

Uniwersytet Jagielloński w Krakowie
Wydział Fizyki, Astronomii i Informatyki Stosowanej

Wojciech Lepich

Nr albumu: 1146600

Rozpoznawanie cyfr przez sieć neuronową zaimplementowaną na układzie FPGA

Praca licencjacka
na kierunku Informatyka

Praca wykonana pod kierunkiem
dr. Grzegorza Korcyła
z Zakładu Technologii Informatycznych

Kraków 2020

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

.....
Kraków, dnia

.....
Podpis autora pracy

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

.....
Kraków, dnia

.....
Podpis kierującego pracą

Spis treści

1	Wstęp	3
2	Teoria	4
2.1	Architektura FPGA	4
2.2	Przetwarzanie obrazu	4
2.2.1	Formaty pikseli	4
2.3	Sieci neuronowe	5
3	Opis projektu	6
3.1	Zarys projektu	6
3.2	Platforma	6
3.3	Sieć neuronowa	6
3.4	hls4ml	6
3.4.1	Idea hls4ml	6
3.4.2	Precyzja danych	7
3.5	GStreamer	8
3.5.1	Caps filter	8
3.5.2	fpsdisplaysink	8
3.5.3	videoconvert	8
3.5.4	videocrop	8
3.5.5	videoscale	9
3.5.6	videobox	9
3.5.7	xlnxvideosrc i xlnxvideosink	9
3.6	Używanie sieci	9
3.6.1	Generowanie projektu	9
3.6.2	Funkcja	9
3.6.3	Dostosowanie sieci	10
3.7	Część neuralnet	10
3.8	Część gstdxnnet	10
3.9	Małe podsumowanie	10
4	Wyniki i dyskusja	11
4.1	Ewaulacja modelu	11
4.2	Symulacja	11
4.3	Dane rzeczywiste	11
5	Podsumowanie	12

1 Wstęp

Tutaj wstęp

2 Teoria

2.1 Architektura FPGA

Field-programmable gate array (FPGA) to układy scalone, które mogą być elektronicznie przeprogramowane bez potrzeby demontażu samego układu z urządzenia. W porównaniu do układów ASIC znacznie taniej zaprojektować pierwszy działający układ. Elastyczna natura układów FPGA wiąże się z większym zużyciem powierzchni krzemu, opóźnień oraz zużycia energii. (FPGA architecture: survey and challenges)

Podstawowa struktura układów FPGA składa się z różnych bloków logicznych, które mogą być łączone zależnie od projektu. Przykładami takich bloków są: DSP (jednostka przeprowadzająca obliczenia dodawania/mnożenia), LUT (look-up table, de facto tablica prawdy dowolnej funkcji boolowskiej), Flip Flop (przechowują wynik LUT), BRAM (block RAM, pamięć dwuportowa, jest w stanie przechowywać względnie dużą ilość danych).

Układy FPGA przeważnie pracują na kilku-, kilkunastukrotnie niższych częstotliwościach niż CPU. Wysoką wydajność zawdzięczają zrównolegleniu obliczeń.

Dodać
przy-
pis

Przypis

2.2 Przetwarzanie obrazu

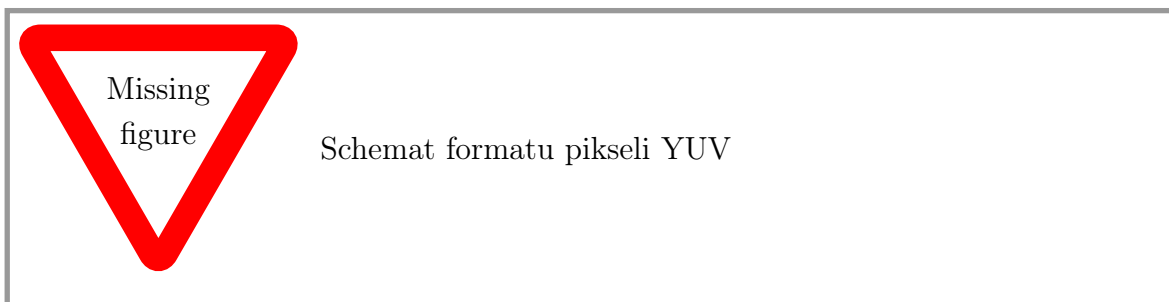
Cyfrowe przetwarzanie obrazu jest problemem wymagającym dużych mocy obliczeniowych ze względu na ilość danych do przetworzenia. Nieskompresowany kolorowy obraz z pikselami w formacie RGB (po 8 bitów na kolor) o wysokości 720 pikseli i szerokości 1280 pikseli to 22118400 bitów ($\approx 2,5\text{MB}$). Obraz przetwarzany w czasie rzeczywistym, na przykład z kamery, zwiela wielokrotnie tę liczbę o liczbę klatek na sekundę (przy trzydziestu klatkach na sekundę liczba danych rośnie do około 79 megabajtów na sekundę). Należy również pamiętać, że dane są dwuwymiarowe co jest ważne przy problemach związanych z rozpoznawaniem wzorców, klasyfikacją przedmiotów na obrazie, filtrowania w celu rozmycia lub wyostrenia obrazów, itp.

2.2.1 Formaty pikseli

Jest wiele modeli przestrzeni barw (a co za tym idzie, sposobów kodowania pikseli) między innymi:

- RGB, używany w aparatach, skanerach, telewizorach
- CMYK, używany w druku wielobarwnym
- HSV
- YUV

Składowe dwóch ostatnich przestrzeni barw oddzielają informację o jasności od informacji o kolorach. Model barw YUV składa się z kanału luminacji Y oraz kanałów kodujących barwę U oraz V, są to kolejno składowa niebieska i składowa czerwona. W projekcie użyty jest format pikseli YUY2 (znany też pod nazwą YUYV), w którym na dwa piksele przypadają 32 bity.



Licząc od najstarszego bitu pierwsze osiem bitów przypada na Y0, to jest luminacja pierwszego piksela, następne osiem bitów na U0, kolejne osiem bitów to luminacja drugiego piksela, a pozostałe bity to składowa czerwona V0. Dla obydwóch pikseli składowe U i V są wspólne. Co istotne w projekcie, łatwo oddzielić luminację, która jest używana w przetwarzaniu obrazu.

2.3 Sieci neuronowe

Sztuczna sieć neuronowa (SSN) jest modelem zdolnym do odwzorowania złożonych funkcji. Najprostsze sieci są zbudowane ze sztucznych neuronów, z których każdy posiada wiele wejść oraz jedno wyjście, które może być połączone z wejściami wielu innych neuronów. Każde z wejść neuronu jest związane ze znalezioną w procesie trenowania wagą. Wartość wyjścia to obliczony wynik funkcji aktywacji z sumy ważonych wejść. Sieć może mieć wiele warstw neuronów ukrytych, których wejściami są wyjścia neuronów z poprzedniej warstwy.

Sieci neuronowe są stosowane w problemach związanych z predykcją, klasyfikacją, przetwarzaniem i analizowaniem danych. Do ich zastosowania nie jest potrzebna znajomość algorytmu rozwiązania danego problemu. Obliczenia w sieciach są wykonywane równolegle w każdej warstwie, dzięki czemu implementacja sieci na układzie FPGA może działać wielokrotnie szybciej niż na CPU, pomimo niższej częstotliwości układu.

Schemat
sieci

Przypis

3 Opis projektu

3.1 Zarys projektu

Celem projektu jest napisanie wtyczki do frameworka GStreamer wykorzystującej sieć neuronową do rozpoznawania cyfr w czasie rzeczywistym na układzie Xilinx Zynq MPSoC oraz stworzenie odpowiedniego potoku danych korzystając z bibliotek GStreamer. Zadaniem spoczywającym na innych elementach potoku jest obsługa kamery, kadrowanie i skalowanie obrazu oraz wyświetlenie go na końcowym urządzeniu.

Schemat
nr 1

3.2 Platforma

Sprzęt wykorzystany w projekcie to Xilinx Zynq UltraScale+ MPSoC ZCU104. Na jednym układzie znajduje się czterordzeniowy procesor ARM Cortex-A53, dwurdzeniowy procesor ARM Cortex-R5, układ graficzny Mali-400 oraz zasoby FPGA. Całość projektu została oparta o platformę Xilinx reVISION. Przetwarzane dane dostarczane są z kamery USB, która była dołączona w zestawie z płytą Zynq. Urządzeniem końcowym jest telewizor połączony przewodem HDMI z płytą.

Zdjęcia
stano-
wiska

3.3 Sieć neuronowa

Architektura sieci została dobrana uwzględniając dostępne zasoby programowalnej logiki na płycie, a także możliwości sprzętu na którym dokonywana była jej synteza. Dla problemu klasyfikowania obrazów dobrze nadają się sieci splotowe (konwolucyjne, ang. convolutional neural networks — CNN), których przykładem jest popularna sieć LeNet-5. Architektura ta zawiera zarówno w pełni połączone warstwy oraz warstwy splotowe i łączące. Niestety z powodu ograniczeń sprzętowych w pracy nie została użyta ta architektura.

przypis

Model wykorzystany w projekcie posiada 2 warstwy ukryte, posiadające kolejno 12 i 40 neuronów aktywowanych funkcją ReLU, i warstwę wyjściową złożoną z 10 neuronów z funkcją aktywacji softmax. Do stworzenia sieci wykorzystano bibliotekę TensorFlow. Sieć uczona była na danych z bazy MNIST składającej się łącznie z 70000 przykładów cyfr na obrazach o wielkości 28×28 pikseli, z których każdy przedstawiony jest jako wartość od 0 (kolor czarny) do 255 (kolor biały). Próbkę została podzielona na zbiór uczący, liczący 60000 próbek, oraz zbiór do testów z pozostałych 10000 cyfr.

przypis

Oryginalnie cyfry są białe na czarnym tle co zwiększa dokładność działania sieci (więcej 0 w danych), natomiast trzeba wziąć to pod uwagę przy późniejszym wykorzystaniu sieci, ponieważ docelowo sieć ma rozpoznawać czarne cyfry na białym tle.

przypis
- ba-
dania
własne

3.4 hls4ml

3.4.1 Idea hls4ml

Celem projektu hls4ml jest automatyczne przetłumaczenie wytrenowanego modelu, architektury i wag, do projektu syntezy wysokiego poziomu (HLS). Czerwona część schematu pokazuje ogólną organizację pracy przy projektowaniu odpowiedniego modelu

Tutaj
sche-
mat
hls4ml
work-
flow

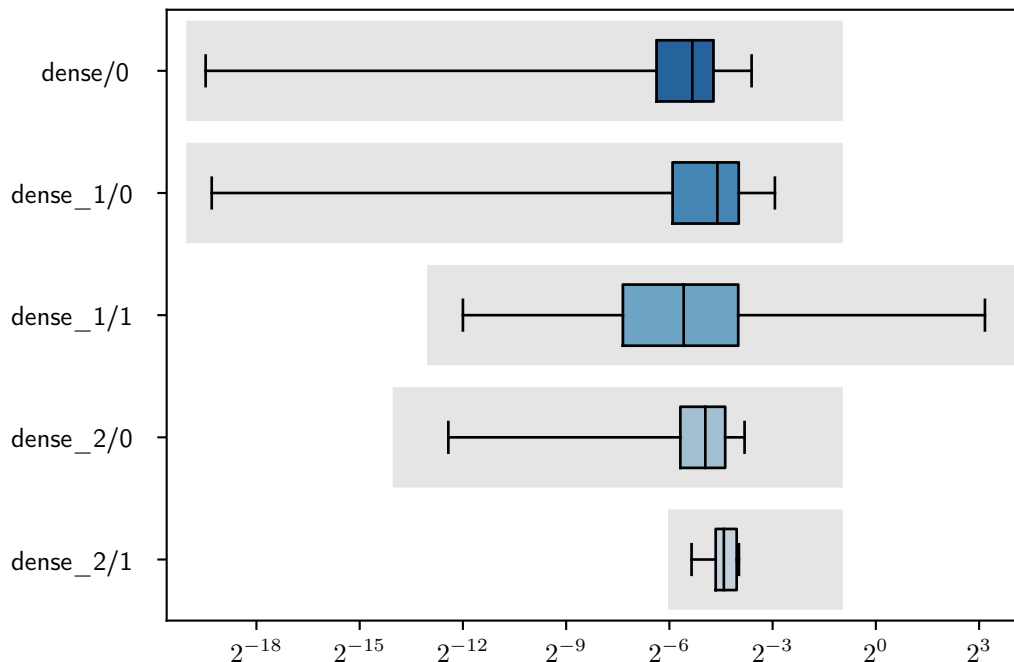
uczenia maszynowego. Niebieska część należy do hls4ml, który tłumaczy dostarczony model z wagami do syntetyzowalnego kodu, który następnie można włączyć do większego projektu lub zaimplementować jako samodzielną część na FPGA. Generowany projekt jest parametryzowany przez plik konfiguracyjny yml zawierający ścieżkę do pliku z zapisanym modelem wraz z wagami, typy danych kodujące wartości wag, nazwę docelowego układu FPGA.

3.4.2 Precyzja danych

Typ danych używany w przekonwertowanym modelu to duże liczby całkowite (`ap_int`) oraz liczby stałoprzecinkowe (`ap_fixed`). Precyzję obu można ustalić do jednego bita. Obliczenia przeprowadzane na liczbach o mniejszej precyzji są szybsze, natomiast zbyt niska może poskutkować bezużytecznością zsyntetyzowanej sieci. Aby odpowiednio dobrać precyzję wag skorzystano z pythonowej biblioteki `hls4ml.profiling`.

Program korzystający z funkcji dostarczanych przez tę bibliotekę analizuje plik konfiguracyjny yml oraz model z pliku h5. Wynikiem działania programu jest wykres przedstawiający rozkład wartości wag każdej z warstw modelu otrzymanych w procesie trenowania. Szare pole w tle wykresu przedstawia zakres wartości, które obejmowane są przez precyzję określoną w pliku konfiguracyjnym. Dobrym punktem początkowym jest wybranie takiej liczby bitów dla każdej z warstw, która obejmuje wszystkie możliwe wagi. Dalsze ustalanie precyzji można wykonać w trakcie analizy wyników symulacji.

Rysunek 1: Rozkład wartości wag modelu



3.5 GStreamer

Zsyntetyzowana sieć jest częścią projektu. Potrzebne również dostarczenie danych do sieci oraz przedstawienie wyniku. Do tego celu skorzystano z biblioteki GStreamer, dzięki której można tworzyć grafy z komponentów (pluginów, elementów) przetwarzających media, zarówno audio jak i video. Każdy z elementów grafu składa się z co najmniej jednego źródła (source), lub ujścia (sink), może mieć również wiele wejść i wyjść. W grafie pierwszy element nie może mieć wejść, natomiast konieczne jest aby posiadał co najmniej jedno wyjście. Poprawnie przygotowany graf nie powinien mieć komponentów oferujących źródło, które nie są z niczym połączone. Pluginy mają ujednolicony interfejs, dzięki czemu można w łatwy sposób włączyć do grafu własny element. Wtyczki charakteryzują się pewnymi własnościami, znanymi jako „caps”. Określają one jakie parametry muszą spełniać media, aby zostały przetworzone przez daną wtyczkę. Łączone ze sobą elementy dokonują negocjacji parametrów mediów, takich jak rozdzielczość obrazu, format pikseli, ilość klatek na sekundę oraz innych.

Wszystkie z elementów użytego grafu opiszę pokrótce:

Alfabetycznie czy zgodnie z kolejnością elementów w pipeline?

Graf
—
sche-
mat
nr 2

3.5.1 Caps filter

Element precyzujący parametry obrazu, które wymuszają dostosowanie się poprzedniego elementu — na przykład videoscale. Zapisuje się je w postaci ciągu znaków objętych w cudzysłowach.

3.5.2 fpsdisplaysink

Wtyczka typu sink (mająca tylko ujście), która jako parametr pobiera inną wtyczkę tego typu, np. xlnxvideosink. Jej użycie pozwala na sprawdzenie liczby klatek na sekundę wyświetlanego obrazu.

3.5.3 videoconvert

Element mający za zadanie dostosować wszystkie parametry obrazu tak, aby móc połączyć ze sobą dwa niekompatybilne pod względem „caps” elementy. Ta niekompatybilność może być spowodowana na przykład tym, że dwie wtyczki potrzebują innego formatu pikseli i jednocześnie nie oferują możliwości konwersji z jednego formatu na inny.

3.5.4 videocrop

Wtyczka służąca do wykadrowania obrazu w zdefiniowanym obszarze. Wykorzystana została aby otrzymać obraz o tej samej długości i szerokości wynoszącej 224 (co jest ośmiokrotnością 28, czyli długością boku obrazów, którymi wytrenowana została sieć) wycięty ze środka wideo o rozmiarze 1920×1080.

3.5.5 videoscale

Skaluje obraz do wynegocjowanych pomiędzy sąsiadującymi elementami parametrów, przy czym pierwsza próba negocjacji to ta sama wielkość obrazu przy ujściu jak i w źródle, aby skalowanie nie było potrzebne.

3.5.6 videobox

Oprócz możliwości videocrop oferuje także możliwość osadzenia przychodzącego wideo w tym o większym rozmiarze dodając dookoła ramkę w wybranym kolorze. Własność autocrop oznacza automatyczne obliczenie wielkości ramek na podstawie parametrów określonych przez kolejny element tak, aby obraz przychodzący do videobox był wycentrowany a ramki były tej samej wielkości.

3.5.7 xlnxvideosrc i xlnxvideosink

Są to pluginy dostarczone przez firmę Xilinx wraz z platformą reVISION. Obydwa korzystają biblioteki Xilinx `video_lib`. Pierwszy z nich ułatwia odczytywanie danych ze źródeł, dla których potrzebne byłyby dodatkowe działania. Są to między innymi kamera USB (użyta w projekcie), HDMI, MIPI CSI (sprzętowy interfejs do transmisji obrazów i wideo). Sam element zbudowany jest w oparciu o element `v4l2src`, dostępny w standardowej instalacji GStreamer. `Xlnxvideosink` również jest oparty o inny element — `kmssink`. Zapewnia odpowiednią konfigurację połączenia z wyświetlaczami podłączonymi przez HDMI oraz DisplayPort.

Przypis
UG1221,
s.32

3.6 Używanie sieci

3.6.1 Generowanie projektu

Architekturę sieci wraz z wagami zapisano do pliku `h5`. Stworzono plik konfiguracyjny `hls4ml`. Następnie na jego podstawie wygenerowano projekt z przekonwertowaną siecią. Wśród wygenerowanych plików znajduje się również kod służący do symulacji działania projektu. Przygotowane zostały pliki z danymi testującymi sieć — 10000 przetworzonych przykładów z bazy MNIST tak, aby cyfry były koloru czarnego, tło białego. Zakres wartości od 0 do 255.

3.6.2 Funkcja

Cała sieć jest jedną funkcją. Jej parametrami są dwie tablice: `input []`, w której są dane do przetworzenia, oraz `output []`, do której funkcja zapisuje obliczone predykcje.

Listing 1: Nagłówek funkcji

```
typedef unsigned char c_input_t;
typedef float         c_output_t;

void nn(
    c_input_t input1[N_INPUT_1_1],
    c_output_t output[N_LAYER_6]);
```

3.6.3 Dostosowanie sieci

W celu poprawienia wyników działania sieci dokonano pewnych usprawnień. Sieć została wytrenowana oryginalnymi danymi, w których dane przedstawiające cyfry mają wartości równe 255 lub bliskie, a białe tło jest przedstawione jako wartość 0. Ponadto rzeczywiste dane z kamery mogą być zaszumione, przedstawione obiekty zacienione, a same cyfry mogą nie być idealnie czarne.

Listing 2: kod

```
for(int i=0; i<N_INPUT_1_1; i++) {
    #pragma HLS pipeline II=1
    input1[i] = (input[i] < 140) ? 255 : 0;
}
```

gdzie `N_INPUT_1_1` to liczba pikseli, `input` to parametr funkcji

3.7 Część neuralnet

Czytanie obrazu, podział na część luma i chroma, wywołanie funkcji sieci, zapis z powrotem, synteza do biblioteki dzielonej „so”

3.8 Część gststdxnet

De facto plugin `gststreamera`, w którym są wywoływane funkcje z biblioteki dzielonej `neuralnet.so`,

3.9 Małe podsumowanie

4 Wyniki i dyskusja

4.1 Ewaulacja modelu

Wyniki z samego pythona z danymi testowymi z mnista

4.2 Symulacja

Tutaj wyniki z symulacji z danymi testowymi z mnista

4.3 Dane rzeczywiste

Wyniki z kamerki. Zdjęcia danych testowych, co wpływa na wynik, czy wszystko rozpoznaje itd,

5 Podsumowanie

W projekcie zostało zrobione to i to. Wyszło to tak i tak. Problem sprawiło tanto i owamto. Można to poprawić w ten sposób. Można część funkcjonalności z pipeline przenieść na fpga (w końcu przetwarzanie obrazu na fpga jest szybkie)