

دانشگاه صنعتی امیرکبیر  
(پلی تکنیک تهران)  
دانشکده مهندسی کامپیوتر

گزارش پروژه نهایی درس طراحی سیستم‌های قابل بازپیکربندی

## طراحی و شبیه‌سازی شبکه عصبی CNN با هدف تشخیص ارقام دست‌نویس به وسیله HLS

نگارش

رضا آدینه پور

استاد درس

جناب آقای دکتر صاحب‌الزمانی

بهمن ۱۴۰۳



## سپاس

از استاد گرانقدر خود، جناب آقای دکتر صاحب الزمانی، به خاطر ارائه‌های بی‌نظیرشان در طول ترم خالصانه تشکر و قدردانی می‌نمایم. همچنین از جناب آقای دکتر ملکوتی، تدریس‌یار محترم درس نیز به دلیل راهنمایی‌های بی‌نظیر و حمایت‌های بی‌دریغ ایشان در طول این پروژه، صمیمانه تشکر می‌نمایم. بازخوردها و کمک‌های سازنده ایشان نقش بسزایی در شکل‌گیری این پروژه داشته است.

## چکیده

شبکه‌های عصبی پیچشی یکی از پرکاربردترین مدل‌ها در حوزه یادگیری عمیق هستند که در بسیاری از کاربردها مانند شناسایی تصاویر و پردازش داده‌های بصری مورد استفاده قرار می‌گیرند. با توجه به نیاز روزافزون به پردازش سریع و بهینه، استفاده از سخت‌افزارهایی مانند FPGA به دلیل قابلیت پردازش موازی و توان مصرفی پایین، گزینه‌ای ایده‌آل برای پیاده‌سازی این شبکه‌ها محسوب می‌شود.

در این پروژه، هدف پیاده‌سازی یک شبکه عصبی پیچشی برای شناسایی ارقام دست‌نویس بر روی FPGA با استفاده از روش سنتز سطح بالا است. فرآیند پیاده‌سازی شامل دو فاز اصلی بود: در فاز نرم‌افزاری، شبکه مورد نظر آموزش داده شد و وزن‌های آن ذخیره گردید. سپس در فاز سخت‌افزاری، وزن‌های ذخیره‌شده به FPGA منتقل شده و داده‌های ورودی به شبکه ارسال شدند. نتایج خروجی به منظور ارزیابی عملکرد و صحت شناسایی پردازش شدند. این پیاده‌سازی ترکیبی از کارایی بالا و انعطاف‌پذیری FPGA را با قدرت یادگیری عمیق ادغام کرده و امکان بهره‌وری بیشتر در کاربردهای عملی را فراهم می‌کند.

کلیدواژه‌ها: شبکه‌های عصبی، یادگیری عمیق، شبکه عصبی پیچشی، FPGA

# فهرست مطالب

۱	مقدمه	۱
۱-۱	تعریف مسئله	۱
۲-۱	اهمیت پژوهش	۲
۳-۱	اهداف پژوهش	۳
۴-۱	ساختار پژوهش	۳
۲	مفاهیم اولیه	۴
۱-۲	شبکه عصبی CNN	۴
۲-۲	اجزای اصلی شبکه CNN	۵
۱-۲-۲	لایه کانولوشن	۵
۲-۲-۲	لایه فعال سازی	۵
۳-۲-۲	لایه تجمیع	۵
۴-۲-۲	لایه تمام متصل	۶
۵-۲-۲	لایه خروجی	۶
۳-۲	نحوه عملکرد شبکه CNN	۷
۴-۲	ساختار FPGA	۷
۵-۲	اجزای مهم FPGA	۸
۱-۵-۲	بلوک های منطقی قابل پیکربندی (CLB)	۸

۸	۲-۵-۲ منابع اتصالات (Routing Resources)
۸	۳-۵-۲ بلوک‌های ورودی/خروجی (I/O Blocks)
۹	۴-۵-۲ حافظه‌های داخلی
۹	۵-۵-۲ واحدهای DSP
۹	۶-۲ قابلیت بازپیکربندی FPGA
۹	۷-۲ ابزار HLS
۱۰	۱-۷-۲ مزایای استفاده از Xilinx Vitis HLS
۱۱	۳ طراحی و شبیه‌سازی
۱۱	۱-۳ فاز نرم‌افزاری
۱۱	۱-۱-۳ انتخاب معماری
۱۳	۲-۱-۳ آموزش شبکه
۱۵	۳-۱-۳ ذخیره پارامترها
۱۶	۲-۳ فاز سخت‌افزاری
۱۶	۱-۲-۳ Fix-Point کردن داده‌ها
۱۷	۲-۲-۳ لایه کانولوشن
۱۸	۳-۲-۳ Fully Connected لایه
۱۹	۴-۲-۳ Flatten لایه
۱۹	۵-۲-۳ ماژول اصلی (Top Module)
۲۰	۶-۲-۳ Data Flow ماژول
۲۱	۳-۳ سنتز مدل
۲۲	۴-۳ تست مدل
۲۶	۴ نتایج
۲۶	۱-۴ بدون بهینه‌سازی

۲-۴ نیمه بهینه ..... ۲۷

۳-۴ بهینه سازی کامل ..... ۲۷

۵ نتیجه گیری و جمع بندی ..... ۲۹

مراجع ..... ۳۱

## فهرست جداول

۱۲	..... معماری‌های بررسی شده	۱-۳
۱۲	..... اطلاعات مربوط به معماری‌های مختلف	۲-۳
۲۸	..... اطلاعات مربوط به معماری‌های مختلف	۱-۴



## فهرست تصاویر

۱-۱	مسئله طبقه‌بندی [۱]	۱
۲-۱	مسئله طبقه‌بندی [۲]	۲
۱-۲	ساختار یک شبکه CNN [۳]	۴
۲-۲	عملیات کانولوشن [۴]	۵
۳-۲	تابع فعال‌ساز ReLU [۵]	۶
۴-۲	لایه Max_pooling [۶]	۶
۵-۲	لایه تمام‌متصل [۷]	۷
۶-۲	CLB ها در FPGA	۸
۱-۳	معماری شبکه	۱۳
۲-۳	ساختار شبکه تعریف شده	۱۴
۳-۳	نمودارهای Loss و Accuracy	۱۵
۴-۳	منابع مصرفی پس‌از سنتز (الف)	۲۲
۵-۳	منابع مصرفی پس‌از سنتز (ب)	۲۲
۶-۳	منابع مصرفی پس‌از سنتز (ج)	۲۳
۷-۳	Accuracy پیش‌بینی به‌ازای ۱۰۰ تصویر	۲۳
۸-۳	Accuracy پیش‌بینی به‌ازای ۵۰۰ تصویر	۲۳
۹-۳	تشخیص‌های اشتباه شبکه	۲۴

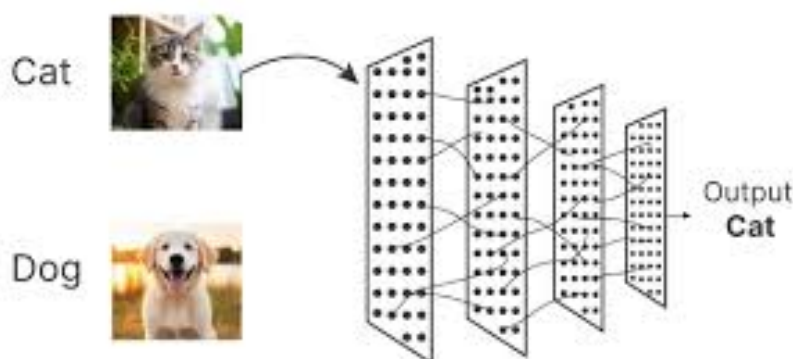
۲۶	.....	۱-۴ خروجی سنتز بهینه نشده
۲۷	.....	۲-۴ خروجی سنتز نیمه بهینه
۲۸	.....	۳-۴ خروجی سنتز بهینه کامل

# فصل ۱

## مقدمه

### ۱-۱ تعریف مسئله

طبقه‌بندی<sup>۱</sup> یکی از مسائل اصلی در حوزه یادگیری ماشین<sup>۲</sup> است که هدف آن تخصیص ورودی‌ها به یکی از دسته‌های از پیش تعریف شده می‌باشد. شبکه‌های عصبی پیچشی<sup>۳</sup> (CNN) به دلیل توانایی بالای خود در استخراج ویژگی‌های سلسله‌مراتبی از داده‌های خام، در بسیاری از مسائل طبقه‌بندی، از جمله شناسایی تصاویر عملکرد بسیار خوبی داشته‌اند. مسئله طبقه‌بندی ارقام دست‌نویس به عنوان یک مسئله مرجع، نقش مهمی در نشان دادن توانایی شبکه‌های عصبی در پردازش داده‌های بصری دارد و به طور گسترده برای ارزیابی روش‌ها و مدل‌های مختلف استفاده می‌شود.



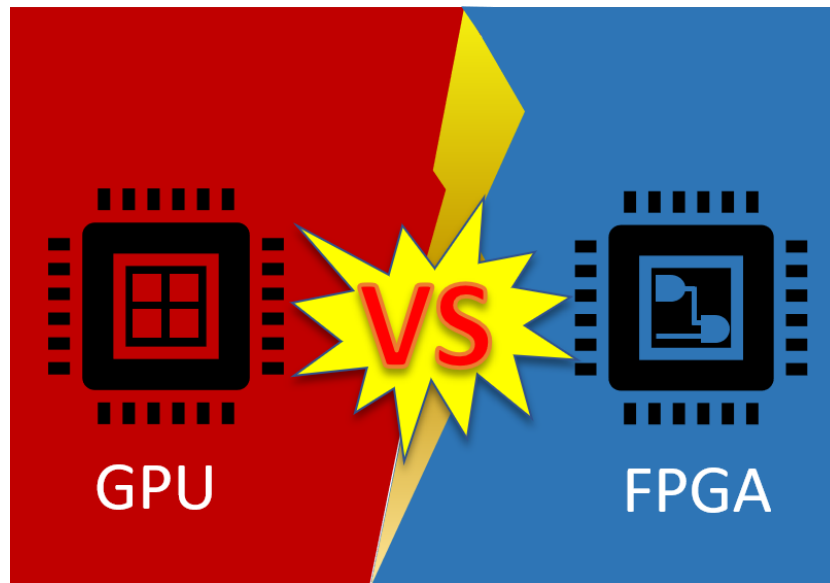
شکل ۱-۱: مسئله طبقه‌بندی [۱]

<sup>1</sup>Classification

<sup>2</sup>Machine Learning

<sup>3</sup>Convolutional Neural Network

با این حال، اجرای مدل‌های CNN در کاربردهای عملی چالش‌هایی مانند پیچیدگی محاسباتی بالا و نیاز به منابع سخت‌افزاری کارآمد را به همراه دارد. در حالی که GPUها به دلیل توان عملیاتی بالا گزینه‌ای مناسب برای آموزش و استنتاج<sup>۴</sup> مدل‌ها هستند، مصرف انرژی بالا و محدودیت‌های آن‌ها در کاربردهای نهفته<sup>۵</sup> و محیط‌هایی با منابع محدود، آن‌ها را برای برخی کاربردها نامناسب می‌سازد. در مقابل، FPGAها با قابلیت پردازش موازی، مصرف انرژی کمتر و قابلیت بازپیکربندی<sup>۶</sup>، گزینه‌ای ایده‌آل برای پیاده‌سازی مدل‌های CNN در کاربردهایی هستند که نیاز به پردازش بی‌درنگ<sup>۷</sup> و بهره‌وری بالا<sup>۸</sup> دارند.



شکل ۱-۲: مسئله طبقه‌بندی [۲]

## ۲-۱ اهمیت پژوهش

پیاده‌سازی شبکه‌های عصبی پیچشی بر روی FPGA نه تنها به دلیل چالش‌های فنی موجود در ترکیب یادگیری عمیق با سخت‌افزارهای نهفته اهمیت دارد، بلکه از جنبه‌های کاربردی نیز تأثیر بسزایی دارد. این پروژه امکان استفاده از مدل‌های یادگیری عمیق در محیط‌هایی با منابع محدود و نیاز به مصرف انرژی کم، مانند دستگاه‌های IoT، سیستم‌های صنعتی بی‌درنگ و تجهیزات پزشکی قابل حمل<sup>۹</sup> را فراهم می‌کند. علاوه بر این، FPGAها به دلیل انعطاف‌پذیری در طراحی و تطبیق‌پذیری با کاربردهای متنوع، می‌توانند بستری مناسب برای توسعه سامانه‌های هوشمند با کارایی بالا باشند. نتایج این پژوهش می‌تواند راهگشای کوچکی برای پژوهشگران و

<sup>4</sup>Inference

<sup>5</sup>Embedded

<sup>6</sup>Reconfigurability

<sup>7</sup>Real-Time

<sup>8</sup>High Performance

<sup>9</sup>Portable

مهندسان در کاهش هزینه‌های طراحی، بهبود سرعت پردازش و افزایش بهره‌وری سیستم‌های مبتنی بر یادگیری عمیق باشد.

## ۳-۱ اهداف پژوهش

هدف اصلی این پژوهش، شتاب‌دهی سخت‌افزاری فاز استنتاج<sup>۱۰</sup> شبکه‌های عصبی پیچشی با بهینه‌سازی مصرف توان و انرژی است. با توجه به نیاز روزافزون به پردازش سریع و کارآمد داده‌ها در کاربردهای بی‌درنگ و نهفته، استفاده از FPGA به عنوان بستری مناسب برای تحقق این هدف در اولویت قرار گرفته است. این پروژه به دنبال دستیابی به معماری سخت‌افزاری است که علاوه بر ارائه سرعت بالا در پردازش، مصرف انرژی را به حداقل برساند و قابلیت پیاده‌سازی در محیط‌های محدود به منابع مانند سیستم‌های IoT، دستگاه‌های قابل حمل و کاربردهای صنعتی را فراهم کند.

## ۴-۱ ساختار پژوهش

این پژوهش در ۴ فصل انجام شده است. در فصل ۱ به مقدمه و اهمیت موضوع پژوهش پرداخته شده است. در فصل ۲ به مفاهیم اولیه و پیش‌نیازها پرداخته شده است. در ادامه در فصل ۳ پژوهش به بررسی معماری‌های مختلف و طراحی بهترین معماری با هدف کمینه کردن تاخیر و منابع مصرفی و در نهایت شبیه‌سازی و تست معماری پرداخته شده است.

---

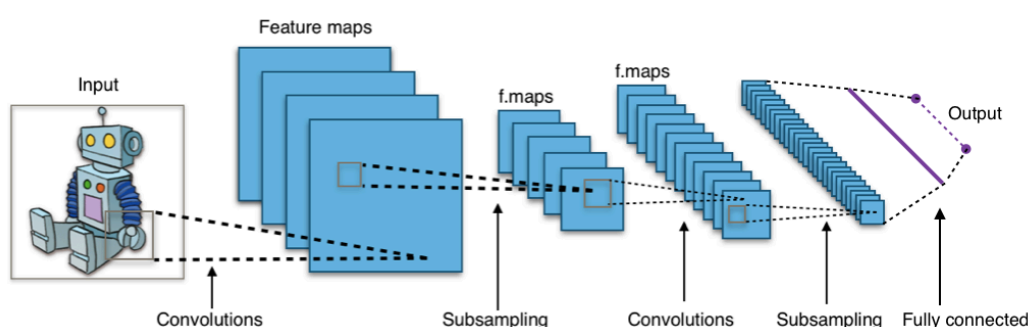
<sup>10</sup>Inference

## فصل ۲

# مفاهیم اولیه

### ۱-۲ شبکه عصبی CNN

شبکه‌های عصبی پیچشی (Convolutional Neural Networks یا CNN) نوعی از شبکه‌های عصبی مصنوعی هستند که به طور خاص برای پردازش داده‌های با ساختار شبکه‌ای مانند تصاویر طراحی شده‌اند. این شبکه‌ها به دلیل توانایی بالای خود در شناسایی الگوها و ویژگی‌ها، به طور گسترده در مسائلی مانند طبقه‌بندی تصاویر<sup>۱</sup> و تشخیص اشیاء<sup>۲</sup> استفاده می‌شوند. CNN‌ها از معماری سلسله‌مراتبی<sup>۳</sup> برای استخراج ویژگی‌ها از داده‌های ورودی استفاده می‌کنند و قادرند ویژگی‌های سطح پایین (مانند لبه‌ها) تا ویژگی‌های سطح بالا (مانند اشکال پیچیده) را به طور خودکار شناسایی کنند.



شکل ۱-۲: ساختار یک شبکه CNN [۳]

<sup>1</sup>Image Classification

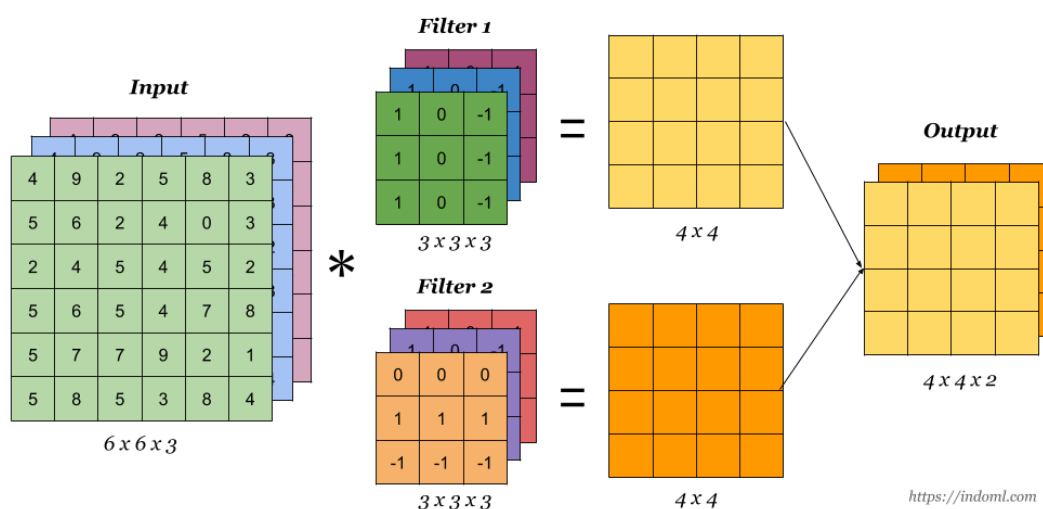
<sup>2</sup>Object Detection

<sup>3</sup>Hierarchical

## ۲-۲ اجزای اصلی شبکه CNN

### ۱-۲-۲ لایه کانولوشن

این لایه وظیفه استخراج ویژگی‌ها از داده‌های ورودی را بر عهده دارد. در این لایه، یک یا چند فیلتر<sup>۴</sup> کوچک بر روی داده‌های ورودی حرکت کرده و عملیات کانولوشن را انجام می‌دهند. نتیجه این عملیات ویژگی‌های مکانی<sup>۵</sup> است که اطلاعات مهم را حفظ کرده و داده‌های غیرضروری را حذف می‌کند.



شکل ۲-۲: عملیات کانولوشن [۴]

### ۲-۲-۲ لایه فعال‌سازی

پس از هر لایه کانولوشن، از یک تابع فعال‌ساز<sup>۶</sup> غیرخطی (مانند ReLU)<sup>۷</sup> استفاده می‌شود. این تابع باعث می‌شود مدل بتواند روابط پیچیده و غیرخطی را یاد بگیرد. تابع ReLU معمولاً بیشترین استفاده را دارد و با نگه‌داشتن مقادیر مثبت و صفر کردن مقادیر منفی، سرعت و کارایی مدل را افزایش می‌دهد.

### ۳-۲-۲ لایه جمع

لایه جمع<sup>۸</sup> وظیفه کاهش ابعاد داده‌ها را دارد تا تعداد پارامترها و پیچیدگی محاسباتی کاهش یابد. معمولاً از روش Max Pooling استفاده می‌شود، که در آن بزرگ‌ترین مقدار در هر ناحیه انتخاب می‌شود. این فرآیند

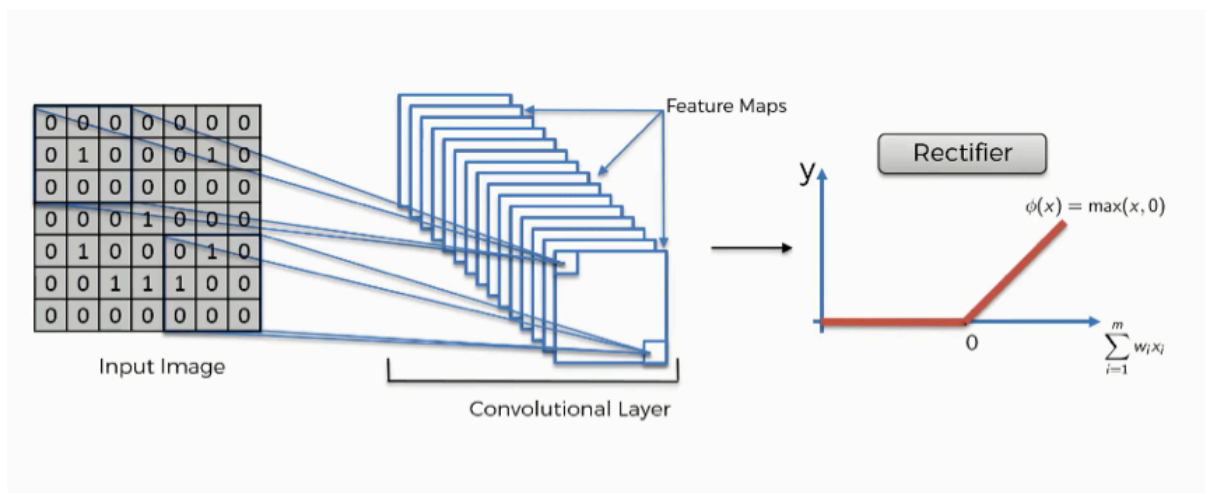
<sup>۴</sup>Kernel

<sup>۵</sup>Spatial Features

<sup>۶</sup>Activation Function

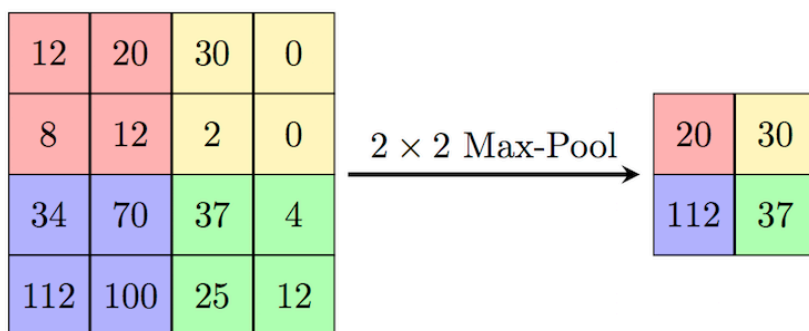
<sup>۷</sup>Rectified Linear Unit

<sup>۸</sup>Pooling



شکل ۲-۳: تابع فعال‌ساز ReLU [۵]

باعث افزایش مقاومت مدل در برابر تغییرات جزئی در داده‌های ورودی (مانند انتقال یا چرخش) می‌شود.



شکل ۲-۴: لایه Max\_pooling [۶]

۲-۲-۴ لایه تمام‌متصل

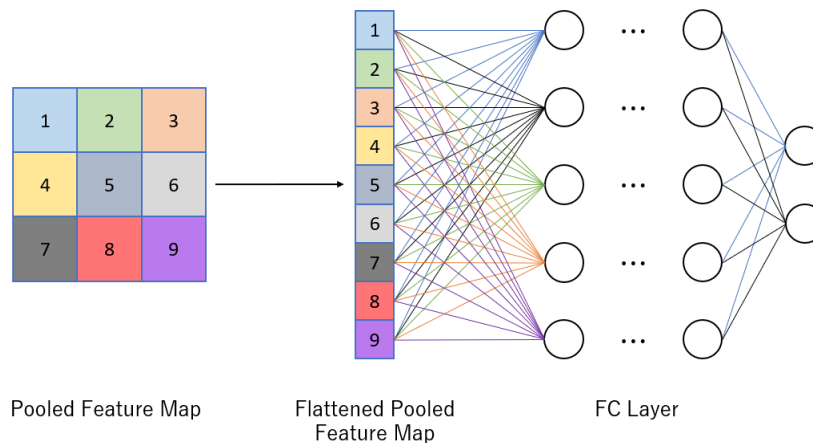
لایه تمام‌متصل<sup>۹</sup>، ویژگی‌های استخراج‌شده توسط لایه‌های قبلی را به صورت یک بردار مسطح درمی‌آیند و به نرون‌های خروجی متصل می‌شوند. این لایه وظیفه تصمیم‌گیری نهایی (مانند طبقه‌بندی) را بر عهده دارد.

۲-۲-۵ لایه خروجی

این لایه از یک تابع فعال‌سازی مانند Sigmoid یا Softmax برای تولید خروجی استفاده می‌کند. خروجی این لایه معمولاً احتمال تعلق ورودی به هر کلاس در مسئله طبقه‌بندی است.

<sup>۹</sup>Fully Connected





شکل ۲-۵: لایه تمام متصل [۷]

## ۳-۲ نحوه عملکرد شبکه CNN

CNN ها با دریافت داده‌های ورودی (مانند تصاویر)، آن‌ها را از طریق لایه‌های مختلف عبور داده و ویژگی‌های مهم را مرحله به مرحله استخراج می‌کنند. در هر مرحله، ویژگی‌ها پیچیده‌تر و خاص‌تر می‌شوند. لایه‌های کانولوشنی ویژگی‌ها را استخراج می‌کنند، لایه‌های تجمیع ابعاد داده‌ها را کاهش می‌دهند و در نهایت، لایه‌های تمام متصل و خروجی تصمیم‌گیری نهایی را انجام می‌دهند. این معماری به CNN ها اجازه می‌دهد تا در کاربردهایی مانند شناسایی چهره، تشخیص اشیاء و تحلیل تصاویر پزشکی عملکردی بسیار دقیق و مؤثر داشته باشند.

## ۴-۲ ساختار FPGA

FPGA<sup>۱۰</sup> یک تراشه سخت‌افزاری قابل برنامه‌ریزی است که به کاربران اجازه می‌دهد ساختار داخلی آن را پس از تولید تغییر دهند و برای کاربردهای خاص طراحی کنند. این قابلیت بازپیکربندی<sup>۱۱</sup> یکی از ویژگی‌های کلیدی FPGA است که امکان اجرای مجموعه‌ای از عملکردهای منطقی و موازی را با انعطاف‌پذیری بالا فراهم می‌کند. FPGA ها از ساختارهایی شامل بلوک‌های منطقی قابل پیکربندی (CLB)<sup>۱۲</sup>، منابع اتصالات قابل برنامه‌ریزی، و منابع ورودی/خروجی تشکیل شده‌اند که با یکدیگر کار می‌کنند تا مدارهای دلخواه را پیاده‌سازی کنند.

<sup>10</sup>Field-Programmable Gate Array

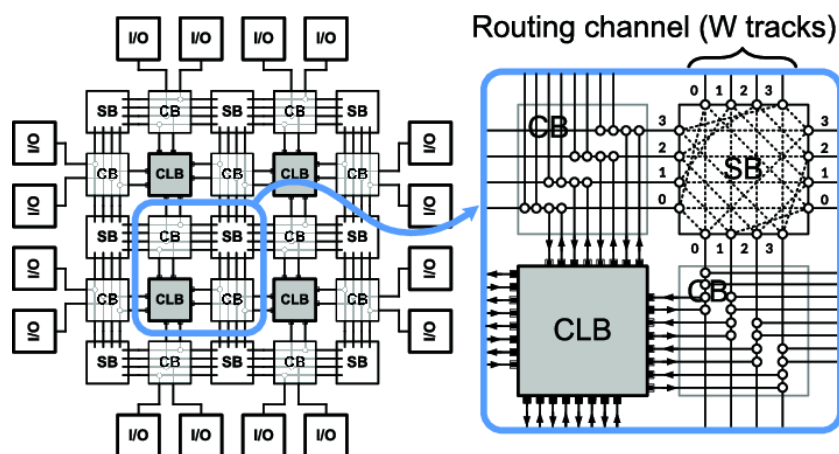
<sup>11</sup>Reconfigurable

<sup>12</sup>Combinational Logic Block

## ۵-۲ اجزای مهم FPGA

### ۱-۵-۲ بلوک‌های منطقی قابل پیکربندی (CLB)

این بلوک‌ها هسته اصلی FPGA هستند و از ترکیب LUT، فلیپ‌فلاپ‌ها، و عناصر منطقی تشکیل شده‌اند. LUT‌ها امکان پیاده‌سازی توابع منطقی را فراهم می‌کنند و فلیپ‌فلاپ‌ها برای ذخیره مقادیر استفاده می‌شوند. با استفاده از CLB‌ها، می‌توان انواع گیت‌های منطقی و توابع پیچیده‌تر را پیاده‌سازی کرد.



شکل ۲-۶: CLB ها در FPGA

### ۲-۵-۲ منابع اتصالات (Routing Resources)

FPGA دارای شبکه‌ای از مسیرهای قابل برنامه‌ریزی است که بلوک‌های منطقی را به یکدیگر و به پورت‌های ورودی/خروجی متصل می‌کند. این منابع شامل سوئیچ‌ها و ماتریس‌های اتصالات است که امکان تنظیم مسیرهای داده در FPGA را فراهم می‌کنند.

### ۳-۵-۲ بلوک‌های ورودی/خروجی (I/O Blocks)

این بلوک‌ها مسئول ارتباط FPGA با دنیای خارجی هستند و امکان تبادل داده با سایر دستگاه‌ها را فراهم می‌کنند. I/O‌ها برای پشتیبانی از پروتکل‌های مختلف ارتباطی قابل پیکربندی هستند.

## ۴-۵-۲ حافظه‌های داخلی

FPGAها شامل حافظه‌هایی مانند RAM بلوکی<sup>۱۳</sup> و حافظه‌های توزیع شده<sup>۱۴</sup> هستند که برای ذخیره داده‌ها و متغیرها در طول اجرای عملیات استفاده می‌شوند.

## ۵-۵-۲ واحدهای DSP

اکثر FPGAهای مدرن دارای واحدهای پردازش سیگنال دیجیتال<sup>۱۵</sup> هستند که برای عملیات ریاضی پیچیده مانند ضرب و جمع بهینه‌سازی شده‌اند. این واحدها نقش مهمی در کاربردهایی مانند پردازش سیگنال و یادگیری عمیق ایفا می‌کنند.

## ۶-۲ قابلیت بازپیکربندی FPGA

FPGAها به کاربران اجازه می‌دهند مدار داخلی خود را با استفاده از ابزارهای سنتز سخت‌افزاری مانند Verilog، VHDL، یا HLS تغییر دهند. این قابلیت به معنای انعطاف‌پذیری بالا برای تغییر یا بهبود طراحی است، حتی پس از ساخت سخت‌افزار. علاوه بر این، برخی از FPGAها از بازپیکربندی پویا<sup>۱۶</sup> پشتیبانی می‌کنند که امکان تغییر بخشی از طراحی را در زمان اجرا بدون اختلال در عملکرد سایر بخش‌ها فراهم می‌کند.

## ۷-۲ ابزار HLS

Xilinx Vitis HLS (High-Level Synthesis) یک پلتفرم توسعه نرم‌افزاری است که به مهندسان این امکان را می‌دهد تا کدهای نرم‌افزاری نوشته شده در زبان‌های سطح بالا مانند C، C++ یا OpenCL را به سخت‌افزار FPGA تبدیل کنند. هدف اصلی این ابزار تسهیل فرآیند طراحی سخت‌افزار است، به گونه‌ای که توسعه‌دهندگان نیازی به درک عمیق از جزئیات معماری FPGA نداشته باشند. با استفاده از Vitis HLS، طراحی‌های سخت‌افزاری به طور خودکار از کدهای سطح بالا استخراج شده و به طراحی‌های RTL<sup>۱۷</sup> که قابل سنتز در FPGA هستند، تبدیل می‌شوند.

---

<sup>13</sup>BRAM

<sup>14</sup>Distributed RAM

<sup>15</sup>DSP

<sup>16</sup>Dynamic Reconfiguration

<sup>17</sup>Register-Transfer Level

## ۱-۷-۲ مزایای استفاده از Xilinx Vitis HLS

- کاهش زمان توسعه استفاده از زبان‌های سطح بالا برای نوشتن کد، به طراحان این امکان را می‌دهد که زمان بیشتری برای الگوریتم‌ها و منطق طراحی صرف کنند و به جای پرداختن به جزئیات معماری FPGA، تمرکز بیشتری بر روی ویژگی‌های عملکردی داشته باشند.
- ارتقاء عملکرد ابزار Vitis HLS این امکان را فراهم می‌کند که طراحی‌های سخت‌افزاری بهینه‌سازی شوند، به‌ویژه در زمینه‌هایی مانند پردازش موازی، سرعت بالا و کارایی انرژی.
- سادگی پیاده‌سازی بدون نیاز به دانش تخصصی در زمینه زبان‌های سخت‌افزاری مانند VHDL یا Verilog، مهندسان می‌توانند به راحتی از C/C++ یا OpenCL برای ایجاد مدارهای پیچیده استفاده کنند.

## فصل ۳

# طراحی و شبیه‌سازی

هدف در این پروژه پیاده‌سازی یک شبکه CNN بر روی FPGA با استفاده از ابزار HLS با هدف تشخیص ارقام دست نویس است. بدین منظور پروژه را به دو فاز تقسیم می‌کنیم:

۱. فاز نرم‌افزاری

۲. فاز سخت‌افزاری

که در ادامه هر یک از بخش‌های پروژه را با جزئیات توضیح می‌دهیم.

## ۳-۱ فاز نرم‌افزاری

### ۳-۱-۱ انتخاب معماری

در ابتدا می‌بایست بهترین معماری را برای شبکه خود انتخاب کنیم. معیارهایی که در انتخاب معماری به آن توجه شده است به صورت زیر می‌باشد:

• دقت<sup>۱</sup> شبکه

• خطا<sup>۲</sup> شبکه

• حجم پارامترهای ذخیره شده

---

<sup>۱</sup>Accuracy

<sup>۲</sup>Loss

در صورت پروژه بیان شده است که دقت آموزش شبکه می‌بایست بالای ۹۰ درصد باشد. همچنین به دلیل آنکه قرار است پارامترهای آموزش دیده را در فاز دوم به HLS ببریم و در آنجا بارگزاری کنیم یکی از مواردی که بسیار برای ما اهمیت دارد حجم پارامترهای ذخیره شده در مدل است. بنابر این با توجه به این سه معیار مطرح شده مدل‌های مختلفی را تست کرده ایم که در جدول زیر آورده شده است.

جدول ۳-۱: معماری‌های بررسی شده

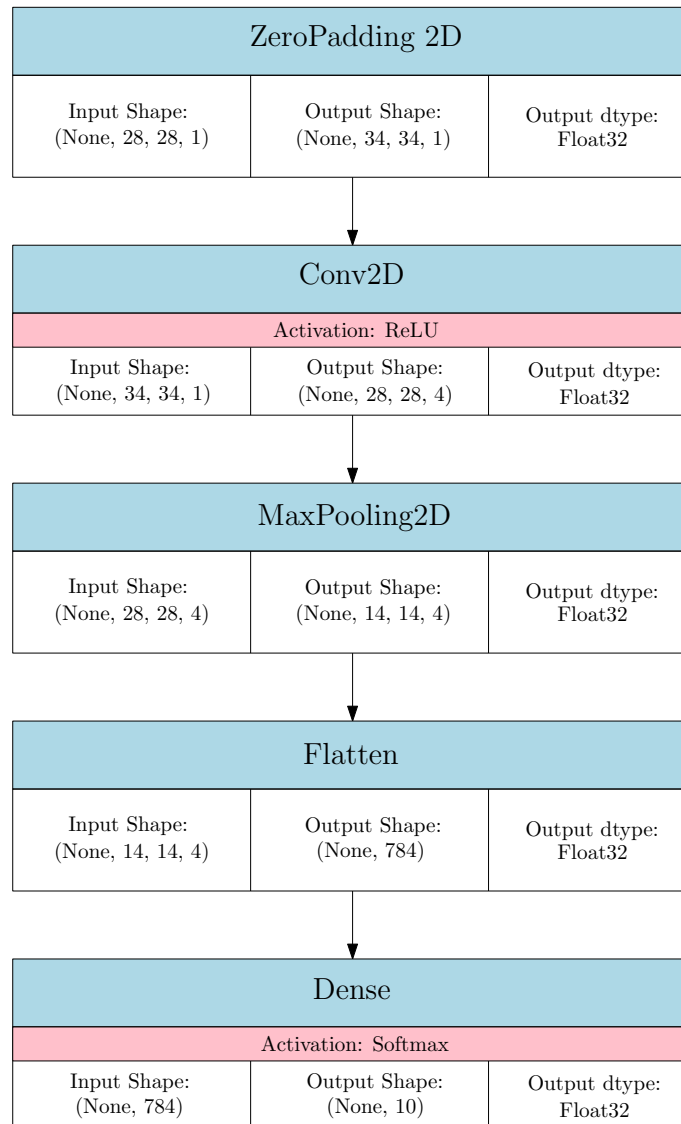
معماری	لایه Zero Padding	لایه Max Pooling	تعداد لایه‌های کانولوشن	تعداد فیلترها	اندازه کرنل	Stride	تابع فعال‌ساز	تعداد لایه‌های Fully Connected
معماری ۱	بله	خیر	۱	۸	۳x۳	۱	ReLU	۴ (۱۲۸, ۶۴, ۶۴, ۱۰)
معماری ۲	بله	خیر	۱	۸	۳x۳	۱	ReLU	۳ (۱۲۸, ۶۴, ۱۰)
معماری ۳	بله	خیر	۱	۸	۳x۳	۱	ReLU	۲ (۱۲۸, ۱۰)
معماری ۴	بله	خیر	۱	۸	۳x۳	۱	ReLU	۱ (۱۰)
معماری ۵	بله	بله	۱	۴	۷x۷	۱	ReLU	۱ (۱۰)

به‌ازای تمامی معماری‌های جدول ۳-۱ یک بار شبکه آموزش داده شده است و تعداد پارامترهای مدل و حجم آن‌ها به‌صورت زیر گزارش می‌شود:

جدول ۳-۲: اطلاعات مربوط به معماری‌های مختلف

معماری	تعداد کل پارامترها	حجم مصرفی	دقت داده‌های آموزش	دقت داده‌های تست
معماری ۱	۹۳۴۸۷۴	۳ MB	۰.۹۹۲۶	۰.۹۷۷۶
معماری ۲	۹۳۰۷۱۴	۳.۵۵ MB	۰.۹۹۵۹	۰.۹۷۶۰
معماری ۳	۹۲۳۰۹۸	۳.۵۲ MB	۰.۹۹۷۰	۰.۹۸۰۸
معماری ۴	۷۲۰۹۰	۲۸۱.۶۰ KB	۰.۹۸۵۷	۰.۹۷۵۹
معماری ۵	۸۰۵۰	۳۱.۴۵ KB	۰.۹۷۷۹	۰.۹۷۵۸

بنابر این طبق نتایج به‌دست آمده، معماری شماره ۵ را به‌عنوان معماری برگزیده انتخاب می‌کنیم. معماری نهایی شبکه به‌صورت زیر ارائه می‌شود:



شکل ۱-۳: معماری شبکه

### ۲-۱-۳ آموزش شبکه

پس از انتخاب معماری شبکه، نوبت به آموزش شبکه می‌رسد. به صورت نرم افزاری شبکه مورد نظر را تعریف کرده و آن را با داده‌های مجموعه داده MNIST آموزش می‌دهیم تا از وزن‌های آن در فاز سخت افزاری استفاده کنیم.

کد نوشته شده برای پیاده سازی شبکه به صورت زیر است:

### Source Code 3-1: Model Definition

```

1
2 def define_model() -> Sequential:
3     # Define model.
4     model = Sequential()
5     model.add(ZeroPadding2D(padding=pad, input_shape=(input_size[0],
6         input_size[1], 1), name="padding_layer"))
7     model.add(Conv2D(conv_filter_num, conv_kernel_size, activation="relu",
8         padding="valid", kernel_initializer="he_uniform",
9         input_shape=(30, 30, 1), name="convolution_layer"))
10    model.add(MaxPooling2D(pool_size, name="max_pooling_layer"))
11    model.add(Flatten(name="flatten_layer"))
12    model.add(Dense(10, activation="softmax", name="dense_layer"))
13
14    # Compile model.
15    model.Compile(optimizer=Adam(), loss="categorical_crossentropy",
16        metrics=["accuracy"])
17
18    # Return model.
19    return model
20

```

با کامپایل کردن مدل نوشته شده خروجی شبکه تعریف شده به صورت زیر می شود:

Model: "sequential"

Layer (type)	Output Shape	Param #
padding_layer (ZeroPadding2D)	(None, 34, 34, 1)	0
convolution_layer (Conv2D)	(None, 28, 28, 4)	200
max_pooling_layer (MaxPooling2D)	(None, 14, 14, 4)	0
flatten_layer (Flatten)	(None, 784)	0
dense_layer (Dense)	(None, 10)	7,850

Total params: 8,050 (31.45 KB)

Trainable params: 8,050 (31.45 KB)

Non-trainable params: 0 (0.00 B)

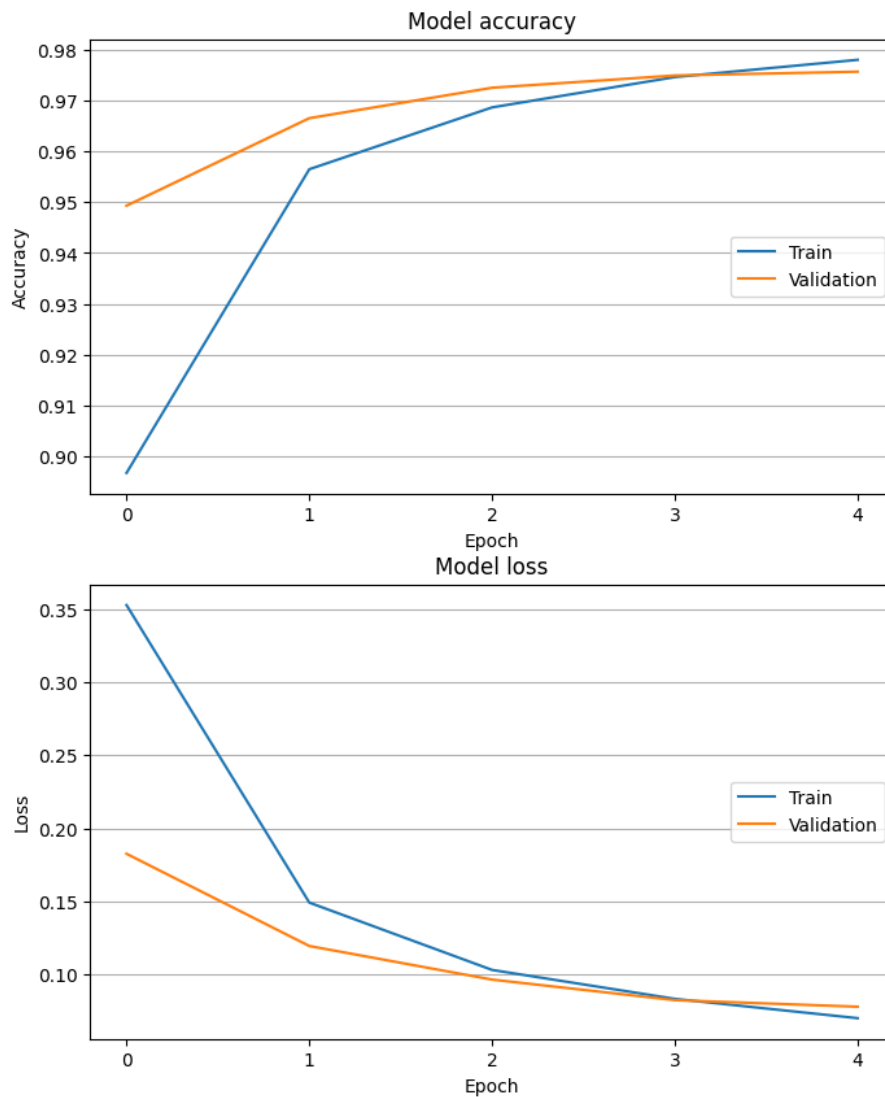
شکل ۳-۲: ساختار شبکه تعریف شده

پس از تعریف ساختار شبکه، شبکه را با استفاده از دیتاست MNIST در ۵ Epoch آموزش می دهیم. که نمودار Loss و Accuracy شبکه برای داده های Train و Validation به صورت شکل (۳-۳) به دست می آید:

دقت شبکه بر روی داده های آموزش و تست به ترتیب ۹۷۷۹٪ و ۹۷۵۸٪ به دست آمده است.

همچنین زمان انجام فاز Inference نرم افزاری برای ۱۰۰ داده از مجموعه داده MNIST، ۳۹/۹۷۳۲





شکل ۳-۳: نمودارهای Accuracy و Loss

میلی ثانیه به دست آمده است.

### ۳-۱-۳ ذخیره پارامترها

در نهایت تمامی وزن ها و پارامترهای مورد نیاز شبکه برای فاز سخت افزاری را در ۳ فایل با نام های:

- conv\_weights.h
- dense\_weights.h
- definitions.h

ذخیره می‌کنیم. فایل `conv_weights.h` شامل وزن‌های به‌دست آمده از لایه‌های کانولوشن، فایل `dense_weights.h` نیز شامل وزن‌های لایه Fully Connected و فایل `definitions.h` شامل برخی از ضرایب ثابت مورد استفاده در فاز سخت‌افزاری است.

در ادامه تمامی داده‌های تست MNIST شامل تصاویر و Label های مربوط به آن را در دو فایل `in.dat` و `out.dat` ذخیره می‌کنیم. مراحل انجام به طور کامل در فایل `gen_data.ipynb` مشخص شده است.

## ۲-۳ فاز سخت‌افزاری

در این فاز نیاز است کد های همه لایه‌ها را به‌صورت سخت‌افزاری در HLS نوشته و با همان معماری فاز نرم‌افزاری پیاده‌سازی کنیم.

### ۱-۲-۳ Fix-Point کردن داده‌ها

قبل از هرچیزی نیاز است وزن‌های Floating-Point ذخیره شده از فاز نرم‌افزاری را کاهش منابع مصرفی، به Fix-Point تبدیل کنیم. بدین منظور از نوع داده `ap_fixed` موجود در کتابخانه `ap_fixed.h` استفاده می‌کنیم.

این نوع داده به‌صورت زیر استفاده می‌شود:

```
ap_fixed<WL, IWL>
```

که:

- WL تعداد کل بیت‌ها (کل عرض داده).
- IWL تعداد بیت‌های بخش صحیح داده.

به عنوان مثال می‌توان نوشت:

```
ap_fixed<16, 8> my_number;
```

- 16: طول کل بیت‌ها (بخش صحیح + اعشاری).
- 8: تعداد بیت‌های بخش صحیح.

## ۲-۲-۳ لایه کانولوشن

برای مثال کد لایه کانولوشن به صورت زیر نوشته شده است:

**Source Code 3-2: HLS Implementation of Convolution Layer**

```

1 void convolution( float pad_img [PAD_IMG_ROWS][PAD_IMG_COLS],
2                 int filter,
3                 hls::stream<float> & conv_to_pool_stream )
4 {
5     float w_sum = 0.0; // Weighted sum.
6
7     // outer loops (r and c) loop over all pooling regions
8     conv_for_rows: for(int r = 0; r < IMG_ROWS; r += POOL_ROWS)
9     {
10         conv_for_cols: for(int c = 0; c < IMG_COLS; c += POOL_COLS)
11         {
12             // middle loops (pr and pc) loop over all pixels
13             // in selected pooling region
14             pool_for_rows: for(int pr = 0; pr < POOL_ROWS; ++pr)
15             {
16                 pool_for_cols: for(int pc = 0; pc < POOL_COLS; ++pc)
17                 {
18                     w_sum = 0.0;
19
20                     // inner loops (kr and kc) loop over all filter
21                     coefficients
22                     // applied to neighborhood of selected pixel
23                     krn_for_rows: for(int kr = 0; kr < KRN_ROWS; ++kr)
24                     {
25                         krn_for_cols: for(int kc = 0; kc < KRN_COLS; ++kc)
26                         {
27                             float w      = conv_weights[filter][kr][kc];
28                             float pixel = pad_img[r+pr+kr][c+pc+kc];
29                             w_sum += w * pixel;
30                         }
31                     }
32                     conv_to_pool_stream.write(relu(w_sum + conv_biases[
33 filter])));
34                 }
35             }
36         }
37     }
38 }

```

در این طراحی، کانولوشن چند کاناله را به صورت چند لایه کانولوشن تک کاناله پیاده سازی کرده ایم. در فاز نرم افزاری یک لایه کانولوشن ۴ کاناله داشتیم اما برای راحتی پیاده سازی، در فاز سخت افزاری، آن را به ۴ لایه کانولوشن تک کاناله شکسته ایم و آن را به صورت زیر پیاده سازی کرده ایم:

**Source Code 3-3: HLS Implementation of Convolution Layers**

```

1 void convolutional_layer(float pad_img0 [PAD_IMG_ROWS][
2 PAD_IMG_COLS],

```

```

3     float pad_img1 [PAD_IMG_ROWS][PAD_IMG_COLS],
4     float pad_img2 [PAD_IMG_ROWS][PAD_IMG_COLS],
5     float pad_img3 [PAD_IMG_ROWS][PAD_IMG_COLS],
6     hls::stream<float> conv_to_pool_streams [FILTERS] )
7     {
8         convolution(pad_img0, 0, conv_to_pool_streams[0]);
9         convolution(pad_img1, 1, conv_to_pool_streams[1]);
10        convolution(pad_img2, 2, conv_to_pool_streams[2]);
11        convolution(pad_img3, 3, conv_to_pool_streams[3]);
12    }
13

```

## ۳-۲-۳ لایه Fully Connected

کد لایه Fully Connected نیز به صورت زیر نوشته شده است:

Source Code 3-4: HLS Implementation of Dense Layer

```

1
2 void dense( hls::stream<float> & flat_to_dense_stream,
3            int filter,
4            hls::stream<float> & dense_to_softmax_stream )
5 {
6     float flat_value;
7     float dense_array[DENSE_SIZE] = { 0 };
8
9
10    #pragma HLS ARRAY_PARTITION variable=dense_array complete
11    #pragma HLS PIPELINE II=1
12
13    dense_for_flat: for(int i = 0; i < FLAT_SIZE / FILTERS; ++i)
14    {
15        flat_value = flat_to_dense_stream.read();
16
17        for(int d = 0; d < DENSE_SIZE; ++d)
18        {
19            int index = filter * (FLAT_SIZE / FILTERS) + i;
20            dense_array[d] += dense_weights[index][d] * flat_value;
21        }
22    }
23
24    for(int j = 0; j < DENSE_SIZE; ++j)
25    {
26        dense_to_softmax_stream.write(dense_array[j]);
27    }
28 }

```

### ۴-۲-۳ Flatten لایه

همچنین برای Flat کردن Feature Map های به دست آمده از خروجی لایه های کانولوشنی، ماژولی به نام flattening به صورت زیر تعریف شده است:

Source Code 3-5: HLS Implementation of Flatten Layers

```
1 void flattening( hls::stream<float> & pool_to_flat_stream,
2                 hls::stream<float> & flat_to_dense_stream )
3 {
4
5     #pragma HLS ARRAY_PARTITION variable=pool_to_flat_stream complete
6     #pragma HLS ARRAY_PARTITION variable=flat_to_dense_stream complete
7
8     flat_for_rows: for(int r = 0; r < POOL_IMG_ROWS; ++r)
9     {
10         flat_for_cols: for(int c = 0; c < POOL_IMG_COLS; ++c)
11         {
12             #pragma HLS UNROLL
13
14             flat_to_dense_stream.write(pool_to_flat_stream.read());
15         }
16     }
17 }
18 }
```

### ۵-۲-۳ ماژول اصلی (Top Module)

تاپ ماژول طراحی به نام CNN به صورت زیر تعریف شده است:

### Source Code 3-6: HLS Implementation of CNN Top Module

```
1 void cnn(float img_in[IMG_ROWS][IMG_COLS], float prediction[DIGITS])
2 {
3     /***** Pre-processing data. *****/
4
5     float pad_img0[PAD_IMG_ROWS][PAD_IMG_COLS] = { 0 };
6     normalization_and_padding(img_in, pad_img0);
7
8     #if 0
9     #ifndef __SYNTHESIS__
10         printf("Padded image.\n");
11         print_pad_img(pad_img);
12     #endif
13 #endif
14
15     /* Allow parallelism cloning the padded image. */
16     float pad_img1[PAD_IMG_ROWS][PAD_IMG_COLS];
17     float pad_img2[PAD_IMG_ROWS][PAD_IMG_COLS];
18     float pad_img3[PAD_IMG_ROWS][PAD_IMG_COLS];
19
20     float value;
21
22     clone_for_rows: for(int i = 0; i < PAD_IMG_ROWS; ++i)
23     {
24         clone_for_cols: for(int j = 0; j < PAD_IMG_COLS; ++j)
25         {
26             pad_img1[i][j] = pad_img0[i][j];
27             pad_img2[i][j] = pad_img0[i][j];
28             pad_img3[i][j] = pad_img0[i][j];
29         }
30     }
31
32
33     /* Parallel executions start here. */
34     dataflow_section(pad_img0, pad_img1, pad_img2, pad_img3, prediction);
35 }
36
```

### ۶-۲-۳ ماژول Data Flow

برای مشخص کردن جریان انجام محاسبات در Top Module، ماژولی به نام dataflow\_section تعریف شده است که کد آن به صورت زیر ارائه می شود:

### Source Code 3-7: HLS Implementation of Data flow Module

```
1 void dataflow_section( float pad_img0 [PAD_IMG_ROWS][PAD_IMG_COLS],
2                       float pad_img1 [PAD_IMG_ROWS][PAD_IMG_COLS],
3                       float pad_img2 [PAD_IMG_ROWS][PAD_IMG_COLS],
4                       float pad_img3 [PAD_IMG_ROWS][PAD_IMG_COLS],
5                       float prediction [DIGITS] )
6 {
7     #pragma HLS DATAFLOW
8     /***** Convolution layer. *****/
9
```

```

10  /*
11  An array to collect the convolution results:
12  FILTERS resulting feature maps, one for each filter.
13  */
14
15  hls::stream<float, IMG_ROWS * IMG_COLS>
16  conv_to_pool_streams[FILTERS];
17
18  // Convolution with relu as activation function.
19  convolutional_layer(pad_img0, pad_img1, pad_img2, pad_img3,
20  conv_to_pool_streams);
21
22  #if 0
23  #ifndef __SYNTHESIS__
24  // Print results.
25  print_features(conv_to_pool_streams);
26  #endif
27  #endif
28
29  /***** Maxpooling layer. *****/
30  hls::stream<float, POOL_IMG_ROWS * POOL_IMG_COLS>
31  pool_to_flat_streams[FILTERS];
32
33  max_pooling_layer(conv_to_pool_streams, pool_to_flat_streams);
34
35  #if 0
36  #ifndef __SYNTHESIS__
37  print_pool_features(pool_to_flat_streams);
38  #endif
39  #endif
40
41  /***** Flatten layer. *****/
42  hls::stream<float, FLAT_SIZE / FILTERS> flat_to_dense_streams[FILTERS
43  ];
44  flattening_layer(pool_to_flat_streams, flat_to_dense_streams);
45
46  /***** Dense layer. *****/
47  hls::stream<float, DENSE_SIZE> dense_to_softmax_streams [FILTERS];
48  dense_layer(flat_to_dense_streams, dense_to_softmax_streams);
49
50  /***** Softmax. *****/
51  dense_layer_soft_max(dense_to_softmax_streams, prediction);
52  }

```

ماژول‌های دیگری برای انجام عملیات‌هایی مانند Normalization، Pooling، Zero Padding و ... تعریف شده است که به‌علت طولانی نشدن گزارش آورده نشده است.

### ۳-۳ سنتر مدل

در نهایت ماژول CNN را سنتر می‌کنیم. گزارشات سنتر شبکه در شکل‌های «۴-۳» و «۵-۳» و «۶-۳» آورده شده است.

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
cnr				-0.00	46403	4.640E5	-	46404	-	no	116	300	53635	79001
cnr_Pipeline_1				-	1158	1.158E4	-	1158	-	no	0	0	13	63
cnr_Pipeline_pad_for_rows_pad_for_cols				-	800	8.000E3	-	800	-	no	0	0	253	209
cnr_Pipeline_clone_for_rows_clone_for_cols				-	1166	1.166E4	-	1166	-	no	0	0	490	502
dataflow_section	!! Violation		-0.00		43272	4.330E5	-	43272	-	no	104	300	51299	75554

شکل ۳-۴: منابع مصرفی پس از سنتز (الف)

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
cnr				-0.00	46403	4.640E5	-	46404	-	no	116	300	53635	79001	0
cnr_Pipeline_1				-	1158	1.158E4	-	1158	-	no	0	0	13	63	0
cnr_Pipeline_pad_for_rows_pad_for_cols				-	800	8.000E3	-	800	-	no	0	0	253	209	0
cnr_Pipeline_clone_for_rows_clone_for_cols				-	1166	1.166E4	-	1166	-	no	0	0	490	502	0
dataflow_section	!! Violation		-0.00		43272	4.330E5	-	43272	-	no	104	300	51299	75554	0

شکل ۳-۵: منابع مصرفی پس از سنتز (ب)

## ۴-۳ تست مدل

در نهایت فایل test bench ای برای طراحی نوشته شده است که ۱۰۰ تصویر ابتدایی را از فایل تصاویر تولید شده در فاز قبل را بخواند و برای پردازش به شبکه بدهد. و خروجی شبکه میزان دقت تشخیص اعداد و زمان مصرف شده به ازای تمام فرایندها می باشد.

برای مثال، شبکه ما ۱۰۰ تصویر ابتدایی از مجموعه داده های تست دیتاست MNIST را با دقت ۱۰۰٪ تشخیص می دهد «شکل ۳-۷»:

شبکه ما فاز Inference را به ازای ۱۰۰ تصویر در ۴/۴۵ میلی ثانیه انجام داده است. در صورتی که در فاز نرم افزاری همین تعداد تصویر در مدت زمان ۳۹/۹۷ میلی ثانیه انجام شده است.



Cosimulation Report for 'cnn'

General Information

Date: Mon Dec 16 01:32:37 PM +0330 2024

Version: 2023.2 (Build 4023990 on Oct 11 2023)

Project: CNN

Status: Pass

Solution: solution1 (Vivado IP Flow Target)

Product family: zynq

Target device: xc7z010-clg400-3

Cosim Options

Tool: Vivado XSIM

RTL: VHDL

Performance Estimates

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency	Total Execution Time
cnn	46390	46390	46390	46389	46389	46389	463899
cnn_Pipeline_1							
cnn_Pipeline_pad_for_rows_pad_for_cols							
cnn_Pipeline_clone_for_rows_clone_for_cols							
dataflow section							

شکل ۳-۶: منابع مصرفی پس از سنتز (ج)

Cosimulation Report for 'cnn'

General Information

Date: Mon Dec 16 01:32:37 PM +0330 2024

Version: 2023.2 (Build 4023990 on Oct 11 2023)

Project: CNN

Status: Pass

Solution: solution1 (Vivado IP Flow Target)

Product family: zynq

Target device: xc7z010-clg400-3

Cosim Options

Tool: Vivado XSIM

RTL: VHDL

Performance Estimates

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency	Total Execution Time
<div><div></div><div>cnn</div></div>	46390	46390	46390	46389	46389	46389	463899
<div><div></div><div>cnn.Pipeline_1</div></div>							
<div><div></div><div>cnn.Pipeline_pad_for_rows_pad_for_cols</div></div>							
<div><div></div><div>cnn.Pipeline_clone_for_rows_clone_for_cols</div></div>							
<div><div></div><div>dataflow section</div></div>							

شکل ۳-۷: Accuracy پیش‌بینی به‌ازای ۱۰۰ تصویر

هرچه تعداد تصاویر را زیاد کنیم دقت شبکه کم می‌شود. برای مثال به ازای ۵۰۰ تصویر دقت تشخیص شبکه ۹۹ درصد به‌دست می‌آید.

Cosimulation Report for 'cnn'

General Information

Date: Mon Dec 16 01:32:37 PM +0330 2024

Version: 2023.2 (Build 4023990 on Oct 11 2023)

Project: CNN

Status: Pass

Solution: solution1 (Vivado IP Flow Target)

Product family: zynq

Target device: xc7z010-clg400-3

Cosim Options

Tool: Vivado XSIM

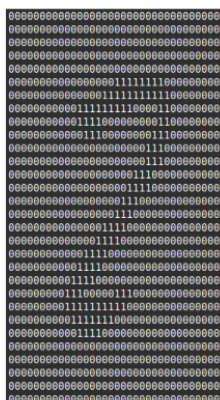
RTL: VHDL

Performance Estimates

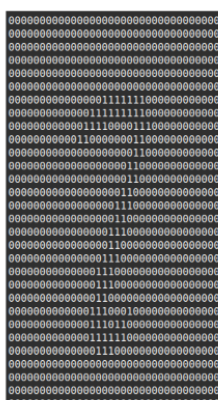
Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency	Total Execution Time
<div><div></div><div>cnn</div></div>	46390	46390	46390	46389	46389	46389	463899
<div><div></div><div>cnn.Pipeline_1</div></div>							
<div><div></div><div>cnn.Pipeline_pad_for_rows_pad_for_cols</div></div>							
<div><div></div><div>cnn.Pipeline_clone_for_rows_clone_for_cols</div></div>							
<div><div></div><div>dataflow section</div></div>							

شکل ۳-۸: Accuracy پیش‌بینی به‌ازای ۵۰۰ تصویر

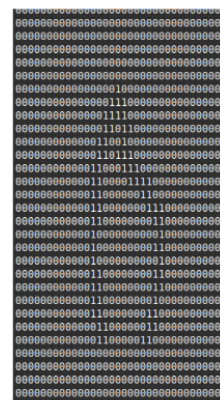
در این حالت شبکه ۵ تصویر زیر را به اشتباه تشخیص داده است:



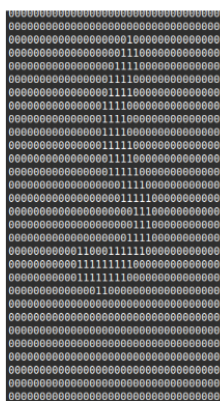
(ج) تشخیص اشتباه عدد ۲



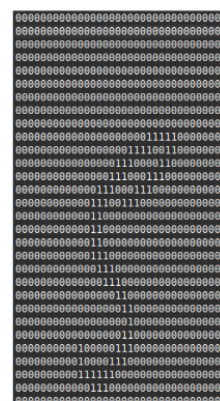
(ب) تشخیص اشتباه عدد ۲



(آ) تشخیص اشتباه عدد ۰



(ه) تشخیص اشتباه عدد ۵



(د) تشخیص اشتباه عدد ۵

شکل ۳-۹: تشخیص‌های اشتباه شبکه

مقادیری که شبکه برای هر خروجی اشتباه به‌دست آورده است نیز به‌صورت زیر ارائه می‌شود:

Prediction for C:	Prediction for B:	Prediction for A:
0: 0.268470	0: 0.000006	0: 0.000014
1: 0.006770	1: 0.000024	1: 0.000002
2: 0.000006	2: 0.078408	2: 0.000000
3: 0.000007	3: 0.048098	3: 0.806685
4: 0.025127	4: 0.000000	4: 0.001617
5: 0.002049	5: 0.000003	5: 0.037086
6: 0.006413	6: 0.000000	6: 0.000001
7: 0.627097	7: 0.596224	7: 0.000960
8: 0.047987	8: 0.271232	8: 0.003560
9: 0.016075	9: 0.006006	9: 0.150075

Prediction for E:	Prediction for D:
0: 0.000000	0: 0.000006
1: 0.000000	1: 0.005681
2: 0.137538	2: 0.000050
3: 0.003166	3: 0.955743
4: 0.000000	4: 0.000271
5: 0.000000	5: 0.037680
6: 0.000000	6: 0.000232
7: 0.000023	7: 0.000000
8: 0.859272	8: 0.000259
9: 0.000001	9: 0.000077

## فصل ۴

## نتایج

در این فصل به بررسی نتایج به دست آمده از سنتزهای مختلف می‌پردازیم. مدل توضیح داده شده در فصل قبل را در سه حالت مختلف سنتز کرده و خروجی به دست آمده را از نظر تاخیر و منابع مصرفی با هم مقایسه می‌کنیم.

### ۱-۴ بدون بهینه‌سازی

در این قسمت از هیچ Pragma ای برای بهینه‌سازی کد استفاده نشده است. منابع مصرفی و تاخیرها برای این حالت به صورت شکل «۱-۴» گزارش می‌شود.

cnn.cpp

Synthesis Summary(solution1)

Synthesis Summary Report of 'cnn'

General Information

Date: Thu Jan 23 14:50:59 2025

Version: 2023.2 (Build 4023990 on Oct 11 2023)

Project: CNN\_Optimal

Solution: solution1 (Vivado IP Flow Target)

Product family: zynq

Target device: xc7z010-clg400-1

Timing Estimate

Target

Estimated

Uncertainty

10.00 ns

8.643 ns

2.70 ns

Performance & Resource Estimates

Modules

Loops

ops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
pipeline_1			-1.34	-	46544	4.650E5	-	46545	-	no	116	300	53242	79297	0
ipeline_pad_for_rows_pad_for_cols	Timing Violation		-0.63	-	1158	1.158E4	-	1158	-	no	0	0	13	51	0
ipeline_clone_for_rows_clone_for_cols	Timing Violation		-1.34	-	804	8.040E3	-	804	-	no	0	0	208	249	0
ow_section	II Violation		-1.12	-	1167	1.167E4	-	1167	-	no	0	0	509	492	0
			-1.12	-	43408	4.340E5	-	43408	-	no	104	300	50932	75786	0

Performance Pragma

شکل ۱-۴: خروجی سنتز بهینه نشده

در این حالت تاخیر در بدترین (بیشترین) حالت نسبت به دو حالت دیگر قرار دارد اما منابع مصرفی (LUT ها، Flip Flop ها و ... در کمترین مقدار خود قرار دارند.

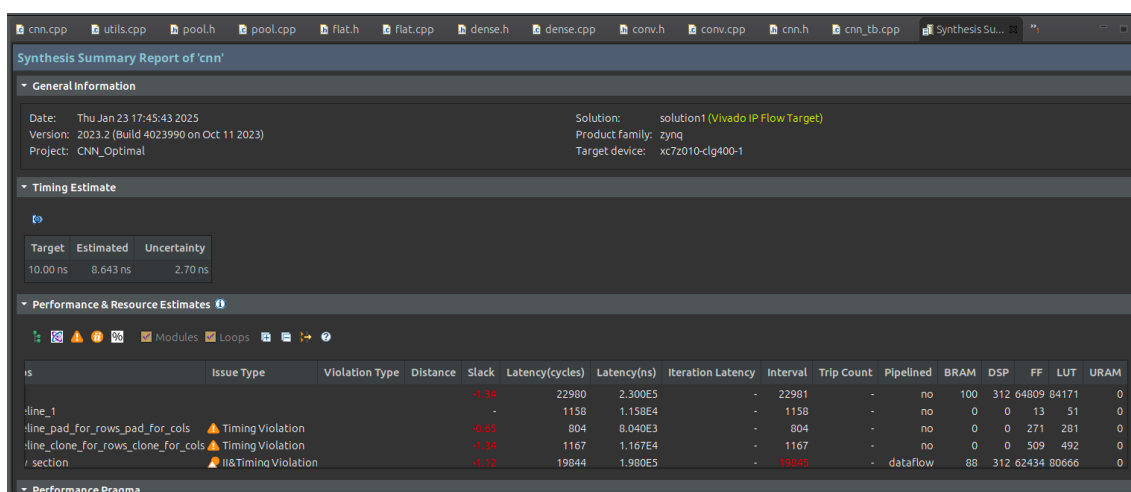
## ۲-۴ نیمه بهینه

در این حالت صرفاً با استفاده از دو پراگما:

1. `#pragma HLS PIPELINE`

2. `#pragma HLS UNROLL`

حلقه‌ها را Unroll می‌کنیم و ساختاری Pipeline طور در محاسبات حلقه‌ها ایجاد کنیم. در این حالت، میزان منابع مصرفی نسبت به حالت قبل، افزایش داشته است اما تاخیر کلی مدار کاهش چشمگیری پیدا کرده است. خروجی این حالت به صورت شکل «۲-۴» گزارش می‌شود.



The screenshot shows the 'Synthesis Summary Report of 'cnn'' in a Vivado IDE. The report is divided into sections: General Information, Timing Estimate, and Performance & Resource Estimates. The Timing Estimate section shows a target of 10.00 ns, an estimated time of 8.643 ns, and an uncertainty of 2.70 ns. The Performance & Resource Estimates section shows a table of violations and resource usage.

is	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
line_1			-1.34	22980	2.300E5	-	22981	-	-	no	100	312	64809	84171	0
line_pad_for_rows_pad_for_cols	Timing Violation		-	1158	1.158E4	-	1158	-	-	no	0	0	13	51	0
line_clone_for_rows_clone_for_cols	Timing Violation		-0.63	804	8.040E3	-	804	-	-	no	0	0	271	281	0
/ section	II&Timing Violation		-1.12	1167	1.167E4	-	1167	-	-	no	0	0	509	492	0
			-1.12	19844	1.980E5	-	19845	-	-	dataflow	88	312	62434	80666	0

شکل ۲-۴: خروجی سنتز نیمه بهینه

## ۳-۴ بهینه‌سازی کامل

در نهایت در آخرین گام بهینه‌سازی، پراگماهای دیگری مثل:

1. `#pragma HLS ARRAY_PARTITION variable=X complete`

2. `#pragma HLS ARRAY_PARTITION variable=X block factor=4 dim=1`

را به طراحی اضافه می‌کنیم تا با شکستن ورودی مازول‌های مختلف (که عمدتاً ماتریس‌های بزرگی هستند) با اجزاء کوچکتر، سریعتر محاسبات را انجام دهند اما با انجام این کار به صورت حسی نیز افزایش منابع مصرفی

توجیح می‌شود. خروجی‌های ارائه شده در شکل «۳-۴» این حرف را تایید می‌کند. منابع مصرفی به ضدت افزایش پیدا کرده است. و در مقابل آن تاخیر نه‌تنها بهتر نشده است، بلکه مقدار بسیار کمی افزایش یافته است. این پدیده را اینطور می‌توان توجیح نمود که ابزار سنتز در تلاش برای سنتز مدار و Schedule کردن مدار تولیدی بوده است اما به دلیل منابع<sup>۱</sup> محدود در این مدل FPGA انتخاب شده<sup>۲</sup> ابزار نتوانسته است به درستی فرآیند سنتز، جایابی<sup>۳</sup> و مسیریابی<sup>۴</sup> را به درستی انجام دهد؛ بنابراین تاخیر کمی نسبت به حالت قبل زیاد شده است.

Synthesis Summary (solution1) 33cnn.cpputils.hutils.cpppool.cpp

Synthesis Summary Report of 'cnn'

General Information

Date: Fri Jan 24 15:13:35 2025

Version: 2023.2 (Build 4023990 on Oct 11 2023)

Project: CNN\_Optimal

Solution: solution1 (Vivado IP Flow Target)

Product family: zynq

Target device: xc7z010-clg400-1

Timing Estimate

TargetEstimatedUncertainty

10.00 ns10.274 ns2.70 ns

Performance & Resource Estimates

Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
line_1			-2.97	22999	2.360E5	-	23000	-	no	448	732	816987	453049	0
line_pad_for_rows_pad_for_cols	Timing Violation	-1.02	812	8.120E3	-	812	-	no	0	0	492	460	0	0
line_clone_for_rows_clone_for_cols	Timing Violation	-1.34	1169	1.169E4	-	1169	-	no	0	0	818	705	0	0
v section	II Violation	-2.97	19044	2.040E5	-	19044	-	dataflow	104	732	813274	444638	0	0

شکل ۳-۴: خروجی سنتز بهینه کامل

توضیحات این سه قسمت در جدول زیر خلاصه می‌شود:

جدول ۱-۴: اطلاعات مربوط به معماری‌های مختلف

مدل	تاخیر (ns)	تعداد LUT	میزان بهبود تاخیر	میزان افزایش منابع
بدون بهینه‌سازی	۴/۶۵۰	۷۹۲۹۷	-	-
نیمه بهینه	۲/۳۰۰	۸۴۱۷۱	۲۰۲ %	۱/۰۶ %
بهینه کامل	۲/۳۶۰	۴۵۳۰۴۹	۱۹۷ %	۵/۷۱ %

ذکر این نکته الزامی است که؛ واژه بهینه کامل و نیمه بهینه از بابت مصرف pragma ها استفاده شده است و نه از بابت خروجی منابع مصرفی و تاخیر. با توجه به گزارشات به دست آمده، مدل نیمه بهینه عملکرد بهتری را از دو مدل دیگر هم از نظر تاخیر و هم از نظر منابع مصرفی داشته است.

<sup>1</sup>Resource

<sup>۲</sup>در این طراحی از FPGA ای با پارت نامبر xc7z010-clg400-1 استفاده شده است.

<sup>3</sup>Placement

<sup>4</sup>Routing

## فصل ۵

### نتیجه‌گیری و جمع‌بندی

این پژوهش به بررسی و پیاده‌سازی شبکه‌های عصبی CNN بر روی سخت‌افزار FPGA با هدف بهبود سرعت پردازش و کاهش مصرف انرژی در فاز استنتاج پرداخته است. شبکه‌های عصبی پیچشی به دلیل توانایی بالا در استخراج ویژگی‌های سلسله‌مراتبی از داده‌های خام، به‌ویژه در مسائل طبقه‌بندی تصاویر، عملکرد بسیار خوبی دارند. با این حال، اجرای این مدل‌ها در کاربردهای عملی با چالش‌هایی مانند پیچیدگی محاسباتی بالا و نیاز به منابع سخت‌افزاری کارآمد مواجه است. در این راستا، FPGAها به دلیل قابلیت پردازش موازی، مصرف انرژی کمتر و انعطاف‌پذیری در طراحی، گزینه‌ای ایده‌آل برای پیاده‌سازی مدل‌های CNN در کاربردهای بی‌درنگ و محیط‌های با منابع محدود محسوب می‌شوند.

در این پژوهش، ابتدا معماری مناسب برای شبکه عصبی پیچشی با توجه به معیارهای دقت، خطا و حجم پارامترها انتخاب شد. سپس، مدل انتخاب‌شده با استفاده از مجموعه داده MNIST آموزش داده شد و پارامترهای آن برای پیاده‌سازی سخت‌افزاری ذخیره گردید. در فاز سخت‌افزاری، کدهای مربوط به لایه‌های مختلف شبکه (مانند کانولوشن، Pooling، Connected Fully و Flatten) به زبان C نوشته شده و با استفاده از ابزار HLS بر روی FPGA پیاده‌سازی شدند. در نهایت، مدل سنتز شده و عملکرد آن از نظر تاخیر و منابع مصرفی مورد ارزیابی قرار گرفت.

نتایج نشان داد که استفاده از تکنیک‌های بهینه‌سازی مانند Pipeline و Unroll می‌تواند به طور چشمگیری تاخیر پردازش را کاهش دهد، هرچند که این بهبود با افزایش منابع مصرفی همراه است. در این پژوهش، مدل نیمه بهینه که از ترکیب این تکنیک‌ها استفاده کرده بود، بهترین عملکرد را از نظر تعادل بین تاخیر و منابع مصرفی ارائه داد. این مدل توانست فاز استنتاج را برای ۱۰۰ تصویر در مدت زمان ۴/۴۵ میلی‌ثانیه انجام دهد، در حالی که همین فرآیند در فاز نرم‌افزاری ۳۹/۹۷ میلی‌ثانیه طول کشید. همچنین، دقت شبکه در تشخیص ارقام دست‌نویس به ۱۰۰٪ برای ۱۰۰ تصویر اول و ۹۹٪ برای ۵۰۰ تصویر رسید.

در مجموع، این پژوهش نشان داد که پیاده‌سازی شبکه‌های عصبی پیچشی بر روی FPGA می‌تواند به عنوان یک راه‌حل کارآمد برای کاربردهای بی‌درنگ و محیط‌های با منابع محدود مورد استفاده قرار گیرد. با این حال، انتخاب استراتژی‌های بهینه‌سازی مناسب و تعادل بین تاخیر و منابع مصرفی از جمله چالش‌های اصلی در این زمینه است که نیاز به بررسی‌های بیشتر دارد. نتایج این پژوهش می‌تواند به عنوان پایه‌ای برای توسعه سیستم‌های هوشمند با کارایی بالا و مصرف انرژی کم در آینده مورد استفاده قرار گیرد.



# Bibliography

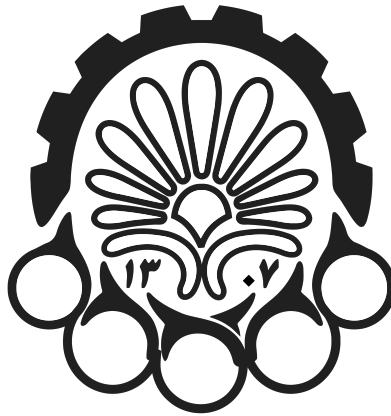
- [1] A. Vidhya. Beginner-friendly project: Cat and dog classification using cnn, 2021. Accessed: 2025-01-23.
- [2] InAccel. Gpus vs fpgas: Which one is better in dl and data centers applications?, 2020. Accessed: 2025-01-23.
- [3] W. contributors. Convolutional neural network. [https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network](https://en.wikipedia.org/wiki/Convolutional_neural_network), 2025. Accessed: 2025-01-23.
- [4] IndoML. Student notes: Convolutional neural networks (cnn) introduction, 2018. Accessed: 2025-01-23.
- [5] SuperDataScience. Convolutional neural networks (cnn) step 1b: Relu layer, n.d. Accessed: 2025-01-23.
- [6] P. with Code. Max pooling, n.d. Accessed: 2025-01-23.
- [7] A. I. Aramendia. Convolutional neural networks (cnns): A complete guide, n.d. Accessed: 2025-01-23.

## **Abstract**

Convolutional Neural Networks (CNNs) are among the most widely used models in the field of deep learning, particularly in applications such as image recognition and visual data processing. Given the growing demand for fast and efficient processing, hardware platforms like FPGA have become an ideal choice for implementing these networks due to their parallel processing capabilities and low power consumption.

In this project, the goal was to implement a Convolutional Neural Network for handwritten digit recognition on an FPGA using High-Level Synthesis (HLS). The implementation process consisted of two main phases: In the software phase, the network was trained, and its weights were stored. In the hardware phase, the stored weights were transferred to the FPGA, and the input data was fed into the network. The outputs were then processed to evaluate the performance and accuracy of recognition. This implementation combines the high efficiency and flexibility of FPGA with the power of deep learning, enabling enhanced productivity in practical applications.

**Keywords:** Neural Networks, Deep Learning, CNN, FPGA



Amirkabir University of Technology

(Tehran Polytechnic)

Department of Computer Engineering

Reconfigurable Systems Design Final Project Report

# **Design and Simulation of CNN Neural Network for Hand Written Digit Recognition Using HLS**

By:

**Reza Adinepour**

Supervisor:

**Prof. Saheb Zamani**

Jan 2025