CNN Accelerator Supporting Standard and Depthwise Convolution with ReLU and Max-Pooling

**1.Introduction:**

The proposed CNN Accelerator efficiently performs 3D convolution operations, supporting both standard convolution and depthwise convolution across multiple channels. It integrates pipelined MAC operations, ReLU activation, and max-pooling with on-chip BRAM-based memory management, enabling scalable processing of high-dimensional image data.

**2. Key capabilities & design highlights:**

* **Parameterized Design:** Configurable image size (n×n), kernel size (k, default 3), input channels (ch), filters (fi), memory address width (MA, default 18).
* **32-bit Signed Datapath:** All pixel, weight, and accumulation values are 32-bit signed.
* **Memory Interface:** Dual-port BRAM (bram\_dual\_port1) for image/kernel reads and pooled output writes.
* **Dedicated 3×3 MAC Core:** mac9 computes patch-level multiply–accumulate with preloaded kernel weights.
* **Index Generation & Line Buffering:**
  + shift\_index\_generetor creates 3×3 window indices.
  + Line buffer stores rows for sliding windows to minimize external memory traffic.
* **Nonlinear + Pooling:** ReLU clamp → 2×2 Max Pool (max\_pool4) with address/index generators.
* **Multi-Channel/Filter Flow:** FSM iterates over channels/filters, combines results, and sets completion flags.
* **Synthesis Pragmas:** (\* ram\_style="block", keep\_hierarchy="yes" \*) aid synthesis and optimization.

**3.Control Flow (FSM Overview)**

* **s0 – Kernel Load:** Stream kernel to karnel\_matrix.
* **s1 – Convolution:** Load image pixels → line buffer → MAC9 → write conv outputs.
* **s2 – Channel Update:** Adjust addresses for next channel/filter.
* **s3 – BRAM Align.**
* **s4 – ReLU + Pool:** Apply ReLU, 2×2 max-pool, store outputs.
* **s5/s6 – Filter Iteration:** Repeat for all filters.
* **s7 – Done/Idle.**

**4.Dataflow & Memory Mapping**

* **Latency Compensation:** FSM accounts for MAC pipeline (~4 cycles).
* **Dual-port BRAM:** Port A for reads, Port B for pooled writes.
* **Streaming Compute:** Single mac9 reused for all patches (area-efficient).
* **Memory Layout:**
  + Image/kernel stored as contiguous words.
  + Kernel: k×k 32-bit values.
  + A group of colorful cubes

    AI-generated content may be incorrect.Pooled results start at maxpool\_store\_addre

**For better understanding purpose image is taken from online so.**

**CNN Accelerator Pipeline Flow Chart**A diagram of a computer

AI-generated content may be incorrect.

**Hardware Utilization of 256×256 Matrix**

A screenshot of a computer

AI-generated content may be incorrect.

**Demonstration of Convolution Operation on a 256×256×3 Image Matrix with a 3×3×3 Kernel**

A blue circuit board with a white background

AI-generated content may be incorrect.

The original image has been separated into three individual channels—Channel 1 (R), Channel 2 (G), and Channel 3 (B)—each with a resolution of 256 × 256 pixels.

import cv2

import numpy as np

# Input and output paths

img\_path = r"D:\python file\standard\_convolution\amp.webp"

out\_file = r"D:\python file\standard\_convolution\pixels\_channelwise.txt"

# Step 1: Read the image

img = cv2.imread(img\_path)

# Step 2: Resize to 256x256

img\_resized = cv2.resize(img, (256, 256))

# Step 3: Split into channels (OpenCV loads as BGR)

B, G, R = cv2.split(img\_resized)

# Step 4: Write pixel values to file (channel-wise)

with open(out\_file, "w") as f:

    # Write B channel

    for row in B:

        for val in row:

            f.write(f"{val}\n")

    # Write G channel

    for row in G:

        for val in row:

            f.write(f"{val}\n")

    # Write R channel

    for row in R:

        for val in row:

            f.write(f"{val}\n")

print("Pixel file generated successfully at:", out\_file)

by using this code I have generated pixels\_channelwise.txt this file .then convert into this hex file now this hex is uploaded to my CNN Accleretor(here depthwise and standard convolution possible\_multichannel \_multifilter possible).After simmuletion it generates this file .Again this file is uploaded into this python code.



import numpy as np

import cv2

# Input pixel file (single channel convolution output)

in\_file = r"D:\memorey test file for cnn\_accleretor\bram\_dump6.txt"

# Image parameters

height = 254

width = 256

num\_pixels = height \* width  # 65024 pixels expected

# Step 1: Read pixel values

with open(in\_file, "r") as f:

    lines = f.readlines()

# Convert to integers

lines = [int(x.strip()) for x in lines]

# Step 2: Sanity check

if len(lines) != num\_pixels:

    raise ValueError(f"Pixel file has {len(lines)} values, but expected {num\_pixels} ({height}x{width})")

# Step 3: Reshape into 2D matrix

img = np.array(lines, dtype=np.float32).reshape((height, width))

# Step 4: Normalize to 0–255 (for saving as image)

img\_norm = cv2.normalize(img, None, 0, 255, cv2.NORM\_MINMAX)

img\_uint8 = img\_norm.astype(np.uint8)

# Step 5: Save as PNG image

out\_file = r"D:\python file\standard\_convolution\multicha\_convolution\_output1.png"

cv2.imwrite(out\_file, img\_uint8)

print(f"Image generated successfully at: {out\_file}")

A close-up of a circuit board

AI-generated content may be incorrect.A close-up of a circuit board

AI-generated content may be incorrect.

This is expected Getting from CNN RLT

# Step 5: Define 3x3x3 kernel (example)

kernel = np.array([

    [[0, -1, 0],

     [-1, 5, -1],

     [0, -1, 0]],   # for Red

    [[0, -1, 0],

     [-1, 5, -1],

     [0, -1, 0]],   # for Green

    [[0, -1, 0],

     [-1, 5, -1],

     [0, -1, 0]]    # for Blue

],

This 3\*3\*3 kernel I had used to my CNN\_Accleretor.

**Required Clock Cycles for 256×256×3 Convolution with 3×3×3 Kernel :**

A screenshot of a computer

AI-generated content may be incorrect.

According to my design when system reach to in state 4 .Standard convolution operation is just completed here we have performed 256\*256\*3 with karnel 3\*3\*3 no. of filter=1.

Required Clk=1966885/10 =1,96,688

According to my calculation=256\*256\*3=1,96,608 which is relatively quiet fast.

<https://ieeexplore.ieee.org/document/10420491#:~:text=Optimizing%20CNN%20Computation%20Using%20RISC%2DV%20Custom%20Instruction%20Sets%20for%20Edge%20Platforms>

According to this IEEE paper

 Achieves **7–8× higher performance** compared to baseline.

 Provides **greater hardware efficiency**.

 Optimized **datapath design** for multi-channel/multi-filter operations.

 Efficient **memory access and reuse** reduce overhead.