}{2}\right), q+\mathrm{c}-int\left(\frac{C}{2}\right)\right] + b_2[p,q]\right]$$

Notes:

1. *int* symbolizes conversion of the calculation in braces from float to integer.

2. These formulas are for the case that padding is applied. With no padding the dot product between kernel and frame-segment cannot start at (0,0).

Accelerator Parameter Training

- The Training is performed using Axiado's ML Library with standard floating point values.
- The training is performed using a labeled training set and is run on a host machine.
- The result are a set of weights and biases $\in \Re$ (floating values, including negative numbers).
- The result of the training can be tested using the labeled test-dataset.
- For inference on the Accelerator the weights and biases need to be scaled (normalized) to the 7+1bit integer range (including negative numbers).
- The scaling and the processing of the scaled parameters is shown on the next two slides on an example with 2 Conv2D layers.

Accelerator Parameter Processing

1. Scale the weights and biases to 8bit integer values (Note: this leads to negligible loss of precision):

$$\mathrm{k}'_1 = int(\alpha \cdot k_1), \qquad b'_1 = int(\alpha \cdot b_1) \qquad f' = f \;\rightarrow The\ input\ frame\ is\ 8 - bit$$

$$\mathrm{k}'_2 = int(\alpha \cdot k_2), \qquad b'_2 = int(\alpha \cdot \beta \cdot b_2) \qquad \alpha, \beta = ScalingFactors$$

2. Run Inference using scaled parameters:

$$z'_1[m,n] \quad = \quad \sigma\left[\sum_r \sum_c k'_1[\mathrm{r}, c] \cdot f'\left[m + \mathrm{r} - int\left(\frac{R}{2}\right), n + \mathrm{c} - int\left(\frac{C}{2}\right)\right] + b'_1[m,n]\right]$$

$$z'_1[m,n] \quad = \quad \sigma\left[\sum_r \sum_c \alpha \cdot k_1[\mathrm{r}, c] \cdot f\left[m + \mathrm{r} - int\left(\frac{R}{2}\right), n + \mathrm{c} - int\left(\frac{C}{2}\right)\right] + \alpha \cdot b_1[m,n]\right]$$

$$z'_1[m,n] \quad = \quad \sigma(\alpha) \cdot z_1[m,n]$$

$$z'_1[m,n] = \beta \cdot \mathrm{z}_1[\mathrm{m},\mathrm{n}] \qquad \beta = \text{const.}$$

Note:

- *int* symbolizes conversion of the calculation in braces from float to integer.
- Assuming for the hidden layers ReLu activation functions (σ).

Accelerator Parameter Processing – cont’

3. Insert the result of the first layer into the second layer:

$$z'_2[p,q] = \sigma\left[\sum_r \sum_c k'_2[\mathrm{r},c] \cdot z'_2\left[p+\mathrm{r}-\boldsymbol{int}\left(\frac{R}{2}\right), q+\mathrm{c}-\boldsymbol{int}\left(\frac{C}{2}\right)\right] + b'_2[p,q]\right]$$

$$z'_2[p,q] = \sigma\left\{\sum_i \sum_l \alpha \cdot k_2[\mathrm{r},c] \cdot \beta \cdot z_1\left[p+\mathrm{r}-\boldsymbol{int}\left(\frac{R}{2}\right), q+\mathrm{c}-\boldsymbol{int}\left(\frac{C}{2}\right)\right] + \alpha \cdot \beta \cdot b_2[p,q]\right\}$$

$$z'_2[p,q] = \sigma(\alpha \cdot \beta) \cdot \sigma\left\{\sum_i \sum_l k_2\,[\mathrm{r},c] \cdot z_1\left[p+\mathrm{r}-\boldsymbol{int}\left(\frac{R}{2}\right), q+\mathrm{c}-\boldsymbol{int}\left(\frac{C}{2}\right)\right] + b_2[p,q]\right\}$$

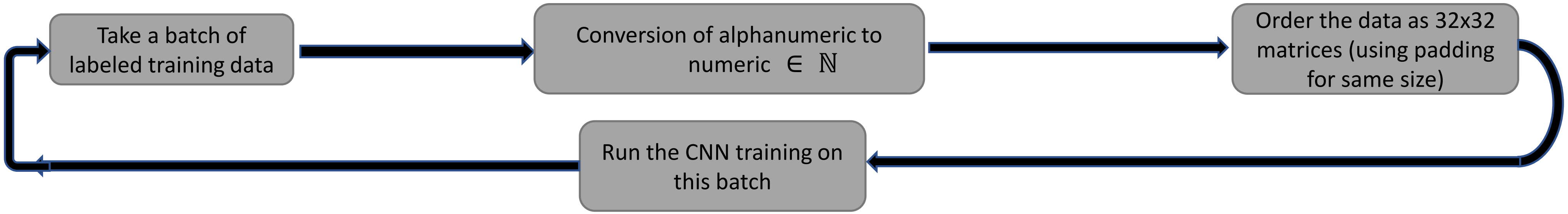
$$z'_2[p,q] = \sigma(\alpha \cdot \beta) \cdot z_2[p,q] \Rightarrow \boldsymbol{\mathit{for\ a\ 2{-}layer\ CNN\ the\ result\ needs\ to\ be\ multiplied}}\ \mathit{with}\ \frac{\mathbf{1}}{(\boldsymbol{\sigma}(\boldsymbol{\alpha} \cdot \boldsymbol{\beta}))}$$

Note:
- *int* symbolizes conversion of the calculation in braces from float to integer.
- Assuming for the hidden layers ReLu activation functions (σ).

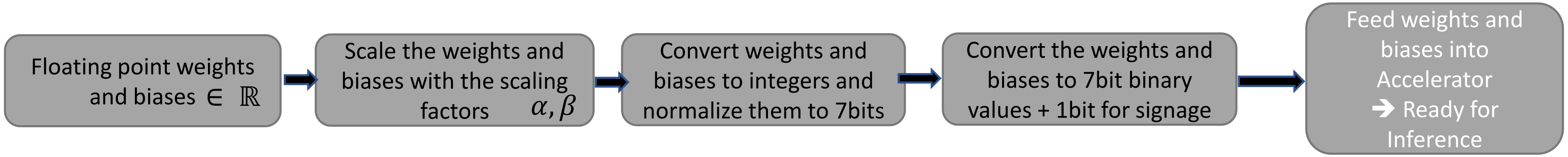
4. This processing can be continued with more Conv2D layers.

Model **Training** on Arbitrary Alphanumeric Data – Performed on Host Computer

1. Train the model Parameter (weights and biases):

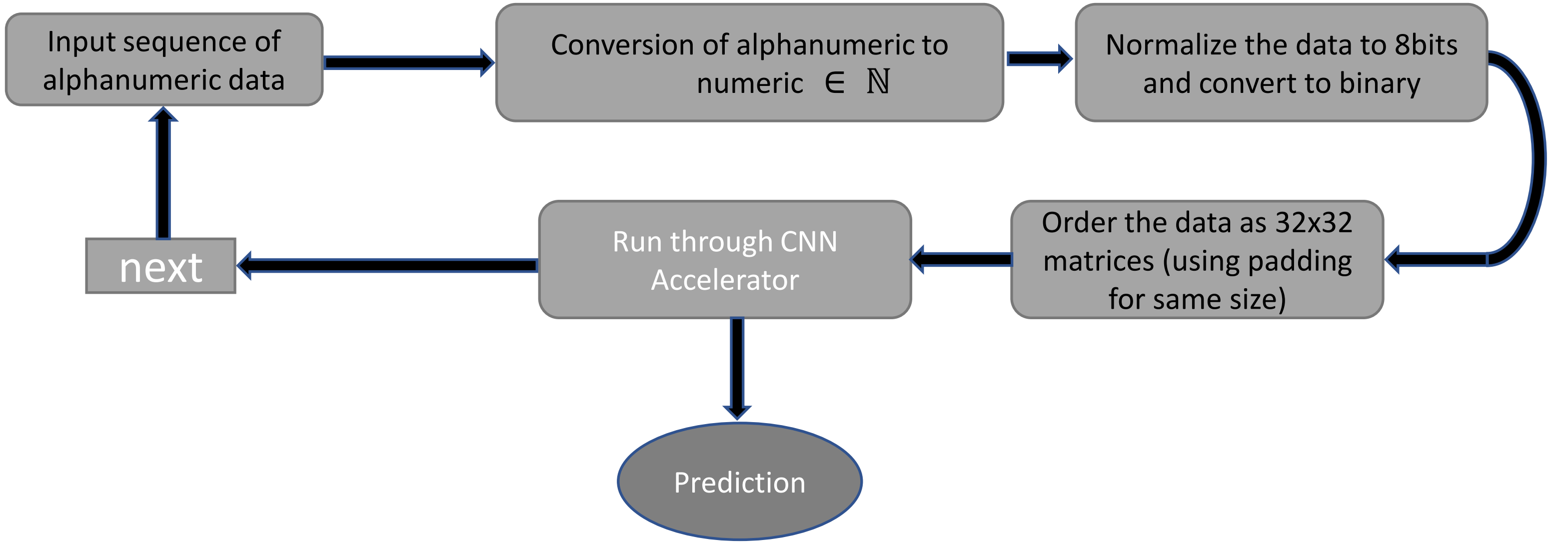


2. Prepare the trained parameters for the CNN Accelerator:

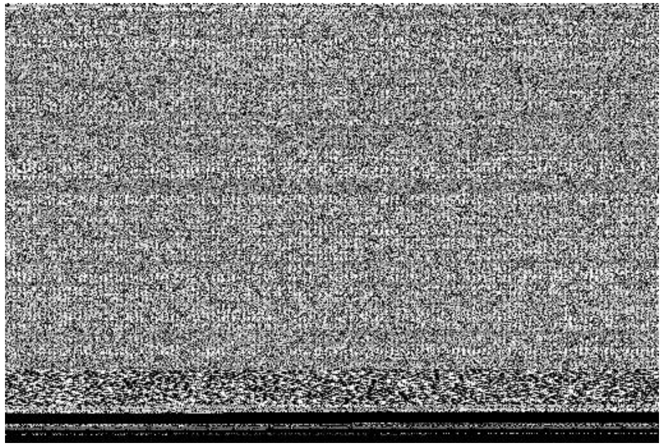
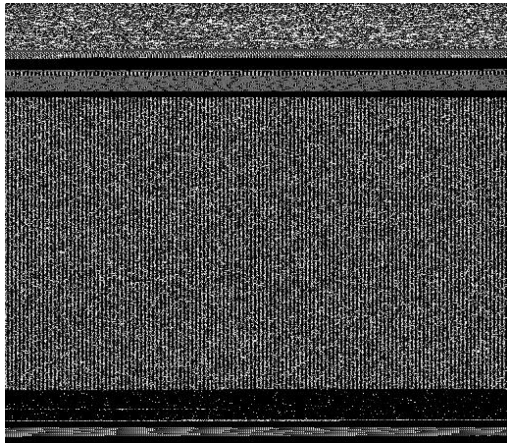
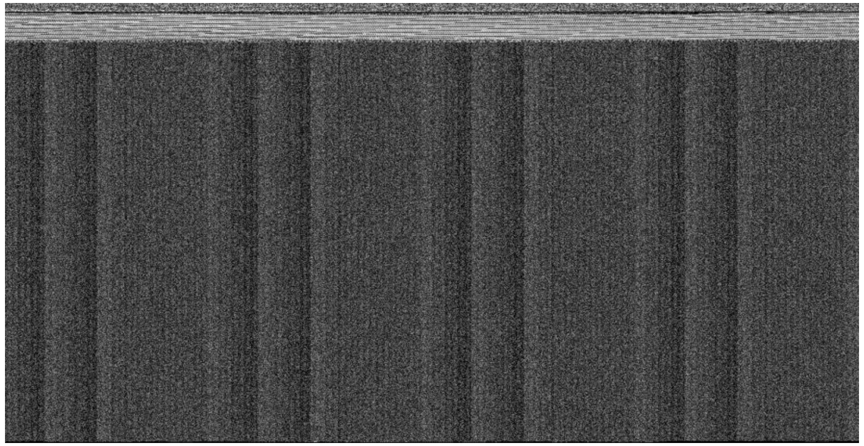
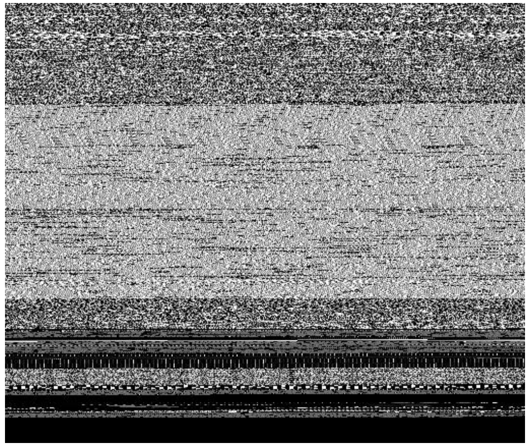


Inference Procedure with the CNN Accelerator

- After the training the resulting weights and biases are scaled, normalized to 7+1bits and converted to binaries.
- The binary weights and biases are loaded on the CNN Accelerator, which is then ready for inferencing.
- The block diagram on the right shows the procedure required for inferencing using alphanumeric input data.



Application of CNN’s on Malware IP Packets

	Example 1	Example 2
Gatak		
Lollipop		

Summary

- These slides show the planned 4 x Conv2D architecture of the Axiado CNN Accelerator.
- The Axiado CNN accelerator requires preprocessing of trained weights and biases, which includes a scaling to 7+1bit integer values with signage.
  - The calculation for this scaling is shown on a smaller 2 x Conv2D architecture.
- The custom calculations for the Accelerator are accommodated by using the Axiado ML Library, which was developed in-house and allows the custom manipulation of model layers and model parameters.
- Results of the Axiado CNN Accelerator and of the Axiado ML Library are cross-checked using TensorFlow 2 API.