

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH

Phạm Duy Hoàng – 21522097
Lương Văn Đại -

ĐỒ ÁN 1
Tìm hiểu và hiện thực hóa mạng tích chập trên hệ
thống SoC cho bài toán phân loại tín hiệu ECG

GIẢNG VIÊN HƯỚNG DẪN
Trần Thị Diễm

TP. HỒ CHÍ MINH, 2024

MỤC LỤC

Chương 1. Giới thiệu đề tài.....	1
1.1. Mạng neural tích chập (CNN).....	1
1.1.1. Tổng quan về mạng neural tích chập.....	1
1.1.1.1. Mạng neural tích chập 1 chiều.....	1
1.1.2. Cấu trúc của một mạng neural tích chập 1D.....	1
1.1.2.1. Lớp tích chập (convolution layer).....	1
1.1.2.2. Lớp gộp (pooling layer).....	2
1.1.2.3. Lớp Fully-connected (dense).....	3
1.1.2.4. Hàm kích hoạt (activation).....	3
Chương 2. Triển khai phần mềm.....	5
2.1. Tổng quan về tập dữ liệu.....	5
2.1.1. Tín hiệu Electrocardiogram (ECG).....	5
2.1.2. Tập dữ liệu Mit-Bih.....	5
2.2. kiến trúc 1D-CNN sử dụng trong đồ án.....	6
2.2.1. Tổng quan[2].....	6
2.2.2. Huấn luyện.....	7
2.2.3. Kết quả phần mềm.....	8
2.2.4. Chuyển mô hình từ python sang C.....	9
2.2.4.1. Chuyển đổi checkpoint từ pytorch sang keras.....	9
2.2.4.2. Chuyển đổi từ keras sang C.....	9
Chương 3. Triển khai phần cứng.....	12
3.1. Chạy phần cứng trên vitis hls.....	12
3.2. Kết quả & báo cáo tài nguyên phần cứng.....	14
3.2.1. Chuyển mô hình từ python sang C.....	14
3.2.2. Nhận xét.....	15

DANH MỤC HÌNH VẼ

Hình 1.1: Ví dụ về một mạng neural tích chập.....	1
Hình 1.2: Ví dụ về lớp tích chập 1D[3].....	2
Hình 2.1: sóng ecg.....	5
Hình 2.3: Val los và train lost sau 100 epochs.....	8
Hình 2.4: mô hình sau khi chuyển sang C.....	10
Hình 2.5: Kết quả khi chạy floating point.....	10
Hình 2.6: Kết quả khi chạy 16bit fixed-point.....	11
Hình 3.1: Vitis hls.....	12
Hình 3.2: chọn top module là CNN trong CNN.cpp.....	13
Hình 3.3: Chạy c synthesis và kết quả.....	13
Hình 3.4: Kết quả mô phỏng dạng sóng với outmodel = 1 (trùng với label).....	14
Hình 3.5: Báo cáo chạy mô phỏng.....	14
Hình 3.6: Kết quả chạy RTL synthesis sau khi đóng gói.....	14

DANH MỤC BẢNG

Bảng 2.1: Số dữ liệu trong các tập train, test, validation.....	6
Bảng 2.2: Tổng quát kiến trúc.....	7
Bảng 2.2: Kết quả phần mềm.....	9
bảng 2.3: Kết quả khi test trên pytorch và keras.....	9
bảng 2.4: So sánh kết quả float-point và fixed-point.....	11
Bảng 3.1: Báo cáo tài nguyên phần cứng.....	15

DANH MỤC TỪ VIẾT TẮT

ACC: độ chính xác

SEN: độ nhạy,

SPEC: độ đặc hiệu

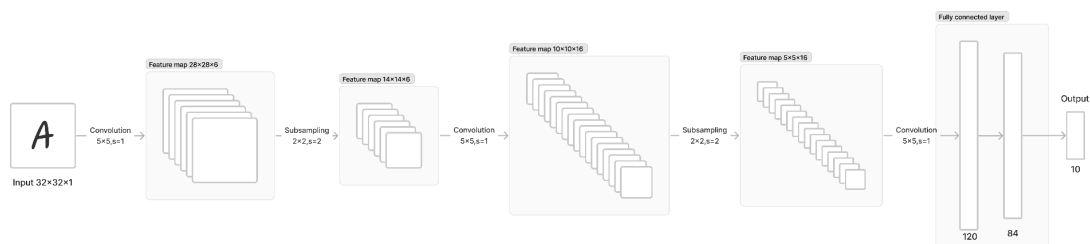
PPV: giá trị dự đoán tích cực

Chương 1. Giới thiệu đề tài

1.1. Mạng neural tích chập (CNN)

1.1.1. Tổng quan về mạng neural tích chập

- Mạng tích chập là mạng thần kinh nhân tạo được áp dụng trong lĩnh vực học sâu. Đây là một trong những mô hình học sâu cực kỳ tiên tiến cho phép xây dựng những hệ thống có độ chính xác cao và thông minh.
- CNN được ứng dụng rộng rãi vào nhiều lĩnh vực khác nhau, đặc biệt là trong các lĩnh vực phân loại và nhận diện hình ảnh.
- Hiện nay ngoài các kiến trúc CNN 2 chiều được sử dụng để phân loại, nhận diện hình ảnh thì còn có kiến trúc CNN 1 chiều được sử dụng để xử lý các tín hiệu như chuỗi thời gian thực hay các tín hiệu âm thanh. Kiến trúc CNN 3d được sử dụng để xử lý video và các mô hình không gian 3 chiều.
- Cấu trúc mạng CNN gồm 3 lớp chính: lớp tích chập, lớp pooling và fully connected (dense).



Hình 1.1: Ví dụ về một mạng neural tích chập

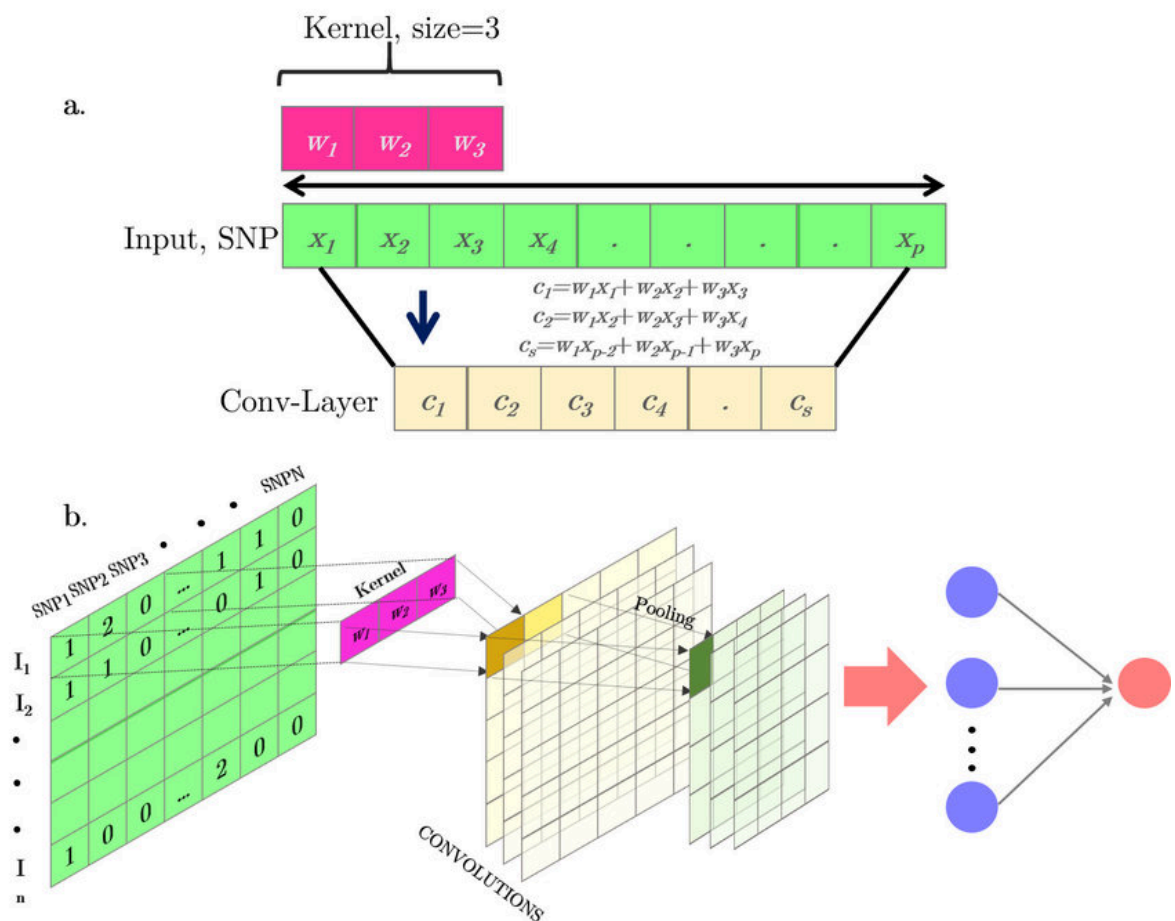
1.1.1.1. Mạng neural tích chập 1 chiều

- Trong đề án này sử dụng kiến trúc mạng neural tích chập 1 chiều để phân loại tín hiệu Electrocardiogram (ECG)
- Độ phức tạp tính toán của CNN 1D thấp hơn đáng kể so với CNN 2D.[\[1\]](#)
- Do yêu cầu tính toán thấp, CNN 1D nhỏ gọn rất phù hợp cho các ứng dụng thời gian thực như là tín hiệu ECG.[\[2\]](#)

1.1.2. Cấu trúc của một mạng neural tích chập 1D

1.1.2.1. Lớp tích chập (convolution layer)

- Mỗi nơ ron tích chập (filter) chỉ kết nối cục bộ với dữ liệu đầu vào.
- Nơ-ron tích chập sẽ trượt từ trái sang phải với bước nhảy (stride). Giá trị của output là tính tổng tất cả các phần tử trên nơ ron tích chập nhân với các phần tử trên input.



Hình 1.2: Ví dụ về lớp tích chập 1D^[3]

- Padding (Bộ đệm): Phần đệm giúp duy trì kích thước của đầu vào khi áp dụng lớp chập, giảm mất thông tin bằng cách thêm các giá trị 0 ở rìa ảnh
- Valid padding: Trường hợp này không áp dụng padding. Trong trường hợp này, phép tích chập cuối cùng sẽ bị loại bỏ nếu các kích thước không khớp nhau.
- Same padding: padding này đảm bảo rằng lớp đầu ra có kích thước giống với lớp đầu vào.

1.1.2.2. Lớp gộp (pooling layer)

- Lớp này được sử dụng để giảm kích thước của feature map và được áp dụng sau lớp tích chập.
- Hoạt động độc lập trên từng bản đồ kích hoạt
- Max pooling: Từng phép pooling chọn giá trị lớn nhất.
- Average pooling: Từng phép pooling lấy trung bình cộng các giá trị.
- Sum Pooling: Từng phép pooling lấy giá trị tổng của các giá trị

1.1.2.3. Lớp Fully-connected (dense)

- Tầng Fully-connected (dense) nhận đầu vào là các dữ liệu đã được làm phẳng, mà mỗi đầu vào đó được kết nối đến tất cả neuron. Trong mô hình mạng CNNs, các tầng kết nối đầy đủ thường được tìm thấy ở cuối mạng và được dùng để tối ưu hóa mục tiêu của mạng ví dụ như độ chính xác của lớp.

1.1.2.4. Hàm kích hoạt (activation)

- Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$
 - Nhận giá trị trong khoảng $[0;1]$
 - Được dùng phổ biến trong lịch sử mạng nơron do chúng mô phỏng tốt tỉ lệ bắn xung (firing rate) của nơ-ron
 - Có 3 nhược điểm:
 - Nơ-ron bão hòa, triệt tiêu Gradient
 - Trung bình đầu ra khác 0
 - Tính toán hàm mũ $\exp()$ tốn kém

$$\frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Tanh:
 - Nhận giá trị trong khoảng $[-1;1]$
 - Hàm Tanh cũng bị bão hoà, triệt tiêu Gradient giống hàm tanh tuy nhiên trung bình đầu ra bằng 0

$$f(x) = \max(0, x)$$

- ReLU:
 - Không bị bão hoà trong vùng dương
 - Tính toán nhanh gấp khoảng 6 lần tanh/sigmoid
 - Nhược điểm: Với các node có giá trị nhỏ hơn 0, qua ReLU activation sẽ thành 0, hiện tượng này gọi là “Dying ReLU”.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

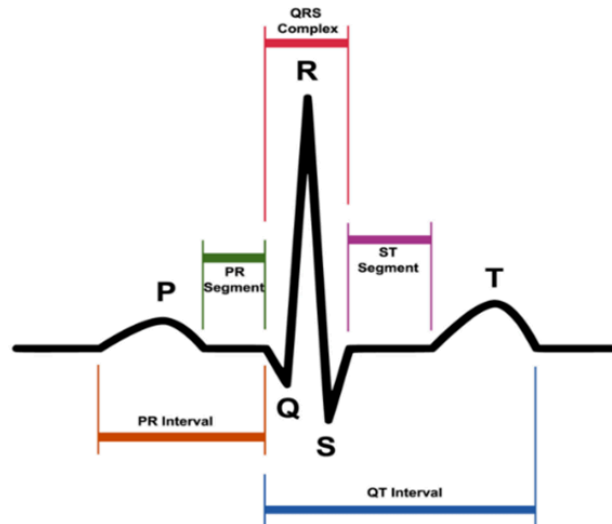
- Softmax:
 - Trong các bài toán phân loại hình ảnh, lớp cuối cùng của CNN thường sử dụng hàm softmax để đưa ra xác suất thuộc về các lớp khác nhau.
 - Xác suất sẽ luôn nằm trong khoảng $(0;1]$.
 - Tổng tất cả các xác suất bằng 1.

Chương 2. Triển khai phần mềm

2.1. Tổng quan về tập dữ liệu

2.1.1. Tín hiệu Electrocardiogram (ECG)

- ECG hay còn gọi là điện tâm đồ, nó ghi lại tất cả các tín hiệu điện ở trong tim
- ECG cung cấp thông tin để chẩn đoán và theo dõi một số bệnh lý về tim



Hình 2.1: sóng ecg

2.1.2. Tập dữ liệu Mit-Bih

- Cơ sở dữ liệu MIT-BIH chứa 48 đoạn trích nửa giờ của bản ghi ECG hai kênh cấp cứu, thu được từ 47 đối tượng được Phòng thí nghiệm của bệnh viện Beth Israel nghiên cứu từ năm 1975 đến năm 1979.
- Tất cả 48 bản ghi ECG thu được từ 47 đối tượng đều được lấy mẫu ở tần số 360Hz và kéo dài hơn 30 phút và lấy từ PhysioNet.
- Tập dữ liệu chỉ có 2 và hầu hết được ghi lại ở kênh MLII
- Ở đồ án này sử dụng dữ liệu từ kênh MLII ngoại trừ 2 bản ghi số 102 và 104 lấy từ kênh V2. Sóng được phân loại vào 5 nhãn: N(Normal), A(APC), V(PVC), L(LBBB), R(RBBB).
- Tách chú thích và thời gian của từng nhãn vào một tập dataset sau đó chia ra thành các tập riêng để huấn luyện, kiểm tra và validation với tỉ lệ: tập huấn luyện 70%, tập kiểm tra 15% và tập validation 15%

Class	Train	Val	Test	Total
N	52535	11258	11257	75050

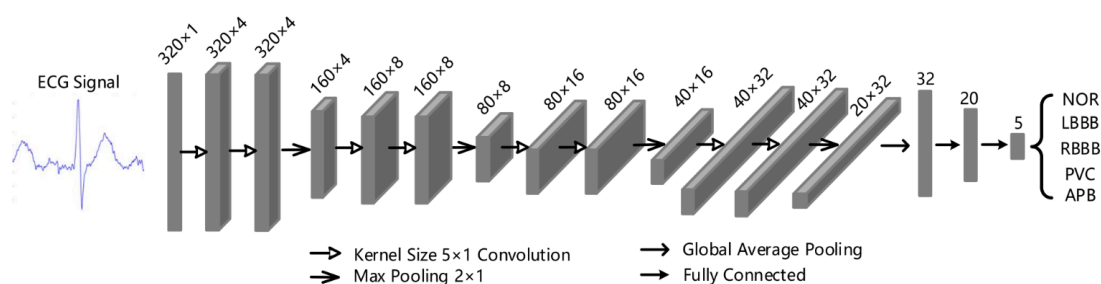
L	5653	1211	1211	8075
R	5081	1089	1089	7259
V	4991	1069	1070	7130
A	1782	382	382	2546
Total	70042	15009	15009	100060

Bảng 2.1: Số dữ liệu trong các tập train, test, validation

2.2. kiến trúc 1D-CNN sử dụng trong đề án

2.2.1. Tổng quan^[2]

- Sử dụng kiến trúc 1D-CNN, bao gồm tám lớp tích chập, bốn lớp max pooling, một lớp GAP và hai lớp Fully-connected.
- Input là tín hiệu ECG có kích thước 320×1



Hình 2.2: tổng quan kiến trúc

Layer	Size	Kernel	Kernel size	Stride	Padding	Activation	Parameter
Input	320	1	-	-	-	-	-
Conv1d	320	4	5	1	Same	Relu	24
Conv1d	320	4	5	1	Same	Relu	84
Maxpool	160	4	2	1	-	-	-
Conv1d	160	8	5	1	Same	Relu	168
Conv1d	160	8	5	1	Same	Relu	328
Maxpool	80	8	2	1	-	-	-
Conv1d	80	16	5	1	Same	Relu	656
Conv1d	80	16	5	1	Same	Relu	1296
Maxpool	40	16	2	1	-	-	-
Conv1d	40	32	5	1	Same	Relu	2592
Conv1d	40	32	5	1	Same	Relu	5152
Maxpool	20	32	2	1	-	-	-
GAP	32	-	-	-	-	-	-
Fully connected	32	20	-	-	-	Relu	660
Fully connected	20	5	-	-	-	-	105
Total parameter	-	-	-	-	-	-	11065

Bảng 2.2: Tổng quát kiến trúc

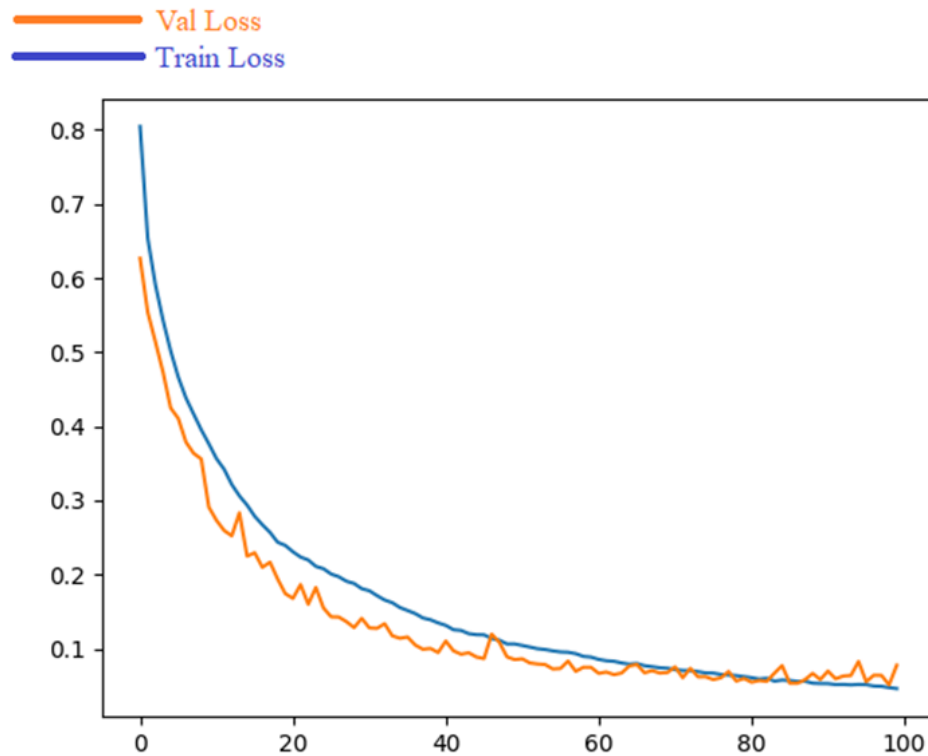
- Ngoài các lớp như mạng CNN thông thường, trong kiến trúc này, lớp Global average pooling sẽ được đặt sau lớp max pooling cuối cùng thay thế cho lớp flatten.
- Global Average Pooling được sử dụng để giảm chiều không gian của các đặc trưng đầu vào cho các lớp FC và giảm ảnh hưởng của các lớp FC lên hiệu suất tổng thể.

2.2.2. Huấn luyện

- Trong quá trình huấn luyện, trọng số của mạng CNN 1-D được khởi tạo bằng thuật toán He_normal. Thuật toán này được thiết kế để cải thiện việc khởi tạo trọng số cho các mạng nơ-ron sâu, đặc biệt là những mạng sử dụng hàm kích hoạt ReLU. Sử dụng thuật toán tối ưu hóa Adam trong quá trình lan truyền ngược (back-propagation) để tăng tốc quá trình huấn luyện.
- Batch size của tập huấn luyện và validation được đặt lần lượt là 20 và 5.
- Learning rate trước 40 epochs là 0.0001 trong khi đó sau 40 epoch sẽ đặt learning rate là 0.00001 và được huấn luyện trong 100 epochs
- Các chỉ số được sử dụng để đánh giá hiệu suất của mô hình 1-D CNN đề xuất là độ chính xác (ACC), độ nhạy (SEN), độ đặc hiệu (SPEC) và giá trị dự đoán tích cực (PPV), được tính toán dựa trên ma trận nhầm lẫn chuẩn hóa (Confusion Matrix) sử dụng các giá trị true positive (TP), true negative (TN), false positive (FP) và false negative (FN). Công thức của các chỉ số là như sau:

$$\bullet \text{ ACC} = \frac{TP + TN}{TP + FP + TN + FN}$$

- $SEN = \frac{TP}{TP + FN}$
- $SPEC = \frac{TN}{FP + TN}$
- $PPV = \frac{TP}{TP + FP}$



Hình 2.3: Val los và train lost sau 100 epochs

2.2.3. Kết quả phần mềm

	[2]	Training trên máy
Tập dữ liệu	Mit-Bih	Mit-Bih
ACC	99.10	98.70
SEN	99.13	95.49
SPEC	98.59	96.3
PPV	99.10	92.5

Bảng 2.2: Kết quả phần mềm

2.2.4. Chuyển mô hình từ python sang C

2.2.4.1. Chuyển đổi checkpoint từ pytorch sang keras

- Tạo mô hình CNN tương tự trên Keras
- Tiến hành chuyển các trọng số weight, bias từ state_dict của file 'max_checkpoint.pth' ở các lớp Conv1d và Linear sang các lớp Conv1D và Dense tương ứng ở model Keras (lớp Linear ở Pytorch chính là lớp Dense ở Keras).
- Lưu ý rằng ở lớp Conv1d ở Pytorch có kích thước là ['out_channels', 'in_channels', 'kernel_size'] còn ở Keras sẽ là ['kernel_size', 'in_channels', 'out_channels'] nên trước khi sử dụng phương thức 'set_weights' cho lớp Conv1D của Keras phải thay đổi thứ tự các trục theo thứ tự (2,1,0) bằng phương thức 'transpose'.
- Tương tự với weight ở lớp Linear có thứ tự ['out_features', 'in_features'], ở lớp Dense là ['in_features', 'out_features'] nên cần hoán vị trước khi sử dụng phương thức 'set_weights'
- Lưu mô hình sau đó tiến hành test với bộ dữ liệu 'Test' đã được chia từ MIT-BIH Dataset

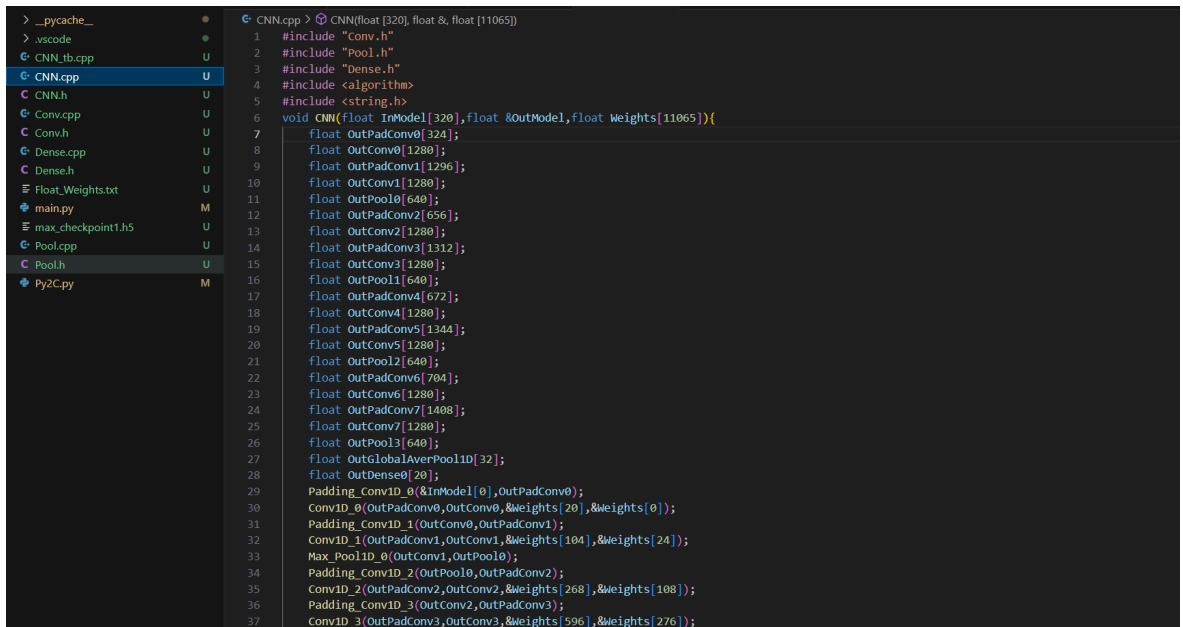
➤ Kết quả:

	Pytorch	Keras
Tập dữ liệu	Mit-Bih	Mit-Bih
ACC	98.70	94.15
SEN	95.49	98.95
SPEC	96.3	98.7
PPV	92.5	94.8
F1	93.98	96.84

bảng 2.3: Kết quả khi test trên pytorch và keras

2.2.4.2. Chuyển đổi từ keras sang C

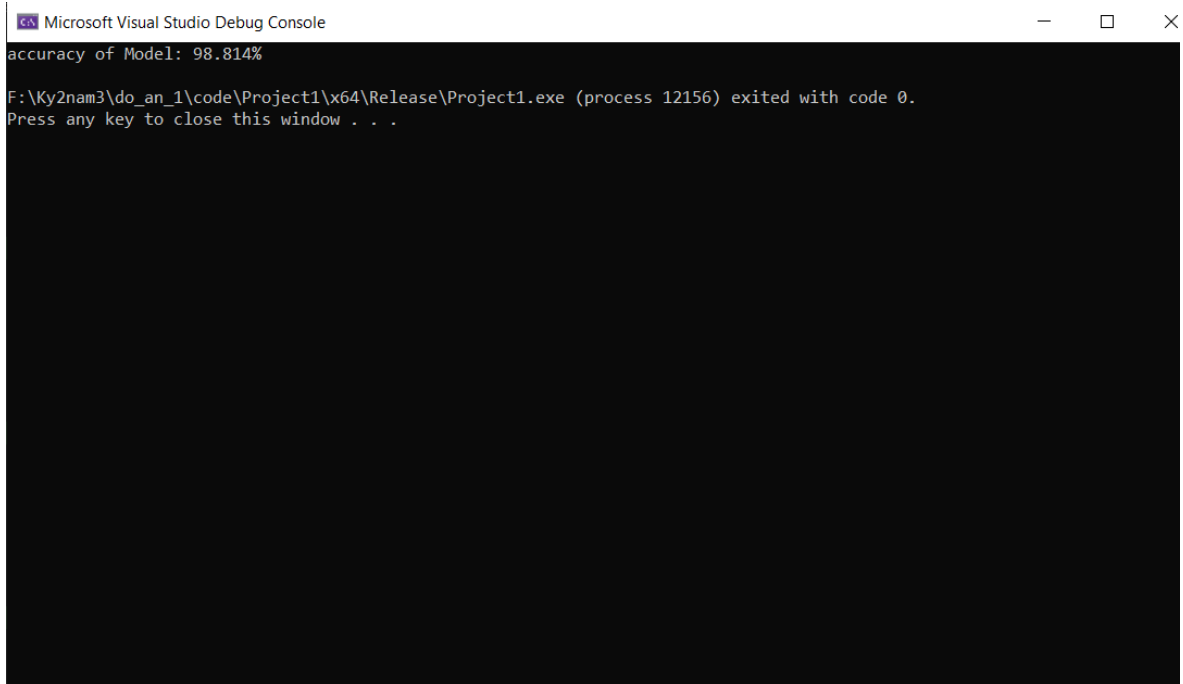
- Sau khi lấy được file checkpoint h5, đưa vào thư mục PytoC để chuyển đổi sang code c.
- Đầu tiên chuyển đổi sang floating point trước và tiến hành test trên visual studio. Sau khi kết quả chính xác thì ta sẽ chạy trên fixed-point 16 bit trong vitis hls



```
> _pycache_
> .vscode
CNN_tb.cpp
CNN.cpp
CNN.h
Conv.cpp
Conv.h
Dense.cpp
Dense.h
Float_Weights.txt
main.py
max_checkpoint1.hs
Pool.cpp
Pool.h
Py2C.py

CNN.cpp > CNN(float [320], float &, float [11065])
1 #include "Conv.h"
2 #include "Pool.h"
3 #include "Dense.h"
4 #include <algorithm>
5 #include <string.h>
6 void CNN(float InModel[320], float &OutModel, float Weights[11065]){
7     float OutPadConv0[324];
8     float OutConv0[1280];
9     float OutPadConv1[1296];
10    float OutConv1[1280];
11    float OutPool0[640];
12    float OutPadConv2[656];
13    float OutConv2[1280];
14    float OutPadConv3[1312];
15    float OutConv3[1280];
16    float OutPool1[640];
17    float OutPadConv4[672];
18    float OutConv4[1280];
19    float OutPadConv5[1344];
20    float OutConv5[1280];
21    float OutPool2[640];
22    float OutPadConv6[704];
23    float OutConv6[1280];
24    float OutPadConv7[1408];
25    float OutConv7[1280];
26    float OutPool3[640];
27    float OutGlobalAverPool1D[32];
28    float OutDense0[20];
29    Padding_Conv1D_0(&InModel[0], OutPadConv0);
30    Conv1D_0(OutPadConv0, OutConv0, &Weights[20], &Weights[0]);
31    Padding_Conv1D_1(OutConv0, OutPadConv1);
32    Conv1D_1(OutPadConv1, OutConv1, &Weights[104], &Weights[24]);
33    Max_Pool1D_0(OutConv1, OutPool0);
34    Padding_Conv1D_2(OutPool0, OutPadConv2);
35    Conv1D_2(OutPadConv2, OutConv2, &Weights[268], &Weights[108]);
36    Padding_Conv1D_3(OutConv2, OutPadConv3);
37    Conv1D_3(OutPadConv3, OutConv3, &Weights[596], &Weights[276]);
```

Hình 2.4: mô hình sau khi chuyển sang C



```
Microsoft Visual Studio Debug Console
accuracy of Model: 98.814%

F:\Ky2nam3\do_an_1\code\Project1\x64\Release\Project1.exe (process 12156) exited with code 0.
Press any key to close this window . . .
```

Hình 2.5: Kết quả khi chạy floating point


```
1|INFO: [SIM 2] ***** CSIM start *****
2|INFO: [SIM 4] CSIM will launch GCC as the compiler.
3|  Compiling ../../../../CNN_tb.cpp in debug mode
4|  Compiling ../../../../CNN.cpp in debug mode
5|  Compiling ../../../../Conv.cpp in debug mode
6|  Compiling ../../../../Dense.cpp in debug mode
7|  Compiling ../../../../Pool.cpp in debug mode
8|  Generating csim.exe
9|accuracy of Model: 95.5893%
10|INFO: [SIM 1] CSim done with 0 errors.
11|INFO: [SIM 3] ***** CSIM finish *****
12|
```

Hình 2.6: Kết quả khi chạy 16bit fixed-point

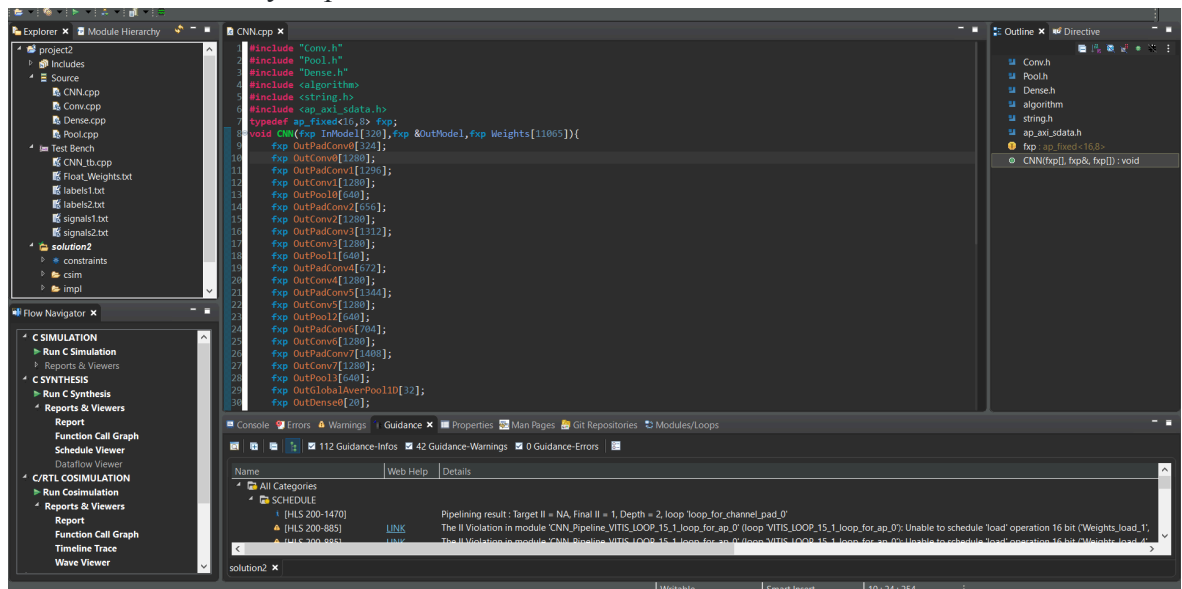
	Float-point	Fixed point
độ dài	32bit	16bit
ACC	98.14	95.58

bảng 2.4: So sánh kết quả float-point và fixed-point

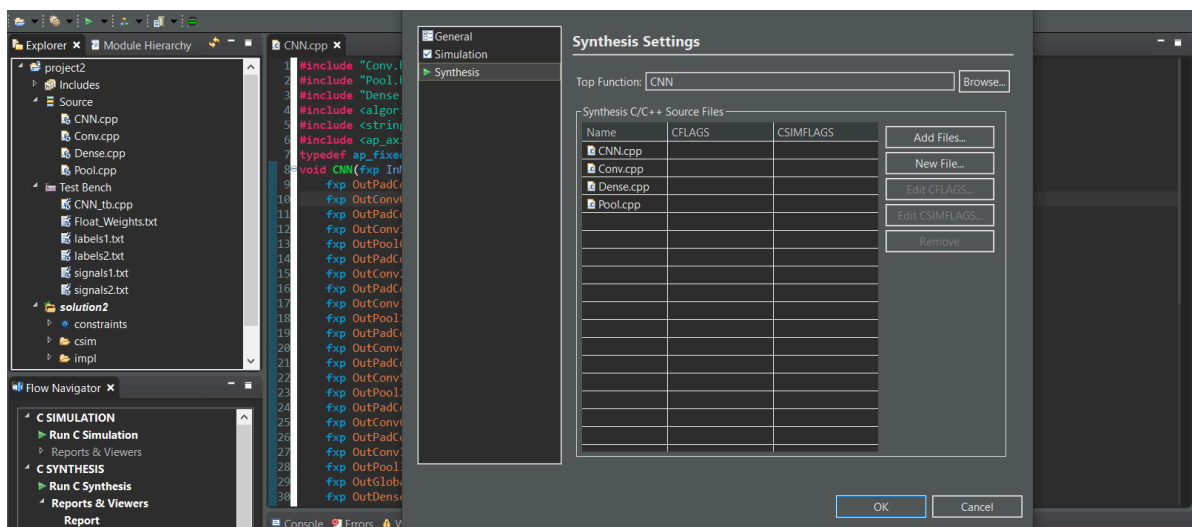
Chương 3. Triển khai phần cứng

3.1. Chạy phần cứng trên vitis hls

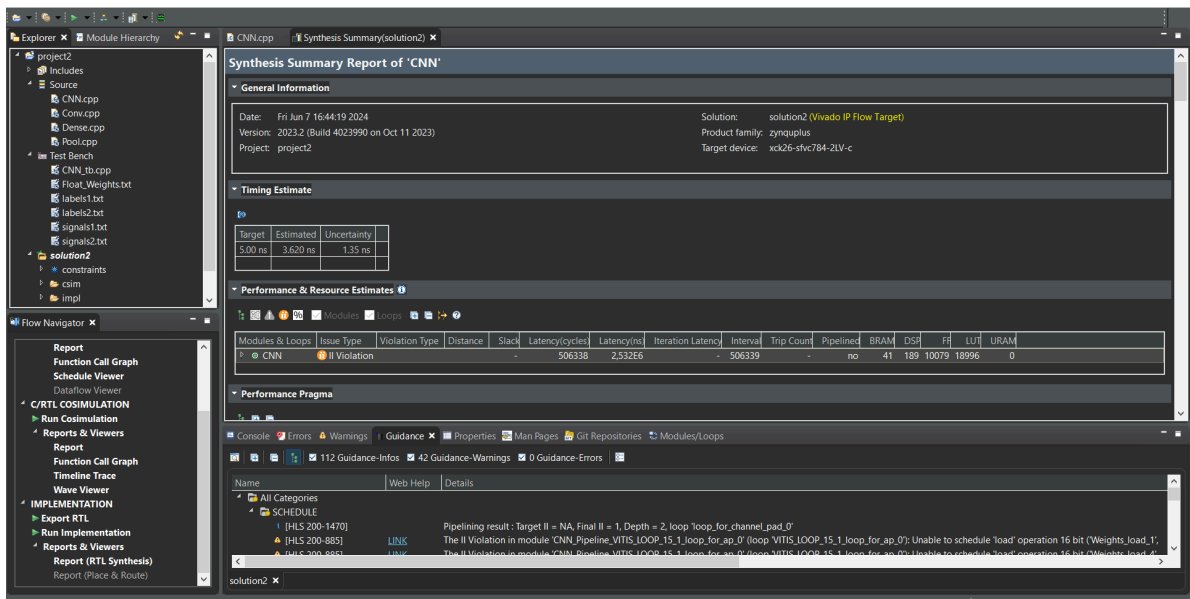
- Tiến hành chạy synthesis và implement trên phần mềm vitis hls với fixed-point 16 bit và clock 200 MHz
- Các file cpp được thêm vào phần source trong khi file test bench và file txt được thêm vào phần test bench và chọn top function là CNN trong file CNN.cpp
- Sau đó tiến hành synthesis, mô phỏng dạng sóng, export RTL để đóng gói ip và chạy implement



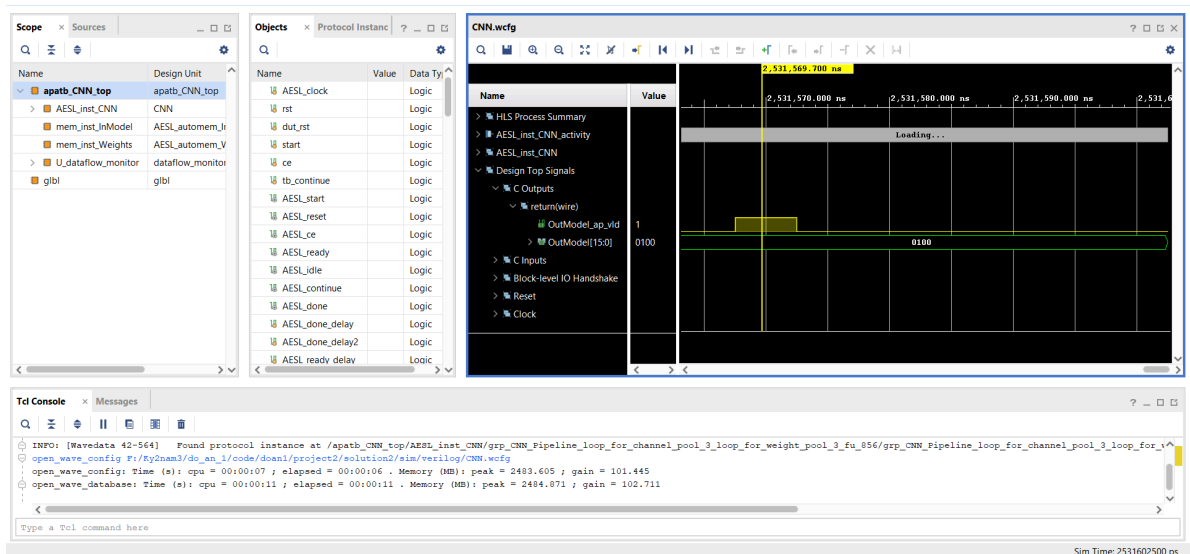
Hình 3.1: Vitis hls



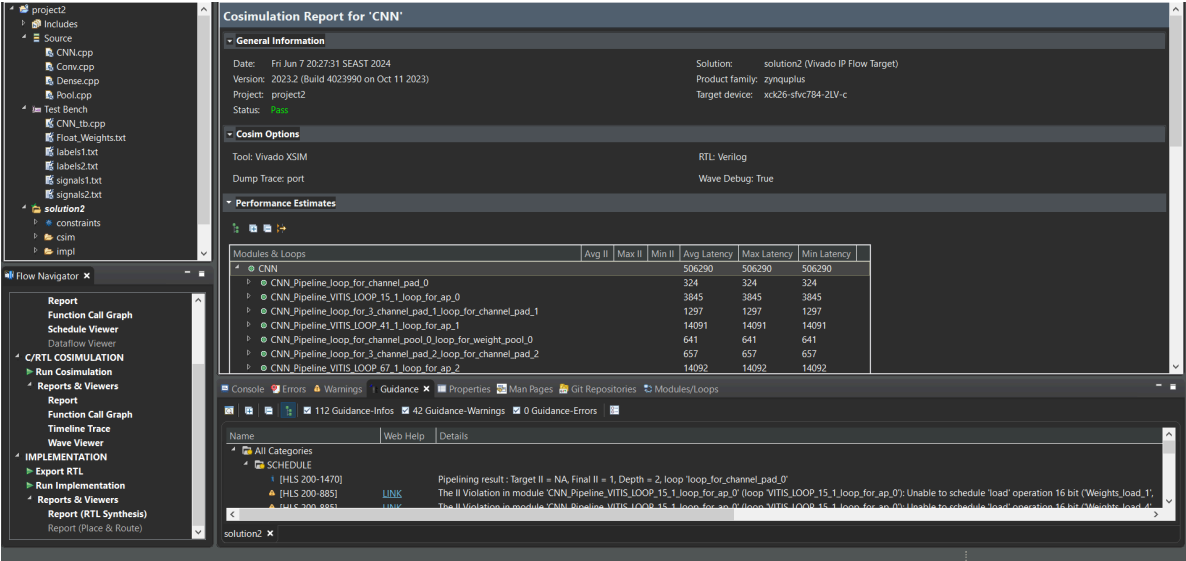
Hình 3.2: chọn top module là CNN trong CNN.cpp



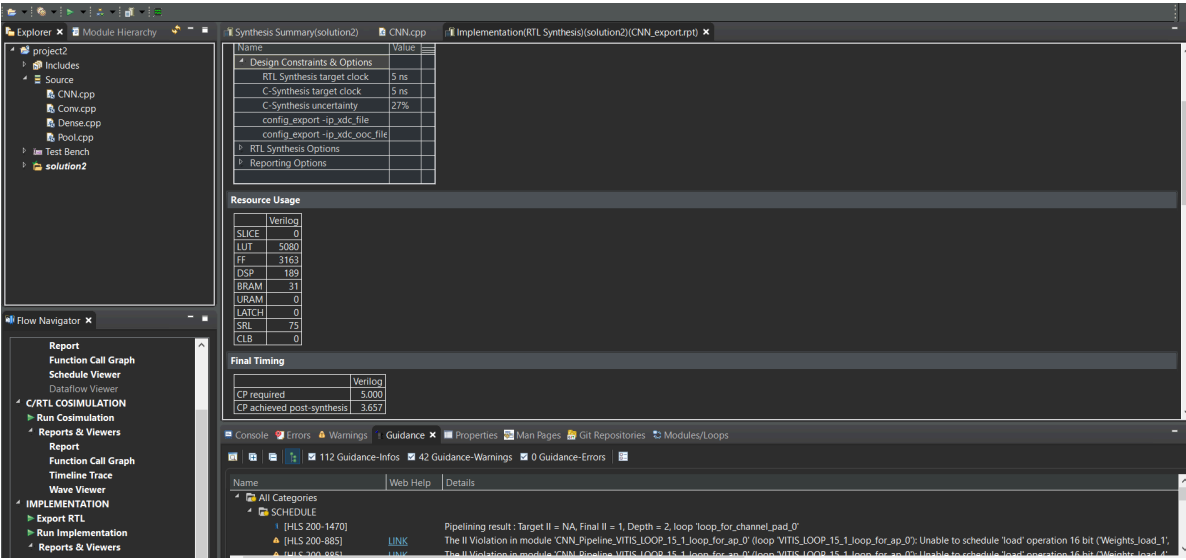
Hình 3.3: Chạy c synthesis và kết quả



Hình 3.4: Kết quả mô phỏng dạng sóng với outmodel = 1 (trùng với label)



Hình 3.5: Báo cáo chạy mô phỏng



Hình 3.6: Kết quả chạy RTL synthesis sau khi đóng gói

3.2. Kết quả & báo cáo tài nguyên phần cứng

3.2.1. Chuyển mô hình từ python sang C

	[2]	Floating	Fixed point	Fixed point
--	-----	----------	-------------	-------------

		point		
CNN model	1D CNN	1D CNN	1D CNN	1D CNN
FPGA	Zynq XC7Z020	KV260 XCK26-SFV C784-2LV-C	KV260 XCK26-SFV C78-2LV-C	KV260 XCK26-SFV C78-2LV-C
Parameter	11065	11065	11065	11065
Precision	16bit fixed	32 bit	16bit fixed	32bit fixed
DSP utilization	80	34	189	80
Resource utilization (kLUT)	1.538	22.420	4.913	14.128
Bram utilization	12	80	31	74
Fmax(MHz)	-	246.63	276.24	274.35
latency(ns)	-	1.739E7	2.532E6	2.762E6
FF	-	29422	3163	9579

Bảng 3.1: Báo cáo tài nguyên phần cứng

3.2.2. Nhận xét

- Khi chạy trên floating point tốn nhiều tài nguyên phần cứng hơn so với trên fixed point

TÀI LIỆU THAM KHẢO

- [1] KIRANYAZ, Serkan, et al. *1D convolutional neural networks and applications: A survey. Mechanical systems and signal processing*, 2021, 151: 107398.
- [2] LU, Jiahao, et al. *Efficient hardware architecture of convolutional neural network for ECG classification in wearable healthcare device. IEEE Transactions on Circuits and Systems I: Regular Papers*, 2021, 68.7: 2976-2985.
- [3] PÉREZ-ENCISO, M.; ZINGARETTI, L. M. *A guide for using deep learning for complex trait genomic prediction. Genes (Basel)* 10: 553. 2019.