

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

<b>1</b>	<b>Wstęp</b>	<b>3</b>
<b>2</b>	<b>Teoria</b>	<b>4</b>
2.1	Architektura FPGA . . . . .	4
2.2	Przetwarzanie obrazu . . . . .	4
2.2.1	Formaty pikseli . . . . .	4
2.3	Sieci neuronowe . . . . .	5
<b>3</b>	<b>Opis projektu</b>	<b>6</b>
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 . . . . .	7
3.6	Używanie sieci . . . . .	7
3.6.1	Dostosowanie sieci . . . . .	7
3.7	Część neuralnet . . . . .	7
3.8	Część gstdxnnet . . . . .	7
3.9	Małe podsumowanie . . . . .	7
<b>4</b>	<b>Wyniki i dyskusja</b>	<b>8</b>
4.1	Ewaulacja modelu . . . . .	8
4.2	Symulacja . . . . .	8
4.3	Dane rzeczywiste . . . . .	8
<b>5</b>	<b>Podsumowanie</b>	<b>9</b>

# 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 kilkakrotnie 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.

Będzie obrazek ze schematem

## 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.

Schemat sieci

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.

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. Wtyczka jest następnie wykorzystana w zdefiniowanym potoku uruchomionym przy pomocy narzędzia gst-launch-1.0. Zadaniem spoczywającym na innych elementach potoku jest obsługa kamery, kadrowanie i skalowanie obrazu oraz wyświetlenie go na końcowym urządzeniu.

### 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 sche-

Tutaj  
sche-  
mat  
hls4ml  
work-  
flow

matu pokazuje ogólną organizację pracy przy projektowaniu odpowiedniego modelu 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.

### 3.4.2 Precyzja danych

Typ danych używany w przekonwertowanym modelu to duże liczby całkowite (`ap_int`) oraz liczby stałoprzecinkowe (`ap_fixed`), których precyzję można ustalić do jednego bita. Obliczenia przeprowadzane na liczbach o mniejszej precyzji są szybsze, natomiast zbyt niska precyzja może poskutkować bezużytecznością zsyntetyzowanej sieci. Aby odpowiednio dobrać precyzję danych skorzystano z narzędzi analizujących dostarczonych przez hls4ml. Narzędzie jest uruchamiane w Pythonie, gdzie należy dostarczyć plik z modelem Tensorflow, plik konfiguracyjny `hsl4ml` oraz dane.

Wynikiem są wykresy przedstawiające rozkład wartości wag każdej z warstw modelu. Szare pole w tle wykresu przedstawia zakres wartości, które obejmowane są przez precyzję określoną w pliku konfiguracyjnym. Dobrym punktem początkowym jest wybranie takiej precyzji dla każdej warstwy, która obejmuje wszystkie możliwe wagi. Dalsze ustalanie precyzji można wykonać w trakcie sprawdzania wyników symulacji.

Wstaw  
wy-  
kres z  
profi-  
ling

## 3.5 Gstreamer

Do czego służy, jaki zbudowałem pipeline

## 3.6 Używanie sieci

Czyli synteza sieci i zrobienie z niej biblioteki statycznej „a”

### 3.6.1 Dostosowanie sieci

W celu poprawienia wyników działania sieci dokonano pewnych usprawnień. Rzeczywiste dane z kamery mogą być zaszumione, obiekty zaciemnione. A biały nie jest białym.

## 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 gstdxnet

De facto plugin gstreamera, 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)