ỨNG DỤNG NEURAL NETWORKS VÀO GIẢI ĐIỀU CHẾ TÍN HIỆU TRONG HỆ THỐNG VLC

LUẬN VĂN TỐT NGHIỆP

Nguyễn Trí Hiếu – 1711298
Giảng viên hướng dẫn
TS. Phạm Quang Thái

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ, BỘ MÔN VIỄN THÔNG

8 – 2021

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh Phúc

Số:_____/BKĐT
Khoa: **Điện – Điện tử**
Bộ môn: **Viễn thông**

NHIỆM VỤ LUẬN VĂN TỐT NGHIỆP

1. Họ và tên: Nguyễn Trí Hiếu, MSSV: 1711298
2. Ngành: Điện – Điện tử, Chuyên ngành: Kỹ thuật Điện tử – Truyền thông
3. Đề tài: Ứng dụng neural networks vào giải điều chế tín hiệu trong hệ thống VLC
4. Nhiệm vụ:
   - Đo đạc trên hệ thống VLC tại phòng thí nghiệm 209B1
   - Giải điều chế tín hiệu sử dụng neural networks
5. Ngày giao nhiệm vụ: 22-1-2020
6. Ngày hoàn thành nhiệm vụ: 1-7-2021
7. Họ và tên người hướng dẫn: Phần hướng dẫn:
   TS. Phạm Quang Thái 100%
   BM Viễn Thông, Khoa Điện – Điện Tử

Nội dung và yêu cầu LVTN đã được thông qua Bộ Môn.

Tp. HCM, Ngày __ tháng __ năm 20__
CHỦ NHIỆM BỘ MÔN

PGS. TS. Hà Hoàng Kha

Tp. HCM, Ngày __ tháng __ năm 20__
NGƯỜI HƯỚNG DẪN CHÍNH

TS. Phạm Quang Thái

PHẦN DÀNH CHO KHOA, BỘ MÔN:
Người duyệt (chấm sơ bộ): _____
Đơn vị:_____
Ngày bảo vệ:_____
Điểm tổng kết:_____
Nơi lưu trữ luận văn:_____

LỜI CÁM ƠN

Để hoàn thành báo cáo này, em xin chân thành gửi lời cảm ơn đến:

Đầu tiên và sâu sắc nhất là thầy TS. Phạm Quang Thái, người đã giúp đỡ em xuyên suốt trong quá trình hoàn thành luận văn tốt nghiệp. Từ những ngày đầu tiên, khi em gặp khó khăn, thầy đã sẵn sàng góp ý, tận tình giúp đỡ, chỉ ra những điểm thiếu sót, gợi ý phương pháp làm bài, nguồn tài liệu, ... Đó là những đóng góp quý báu giúp em có thể hoàn thành báo cáo.

Kế đến cũng không thể không nhắc đến công lao của quý thầy cô khoa Điện-điện tử, đại học Bách Khoa thành phố Hồ Chí Minh, những người đã chỉ dạy em trong những năm vừa qua.

Cuối cùng, xin gửi lời cảm ơn đến các bạn, các anh cùng học tập, nghiên cứu tại phòng thí nghiệm 209B1.

Xin chúc những điều tốt đẹp nhất sẽ luôn đồng hành cùng quý thầy cô và các bạn.

Tp. HCM, Ngày 10 tháng 8 năm 2021

Nguyễn Trí Hiếu

LỜI CAM ĐOAN

Tôi tên: Nguyễn Trí Hiếu (MSSV: 1711298), là sinh viên chuyên ngành Kỹ thuật Điện tử - Truyền thông, tại Trường Đại học Bách Khoa, Đại học Quốc gia thành phố Hồ Chí Minh. Tôi xin cam đoan những nội dung sau đều là sự thật:

- Công trình nghiên cứu này hoàn toàn do chính tôi thực hiện;
- Các tài liệu và trích dẫn trong báo cáo này được tham khảo từ các nguồn thực tế, có uy tín và độ chính xác cao;
- Các số liệu và kết quả của công trình này được tôi tự thực hiện một cách độc lập và trung thực.

Tp. HCM, Ngày 10 tháng 8 năm 2021

Nguyễn Trí Hiếu

TÓM TẮT NỘI DUNG

Trong luận văn này bao gồm hai phần chính là đo đạc tại phòng thí nghiệm 209B1 (Hcmut) và xây dựng những mô hình học máy khác nhau để giải điều chế tín hiệu. Dữ liệu từ việc đo đạc tín hiệu hai mức xung Lorentz trên hệ thống truyền thông bằng ánh sáng khả kiến dùng Organic Light Emitting Diode (OLED) được sử dụng làm tập dữ liệu cho mô hình máy học. Luận văn này sẽ trình chi tiết các quá trình từ đo đạt dữ liệu, xử lý tín hiệu đến phân loại tín hiệu.

Các phương pháp xử lý tín hiệu sẽ khảo sát bao gồm: giảm méo dạng phi tuyến bằng Denoising Autoencoder Neural Networks. Trong phương pháp này, dữ liệu để huấn luyện là tín hiệu hai mức ngẫu nhiên được cộng thêm nhiễu Gaussian. Mục tiêu của mô hình là học được những đặc tính của tín hiệu hai mức, để có thể hiệu chỉnh lại tín hiệu bị méo dạng khi đi qua kênh truyền Visible Light Communication (VLC). Phương pháp thứ hai được khảo sát là trích xuất tín hiệu thành từng frame, mỗi frame sẽ bao gồm bit cần phân loại cùng với hai bits liền trước và hai bits liền sau nó. Ngoài ra, một kỹ thuật phân tích tín hiệu trong cả miền thời gian và tần số là Continuous wavelet transform (CWT) cũng được sử dụng để cải thiện Bit Error Rate (BER).

Để có thể phân loại tín hiệu với hai mức khác nhau, chúng tôi sẽ sử dụng ba phương pháp là Probabilistic Neural Networks (PNN), General Regression Neural Networks (GRNN) và Deep Tensor Neural Networks (DTNN), sau đó sẽ so sánh ba phương pháp với nhau và với phương pháp thông thường là bộ giải điều chế dùng ngưỡng.

Mục tiêu của luận văn này là đi tìm phương pháp tiền xử lý cũng như phân loại tín hiệu tốt nhất để có thể làm giảm BER xuống mức dưới ngưỡng cho phép trong hệ thống VLC là 3.8e - 3, với tốc độ bit và khoảng cách cao nhất có thể.

ABSTRACT

In this thesis, the two main parts are: measuring at laboratory 209B1 (Hcmut) and building different machine learning models for signal demodulation. Data from the measurement of two Lorentz pulses signals on visible light communication systems using OLED use as datasets for the machine learning model. This thesis will detail the processes from data acquisition, signal processing to signal classification.

The signal processing steps to be investigated include: reducing nonlinear distortion with Denoising Autoencoder Neural Networks. The data to train is a two-level randomized pulse with added Gaussian noise. The goal of the model is to learn the characteristics of a two-level pulse to correct the distorted signal when passing through the VLC channel. The second way to be investigated is to extract the signal into frames. Each frame contains the classified bit along with the two preceding and two bits following it. In addition, a time domain and frequency domain CWT technique is also used to improve BER.

To be able to classify signals with two different levels, we will use three methods: PNN, GRNN and DTNN, then compare the three methods with each other and with the demodulator using a threshold.

The goal of this thesis is to find the best method of preprocessing as well as classifying signals that can reduce BER to a level below the allowable threshold in the VLC system of 3.8e - 3, with the highest possible bit rate and distance.

MỤC LỤC

DANH SÁCH HÌNH VẼ

DANH SÁCH BẢNG

DANH SÁCH TỪ VIẾT TẮT

AE Autoencoder

BER Bit Error Rate

CNN Convolutional Neural Networks

CSK Color Shift Keying

CWT Continuous wavelet transform

DAE Denoising Autoencoder

DNN Deep Neural Networks

DSP Digital Signal Processing

DTNN Deep Tensor Neural Networks

GAN Generative adversarial networks

GRNN General Regression Neural Networks

ISI Intersymbol Interference

LED Light Emitting Diode

LOS Light of sight

LSTM Long Short Term Memory

M2M Machine to Machine

MSE Mean Squared Error

NLOS Non light of sight

NRZ Non return to zero

OFDM Orthogonal Frequency Division Multiplexing

OLED Organic Light Emitting Diode

OOK On Off Keying

PDF hàm mật độ xác suất

PLED Polymer Light Emitting Diode

PNN Probabilistic Neural Networks

RF Radio Frequency

SNR Signal to Noise Ratio

SSL Solid State Lighting

VLC Visible Light Communication

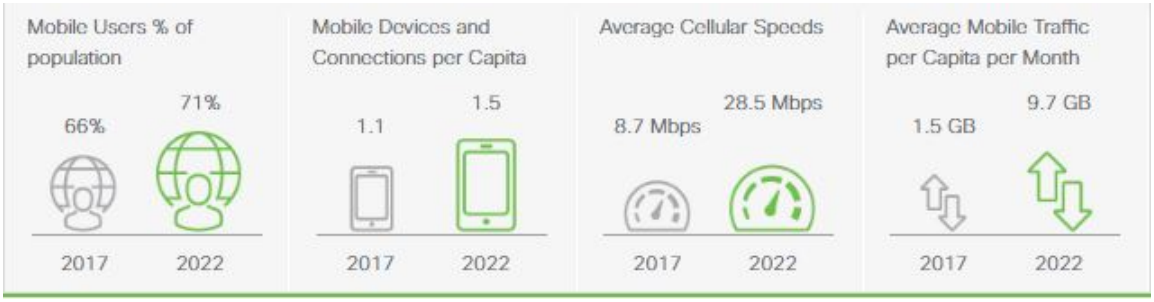
VPPM Variable Pulse Position Modulation

WLAN Wireless local area networks

Chương 1. GIỚI THIỆU

1.1 Đặt vấn đề

Sự giới hạn của phổ tần số là hạn chế của cho việc tăng độ phổ biến và dung lượng kênh truyền. Theo CISCO [6], số lượng thiết bị kết nối vào mạng IP sẽ nhiều hơn gấp 3 lần dân số thế giới vào năm 2023. Cụ thể, sẽ có trung bình 3.6 thiết bị trên một người năm 2023, so với 2.4 thiết bị trên một người 2018. Kết nối Machine to Machine (M2M) sẽ chiếm một nửa các kết nối trên toàn cầu vào năm 2023 (tăng từ 33 % năm 2018 lên 50 % vào năm 2023), và sẽ có 14.7 tỷ kết nối M2M vào năm 2023. Hơn 70 % dân số thế giới có thiết bị kết nối di động năm 2023. Lưu lượng di động trung bình của một người trên một tháng tăng gấp 6 lần từ năm 2017 đến năm 2022. Sự tăng số lượng các thiết bị và nhu cầu sử dụng là nguyên nhân chính cho việc tăng lưu lượng dữ lượng di động. Mặc dù các



Hình 1.1: Dự đoán của Cisco về dữ liệu di động [4]

nhà cung cấp dịch vụ không dây đang triển khai thêm cơ sở hạ tầng bằng cách bổ sung những trạm phát Wi-fi, nhưng hạn chế là việc sử dụng quá mức phổ Radio Frequency (RF) hiện có. Từ đó gây ra can nhiễu và chồng lấn phổ làm tăng độ trễ và giảm dung lượng. Để khắc phục tình trạng trên, một công nghệ mới đã và đang được phát triển đó là VLC ứng dụng trong Solid State Lighting (SSL) hoặc công nghệ hiển thị độ phân giải cao. Mặc dù hệ thống VLC dùng Light Emitting Diode (LED) chiếm tỷ lệ vượt trội khi hỗ trợ tốc độ dữ liệu cao, giá rẻ, ... Tuy nhiên với sự phát triển vượt bậc trong nhưng năm gần đây, cùng với những ưu điểm như như màu sắc dễ chịu, khả năng kiểm soát màu sắc tốt, diện tích phát xạ lớn, ...OLED được kỳ vọng sẽ trở thành công nghệ phát sáng quan trọng trong tương lai.

Hiện nay, trong các hệ thống thông tin quang VLC đa phần dùng kiểu điều chế Orthogonal Frequency Division Multiplexing (OFDM). Việc điều chế bằng OFDM có ưu điểm là giúp cho băng thông của tín hiệu tiết kiệm được rất nhiều. Nếu dùng LED đơn có thể đạt đến 3Gbps cho khoảng băng thông khoảng 380Mhz tương ứng với 7.89bits/s/Hz trong khoảng cách 5cm, và khoảng 80Mbits/s trong 20Mhz tương ứng 4bits/s/Hz trong khoảng cách 1m. Nhưng có một khuyết điểm là OFDM xử lý tín hiệu phức tạp trong miền Digital Signal Processing (DSP), vì ngoài việc xử lý qua nhiều giai đoạn, OFDM rất dễ

nhạy cảm với việc trôi tần số nên ta cần các thuật toán và phần cứng để đồng bộ bước sóng quang và tần số phần điện. Vì vậy, ngoài việc tiếp cận bằng OFDM thì có nhiều cách tiếp cận khác để nâng hiệu quả sử dụng băng thông, trong đó có phương pháp điều chế On Off Keying (OOK) sử dụng mã đường truyền Non return to zero (NRZ).

Do đó, đề tài nghiên cứu việc ứng dụng neural network vào giải điều chế tín hiệu NRZ trong hệ thống VLC dùng OLED.

Câu hỏi nghiên cứu đặt ra của luận văn là:

- Phương pháp neural network có tốt hơn phương pháp giải điều chế truyền thống hay không?
- Mô hình và thông số nào của neural network phù hợp nhất cho việc giải điều chế?
- Mô hình và phương pháp nào là cách tốt nhất cho việc tiền xử lý dữ liệu.

Trong chương 2, cơ sở lý thuyết về hệ thống VLC, neural network và các phương pháp tiền xử lý sẽ được trình bày. Trong chương 3, các kết quả đo đạc và giải điều chế sẽ được so sánh và phân tích. Cuối cùng, chương 4 đưa ra kết luận chung.

1.2 Phạm vi và phương pháp nghiên cứu

- Phạm vi: thử nghiệm việc giải điều chế NRZ dùng 3 phương pháp neural networks.
- Phương pháp nghiên cứu: đo đạc tại lab 209B1, với hệ thống quang không dây dùng OLED. Các phương pháp để tiền xử lý dữ liệu gồm có CWT, framing, Denoising Autoencoder (DAE). Các phương pháp sử dụng để giải điều chế là PNN, GRNN, DTNN.

1.3 Các đóng góp

- Đo đạc tín hiệu NRZ và xung lorentz trên hệ thống VLC dùng OLED, tính toán Signal to Noise Ratio (SNR), BER và xử lý dữ liệu thu được từ osciloscope để đưa vào mô hình phân loại.
- Xây dựng thành công các mô hình neural network cho phép phân loại các mức tín hiệu NRZ và xung lorentz và so sánh của những mô hình trên.
- Nghiên cứu những phương pháp tiền xử lý để gia tăng độ chính xác cho mô hình phân loại.

Chương 2. CƠ SỞ LÝ THUYẾT

2.1 Tổng quan về hệ thống VLC

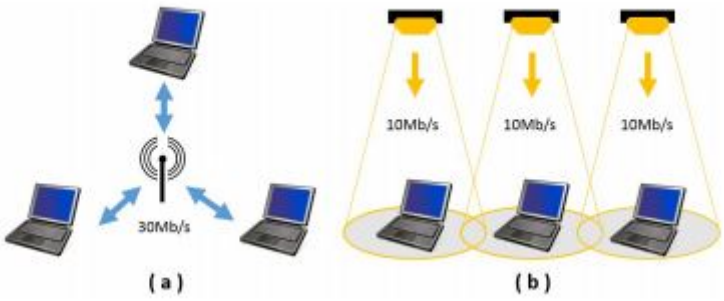
2.1.1 Giới thiệu

Hệ thống VLC sử dụng ánh sáng khả kiến có tần số từ 400 - 800 THz (780–375 nm), có ưu điểm về hiệu quả trong việc sử dụng vùng quang phổ cao, vùng băng thông rộng không cần giấy phép và độ an toàn cao. Khi so sánh với công nghệ truyền thông chủ yếu được sự dụng ở thời điểm hiện tại là RF, VLC có một số ưu điểm như:

- Thứ nhất VLC có băng thông tầng số gấp nhiều lần so với băng thông của RF (300 THz của VLC và 300 Ghz của RF). Trong RF vùng phổ không cần giấy phép là từ 57-66 GHz ở vùng băng thông siêu cao tần. Nó vẫn nhỏ hơn là vùng phổ so với VLC, vì ở VLC thì không cần giấy phép và có thể sử dụng miễn phí. Điều đó có nghĩa là VLC có thể hỗ trợ tốc độ dữ liệu cao và có thể sử dụng lại băng thông.

- Thứ hai, VLC có thể an toàn hơn so với RF. Nó không thể xuyên qua tường và các vật thể trong suốt khác trong điều kiện trong nhà. Điều đó có nghĩa là thông tin truyền đi của người sử dụng sẽ được giới hạn trong phòng. Vì vậy, an toàn thông tin có thể được đảm bảo.

Ví dụ [7], trong hình 2.1 chỉ ra cách hệ thống VLC có thể sử dụng lại quang phổ một cách hiệu quả trong vùng không gian nhỏ. Trường hợp a) chỉ ra rằng kênh truyền WiFi trong đó ba người dùng chia sẽ nhau băng thông 30 Mb/s với trường hợp b) mỗi user sử dụng riêng 10 Mb/s trên kênh truyền VLC. Mặc dù tổng băng thông sử dụng trong hai trường hợp là như nhau nhưng trường hợp b) tốt hơn vì hiệu ứng chồng lấn phổ trên kênh truyền RF.



Hình 2.1: So sánh băng thông sử dụng của RF và VLC

2.1.2 Kỹ thuật điều chế [1]

Khác với điều chế trong giao tiếp RF vì sự không mã hóa đặc trưng thông tin thành pha và biên độ của tín hiệu ánh sáng. Hệ thống mã hóa trong VLC được thực hiện dựa trên cường độ của sóng ánh sáng. Giải điều chế bằng cách phát hiện trực tiếp dựa trên tín hiệu thu được. Có hai vấn đề chính được xem xét trong việc thiết kế hệ thống điều chế cho VLC gồm:

- Dimming: những hoạt động khác nhau yêu cầu độ sáng khác nhau. Như là 30-100 lux yêu cầu cho những hoạt động thị giác thông thường tại nơi công cộng, tuy nhiên có những ứng dụng dành cho văn phòng hay nhà ở cần đến 300 - 1000 lux. Mối quan hệ không tuyến tính của ánh sáng đo đạc (measured light) và ánh sáng cảm nhận được (perceived light) được cho bởi:

$$Perceived\,light(\%) = 100 \times \sqrt{\frac{Measured\,light(\%)}{100}} \tag{2.1}$$

- Flickering: sự thay đổi về độ sáng của ánh sáng điều chế nên được thực hiện theo cách để con người không nhận ra được sự biến đổi. Theo tiêu chuẩn IEEE 802.15.7, dao động của cường độ ánh sáng nên lớn hơn 200 Hz để tránh những tác động có hại cho mắt.

Những phương pháp điều chế thường được sử dụng như:

- OOK
- Pulse Modulation
- OFDM
- Color Shift Keying (CSK)

Bảng 2.1: Bảng so sánh các phương pháp điều chế trong VLC

.

Phương pháp điều chế	BER	SNR (dB)	Data rate
PWM	Cao	Thấp	Thấp nhất
PPM	Cao	Trung bình	Thấp
CSK	Thấp hơn	Trung bình	Trung bình
OOK	Thấp nhất	Cao	Trung bình
MIMO OFDM	Cao	Thấp	Cao nhất

Tiêu chuẩn IEEE 802.15.7 định nghĩa 3 loại lớp vật lý trong hệ thống VLC, được phân loại theo tốc độ dữ liệu. PHY I có tốc độ dữ liệu từ 11.67 kb/s đến 266.7 kb/s, PHY II từ 1.25 Mb/s đến 96 Mb/s, PHY III từ 12 Mb/s đến 96 Mb/s. Loại điều chế được sử dụng cho PHY I và PHY II là OOK và Variable Pulse Position Modulation (VPPM). PHY III sử dụng một kỹ thuật điều chế riêng biệt gọi là CSK, trong đó sử dụng nhiều nguồn sáng tổng hợp để tạo ra ánh sáng trắng. PHY I được tối ưu cho những ứng dụng với tốc độ thấp, khoảng cách xa ứng dụng ngoài trời như với xe cộ, đèn đường. Trong khi đó PHY II được thiết kế để hoạt động trong điều kiện trong nhà và ứng dụng point-to-point sử dụng tốc độ dữ liệu cao.

2.1.3 Nhiễu và méo dạng phi tuyến

1. Méo dạng phi tuyến: Đây là nguyên nhân chính gây ra sự giới hạn băng thông trong OLED. Tính phi tuyến làm cho tín hiệu truyền đi bị xén và méo dạng. Cụ thể, có 2 nguyên nhân chính gây ra sự méo dạng phi tuyến. Thứ nhất là quá trình ánh xạ phi tuyến trong bộ chuyển đổi quang điện. Thứ 2 là do điện áp của OLED nhỏ hơn V_{on} hoặc dòng của OLED vượt quá dòng tối đa cho phép. Để có thể giảm được sự méo dạng phi tuyến, thông thường có 2 phương pháp là:

   - Pre-distortion compensation
   - Post-distortion compensation

2. Nhiễu: [8] Nhiễu trên kênh truyền VLC gồm 2 loại chính: nhiễu shot và nhiễu nhiệt.

   - Nhiễu shot gây ra bởi LED và ánh sáng ở xung quanh. Những bộ lọc không thể loại bỏ được nhiễu từ ánh sáng đã điều chế. Ánh sáng xung quanh có thể đến từ mặt trời hoặc từ những nguồn sáng khác có thể bị phát hiện bởi photodiode. Loại nhiễu này được được gọi là nhiễu trắng cộng thêm.

     $$N_{shot} = 2qIB \tag{2.2}$$

     Trong đó: q là điện tích electron, B là băng thông của photodetector, I là dòng quang điện

   - Nhiễu nhiệt gây ra bởi phía thu do sự di chuyển của electron. Khi không có trường bên ngoài, chuyển động là ngẫu nhiên và không có dòng điện tạo ra. Tuy nhiên trong thực tế, các electron chuyển động theo một hướng không bằng các electron chuyển động theo hướng ngược lại. Điều này làm tăng hoặc giảm điện áp của chất bán dẫn và vật dẫn.

     $$N_{thermal} = \frac{4KTBN_{circuit}}{R} \tag{2.3}$$

     Trong đó: K là hằng số Boltzmann, T là nhiệt độ, $N_{circuit}$ là nhiễu của mạch điện, R là điện trở tải.

   Tổng nhiễu tác động lên hệ thống VLC là:

   $$N_{total} = \sqrt{(N_{shot})^2 + (N_{thermal})^2} \tag{2.4}$$

2.1.4 Ứng dụng của hệ thống VLC - LiFi [1]

Năm 2011, Harald Hass đặt những viên gạch đầu tiên cho thuật ngữ Light Fidelity (Li-Fi). Li-Fi là kết nối hai chiều tốc độ cao sử dụng hệ thống quang không dây bằng ánh sáng khả kiến. Dưới dây là những ứng dụng phổ biến của Li-Fi:

1. Giao tiếp giữa các phương tiện giao thông: cảnh báo kẹt xe, phanh điện tử khẩn cấp, cảnh báo tốc độ, ...

2. Truyền thông dưới nước: tín hiệu RF truyền không tốt trong môi trường nước vì tính dẫn điện tốt của nó. Vì vậy VLC có thể dùng cho mạng truyền thông dưới nước.

3. Trong bệnh viện: một số vùng nhạy cảm với sóng điện từ (như MRI scanner) cũng nên được chuyển đổi sang hệ thống VLC để tránh gây nhiễu lên các thiết bị khác.

4. Thông tin hiển thị thông qua biển báo: hiển thị những thông tin quan trọng tại những nơi như trạm xe bus, sân bay, bảo tàng, bệnh viện, ...

5. Hệ thống ID với ánh sáng khả kiến:

6. Hệ thống giao tiếp bằng âm thanh

7. Ứng dụng trong Wireless local area networks (WLAN): hệ thống VLC dùng LED có thể được ứng dụng trong mạng LAN và cung cấp tốc độ truyền lên đến 10 Gpbs.

2.2 Các thành phần của hệ thống VLC

Hình 2.2: Sơ đồ khối hệ thống quang không dây

2.2.1 OLED

OLED - Công nghệ của tương lai: Trong thập kỷ qua, chúng ta đã chứng kiến sự phát triển mạnh mẽ của công nghệ SSL. Theo Allied Market Research [9], giá trị thị trường của SSL năm 2019 là 32.65 tỷ USD và dự kiến đạt 74.25 tỷ USD vào năm 2027, phát triển dựa trên 3 công nghệ chính là LED, Polymer Light Emitting Diode (PLED) và OLED. Theo Businesswire [10], giữa thời kỳ khủng hoảng COVID-19, thị trường OLED dùng cho chiếu sáng vẫn đạt 67.1 triệu USD vào năm 2020 và có thể đạt được 291.1 triệu USD vào 2027, với tốc độ tăng trưởng hàng năm đạt 23.3 % từ 2020 đến 2027. Tập trung trong các lĩnh vực như nhà ở, văn phòng, công nghiệp, bệnh viện, ngoài trời, cửa hàng và tự động hóa. Những lĩnh vực như kiến trúc, bệnh viện, cửa hàng sẽ là những lĩnh vực đi đầu. Kế đến, tự động hóa cũng nhận được sự cam kết phát triển triển của các công ty, ví dụ như BMW nếu như tuổi thọ và độ tin cậy được cải thiện. Cuối cùng, các lĩnh vực như nhà ở, văn phòng, ngoài trời sẽ đi theo sau nếu như giá sản xuất giảm và tuổi thọ được kéo dài.

OLED mang lại ánh sáng mềm mại, không đổ bóng, không chói mắt, mát khi chạm vào. Đó là thứ ánh sáng thuần khiết và mang một vẻ đẹp huyền diệu. Khi so sánh với LED, OLED mang những ưu điểm nổi trội như màu sắc dễ chịu, khả năng kiểm soát màu sắc tốt, diện tích phát xạ lớn, ... Tuy nhiên, tuổi thọ, hiệu suất và đặc biệt là chi phí sản xuất cao vẫn là những khuyết điểm lớn của OLED khi so với LED.

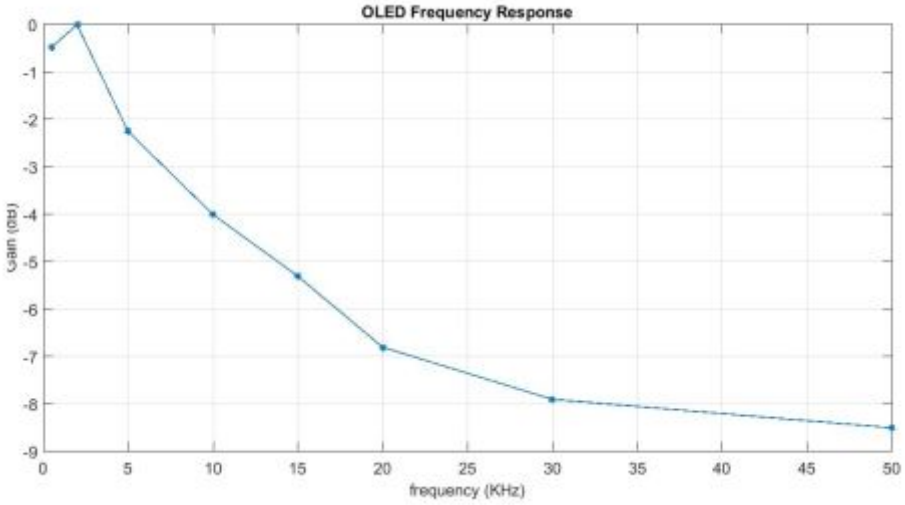
OLED trong hệ thống VLC: Từ những ưu điểm và tiềm năng như trên OLED sẽ một công nghệ mạnh mẽ cho lĩnh vực chiếu sáng trong tương lai, nhưng giờ đây, ngoài khả năng chiếu sáng, khoảng băng tần khả kiến của chúng còn có thể sử dụng cho truyền nhận dữ liệu không dây, mà ở đó cũng chính là nguồn phát dữ liệu.

Hình 2.3: OLED được ứng dụng làm đèn chiếu sáng trong văn phòng [5]

Hình 2.4: Khả năng tùy chỉnh hình dạng của OLED

Cả LED và OLED đều một lựa chọn rất được các nhà nghiên cứu quan tâm để dùng làm nguồn sáng cho công nghệ VLC. Tuy nhiên, OLED lại có băng thông điều chế thấp hơn rất nhiều so với đèn LED thông thường. Không giống như LED thông thường, OLED là tập hợp các phân tử hữu cơ, do đó độ linh động của nó thấp hơn nhiều so với các thiết bị silicon khác, kết quả là sẽ giới hạn băng thông điều chế của nó. Băng thông điều chế của OLED còn phụ thuộc vào nhiều thứ, như là quá trình sản xuất, kích cỡ...nhưng băng thông thông thường của chúng chỉ dừng lại ở vài trăm KHz.

Trước đây, có rất ít nghiên cứu sử dụng OLED trong VLC. Tiêu biểu như, tốc độ là 51.6 Mb/s với OLED 3 màu băng thông là 460 kHz, điều chế OFDM, cân bằng MLP-ANN, với khoảng cách là 5 cm [11]; tốc độ 1.4 Mb/s đạt được với điều chế discrete multitone khi băng thông nhỏ hơn 100 kHz [12]; và tốc độ 2.7 Mb/s với OLED trắng và cân bằng thời gian thực ANN với băng thông là 93 kHz [13]; truyền thời gian thực tốc độ 138 kp/s, băng thông 7 kHz, điều chế DPPM, khoảng cách 40 cm [14].



Hình 2.5: Đáp ứng của OLED Lumiable sáng trằng tiêu chuẩn

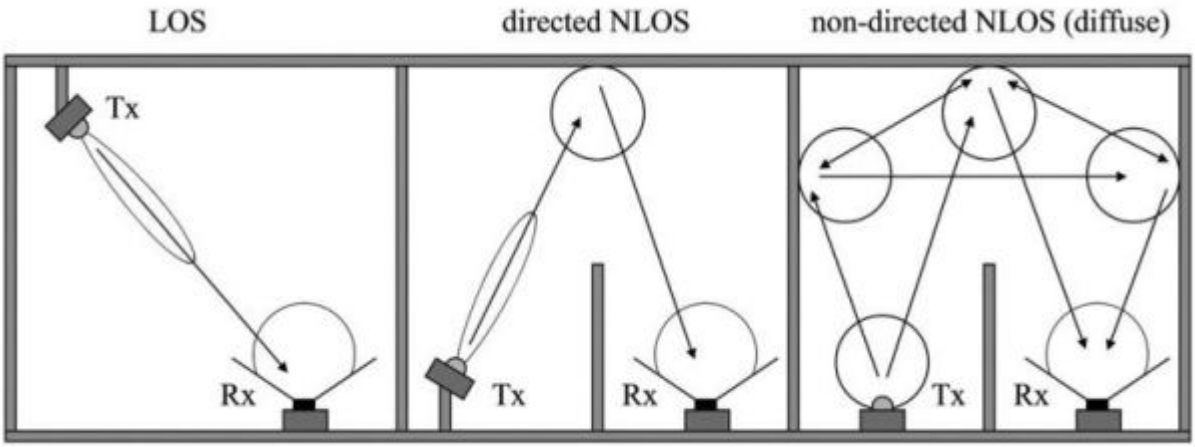
2.2.2 Photodiode

Photodiode là một diode bán dẫn thực hiện biến đổi photon thành dòng điện theo hiệu ứng quang điện, các photon có thể thuộc vùng phổ của ánh sáng khả kiến, hồng ngoại hay tử ngoại,... Photodiode sử dụng trong khảo sát là ADP do thorlab sản xuất, nó gồm một photodiode silic và một bộ khuyếch đại biến đổi trở kháng TIA (Transimpedance Amplifier). Bộ thu được thiết kế để thu được tín hiệu quang có bước sóng từ 400 đến 1000 nm, độ lợi có thể thay đổi từ 0 đến 70dB, điện áp ra 5V với trở kháng tải ngoài 50 Ω .

Hình 2.6: Bộ thu quang APD410A/M của Thorlab

2.2.3 Kênh truyền quang

Kênh truyền quang học đã được chứng minh là một kênh truyền tuyến tính, biến đổi theo thời gian, không nhớ với đáp ứng xung là hữu hạn theo thời gian. Đặc điểm cơ bản của kênh truyền quan là phần suy hao quang học, việc mô hình hóa kênh truyền quang để thể hiện gồm 2 loại tia truyền: Light of sight (LOS) và Non light of sight (NLOS).Việc truyền tốc độ cao sẽ khiến băng thông tín hiệu lớn, khi băng thông coherence lớn, kênh truyền quang được xem như là kênh truyền chọn lọc tần số. Trong đó loại tia truyền LOS cung cấp khả năng cường độ tốt hơn tại bộ phát và thu và băng thông coherence tốt hơn, nó thích hợp với các đường truyền có tốc độ cao, tuy nhiên không phù hợp với các ứng dụng di động cao, đường truyền dễ bị bẻ gãy hoặc bị gián đoạn. Trong khi đó, truyền thông bằng NLOS cung cấp biên độ thấp hơn, băng thông coherence nhỏ hơn tuy nhiên độ di động cao hơn nhiều so với truyền thông LOS do NLOS khai thác các đặc tính phản xạ của các đối tượng, cường độ bức xạ tại máy thu, đồng thời truyền thông NLOS có thể được tăng cường bằng cách khuếch tán.



Hình 2.7: Mô hình truyền thông LOS và NLOS

Đáp ứng xung của kênh truyền có thể được mô hình hóa theo phương trình bên dưới cho cả truyền thông LOS và NLOS:

$$h(t) = g_{h(opt)} U(t) \frac{6a^6}{(t+a)^7} \tag{2.5}$$

Trong đó: g là độ lợi của kênh truyền, U(t) là hàm bước, a là băng thông trải trễ của kênh truyền. Tuy nhiên trong truyền thông LOS, do khoảng cách giữa máy phát và máy thu là ngắn (khoảng 1m). Ngoài ra, không giống như trong truyền thông vô tuyến, trong truyền thông LOS, băng thông coherence được định nghĩa là tốc độ chênh lệch của từng bước sóng trong nguồn phát Laser hay LED. Khi đó, nếu khoảng cách máy phát và máy thu đủ xa thì ta mới thấy sự chênh lệch của các bước sóng nội tại của nguồn, lúc đó ta xem như có nhiều tia sáng trong một luồng LOS, nhiễu Intersymbol Interference (ISI) sẽ xuất hiện, trong trường hợp nhiễu ISI sẽ xuất hiên, khoảng guard interval sẽ được thêm vào để giảm lượng chồng lấn giữa các xung, qua đó làm giảm hiệu suất sử dụng phổ. Tổng quát, trong truyền thông LOS, ta xem như đáp ứng xung của kênh truyền là: $h(t) = \delta(t)$

2.2.4 Mạch pre-emphasis: [2]

Để vượt qua giới hạn băng thông điều chế hẹp của nguồn sáng, mạch pre-emphasis đã được sử dụng để tăng băng thông cho hệ thống VLC dùng OLED.

Về cơ bản, mạch pre-emphasis điều chỉnh lại bộ lọc band-pass tích cực bằng cách sử dụng OP-AMP với mục đích là bù lại phần tần số từ DC đến 100kHz. Băng thông điều chế trong hệ thống OLED VLC có thể tăng 45 lần.

2.3 Ứng dụng neural networks để phân loại tín hiệu

2.3.1 Giới thiệu

Nguyên lý và thiết kế của mạng neural nhân tạo đã có sự phát triển vượt bật trong 20 năm gần đây. Có nhiều ứng dụng ảnh hưởng trực tiếp đến việc xử lý số tín hiệu. Neural network có thể học được trong môi trường giám sát và không giám sát trở nên rất phù hợp trong việc giải quyết những khó khăn của trong vấn đề xử lý tín hiệu. [15]

Một trong những xu hướng chính trong việc ứng dụng neural networks vào xử lý số tín hiệu là phân loại tín hiệu. Tín hiệu được phân loại có thể là tín hiệu thu được từ cảm biến [16], tín hiệu điện não đồ [17], tín hiệu điện tim đồ [18], ...Trong báo cáo này, chúng tôi sẽ ứng dụng các phương pháp phân loại tín hiệu bằng neural networks cho tín hiệu thu được từ hệ thống VLC.

Tín hiệu ban đầu sau khi đi qua một loạt khối và kênh truyền đã bị nhiễu, méo dạng, không còn giống như tín hiệu ban đầu. Việc sử dụng neural network sẽ giúp phân biệt được các mức ngõ ra của tín hiệu. Mô hình được sử dụng ở đây thuộc loại là phân loại (classification). Với ngõ vào của mô hình là tín hiệu ngõ ra bị nhiễu thu được tại photodiode, ngõ ra của mô hình là mức tín hiệu đúng.

Trước đây, đã có khá nhiều nghiên cứu trên lĩnh vực trên như: Dùng Convolutional Neural Networks (CNN) để giải điều chế BPSK [19], sử dụng Deep Belief Network để giải điều chế với kênh truyền AWGN [20], ...Báo cáo khoa học [21] thực hiện đề tài tương tự đối với hệ thống VLC dùng LED.

Báo cáo này sẽ phát triển ý tưởng trên thông qua việc sử dụng 3 phương pháp: PNN, GRNN, DTNN trên hệ thống VLC dùng OLED.

2.3.2 Probabilistic Neural Networks

Neural network đã thường xuyên được ứng dụng trong việc phân loại mẫu bằng cách học từ những ví dụ. Những phương pháp hiện tại như backpropagation yêu cầu một thời gian tính toán dài dành cho việc training và dễ bị mắc phải việc tìm sai cực tiểu. Để khắc phục được vấn đề này, một phương pháp phân loại dựa trên những định lý thống kê đã có được tìm ra.

PNN là một bộ phân loại Bayes-Parzen, nền tảng của phương pháp này xuất hiện từ thập niên 1960, tuy nhiên phương pháp này không phổ biến vì năng lực tính toán yếu của máy tính lúc bấy giờ. PNN được giới thiệu đầu tiên bởi Specht (1990). Ông đã chỉ ra cách mà bộ phân loại Bayes-Parzen có thể được chia thành một số lượng lớn các quy trình đơn giản được thực hiện trong mạng neurons nhiều lớp, mỗi quy trình có thể chạy song song độc lập.

Bộ phân loại Bayes: [22] Xét một bài toán với K lớp $(1, 2, \ldots, k, ..., K)$. Giả sử có một điểm dữ liệu $x \in \mathbb{R}^d$, xác suất để điểm dữ liệu này rơi vào class k.

$$p(y = k|x) \tag{2.6}$$

Hoặc viết gọn thành $p(k|x)$. Tức tính xác suất để đầu ra là class k biết rằng đầu vào là vector x. Từ đó có thể giúp xác định được class của điểm dữ liệu đó bằng cách chọn ra class có xác suất cao nhất:

$$k = \arg \max_{k \in 1,...,k} p(k|x) \tag{2.7}$$

Biểu thức trên khó có thể tính trực tiếp, thay vào đó quy tắc Bayes thường được sử dụng:

$$k = \arg \max_k p(k|\mathbf{x}) \ = \arg \max_k \frac{p(\mathbf{x}|k)p(k)}{p(\mathbf{x})} \ = \arg \max_k p(\mathbf{x}|k)p(k) \tag{2.8}$$

Trong đó $p(k)$ có thể được hiểu là xác suất để một điểm rơi vào lớp k. Giá trị này có thể được tính bằng Maximum likehood estimation, tức tỉ lệ số điểm dữ liệu trong tập training rơi vào class này chia cho tổng số lượng dữ liệu trong tập training, hoặc cũng có thể ước lượng bằng MAP estimation. Phần còn lại $p(x|k)$ là phân phối của các điểm dữ liệu trong lớp k (hàm mật độ xác suất (PDF))

Vấn đề lớn nhất với phương pháp phân loại Bayes là PDF thường chưa biết. Thông thường thì phân phối được giả sử là Gaussian. Độ chính xác của đường biên phụ thuộc nhiều vào độ chính xác của việc ước lượng PDF. Các phương pháp ước lượng thông dụng có thể kể đến như: Parzen windows, k nearest neighbor, Potential function. [23]

Ước lượng Parzen (Parzen window): [24] Đây là phương pháp phổ biến nhất được Parzen (1962) chỉ ra để có thể ước lượng được $f(X)$.

$$f(x) \approx f_n(X) = \frac{1}{n\lambda} \sum_{i=1}^{n} g(\frac{X - X_{Ai}}{\lambda}) \tag{2.9}$$

Trong đó: g gọi là window hoặc hàm kernel, X là biến ngẫu nhiên của một điểm trong không gian mẫu, X_{Ai} là biến ngẫu nhiên độc lập có phân phối giống như X, λ là chiều rộng của window, tham số độ mượt, hay đơn giản là kích thước kernel, là hàm của n sao cho:

$$\lim_{n \to \infty} \lambda = 0 \quad and \quad \lim_{n \to \infty} n\lambda = \infty \tag{2.10}$$

Cacoullos (1966) đã mở rộng kết quả của Parzen để giải quyết trường hợp đa biến. Trong trường hợp cụ thể là Gaussian kernel, ước lượng đa biến có thể được biểu diễn như sau: [25]

$$f_A(X) = \frac{1}{(2\pi)^{p/2}\sigma^p} \frac{1}{m} \sum_{i=1}^{m} \times \exp(-\frac{(X - X_{Ai})^T(X - X_{Ai})}{2\sigma^2}) \tag{2.11}$$

Trong đó:

- i là số mẫu
- m là tổng số mẫu training
- X_{Ai} là mẫu training thứ i của loại θ_A
- σ là độ lệch chuẩn
- p là chiều của không gian đo lường

Kiến trúc Mạng PNN được Specht (1990) giới thiệu gồm có 4 layers: [25] [23]

1. Input layer: là tập hợp p các kết nối để lấy vector ngõ vào và truyền chúng cho các layer tiếp theo.

2. Pattern layer: quá trình này sẽ được thực hiện thông qua 2 bước:

   Bước thứ nhất: mỗi unit sẽ so sánh mẫu dành riêng của nó với vector ngõ vào bằng cách tính khoảng cách $D(x, X_i^{(j)})$. Giá trị của $D(x, X_i^{(j)})$ thể hiện mức độ đóng góp của mẫu training cho ngõ ra trong mẫu test. Nếu $D(x, X_i^{(j)})$ nhỏ, có nghĩa là nó có nhiều đóng góp cho ngõ ra. Nếu $D(x, X_i^{(j)})$ lớn, có nghĩa là nó có rất ít đóng góp cho ngõ ra. Nếu $D(x, X_i^{(j)})$ bằng 0, có nghĩa là dữ liệu test giống với dữ liệu train và ngõ ra của dữ liệu test sẽ là ngõ ra của mẫu train.

   - Thực hiện phép tính dot product: (Specht, 1990)

   $$D(x, X_i^{(j)}) = \sum_{k=1}^{p} x_k \cdot X_{i,k}^{(j)} \tag{2.12}$$

   Trong đó: x_k là phần tử thứ k của vector ngõ vào, $X_{i,k}$ là phần tử training thứ k của mẫu thứ i thuộc về lớp j. Cả 2 vectors phải được chuẩn hóa trước khi thực hiện phép nhân.

   - Khoảng cách Euclidean:

   $$D(x, X_i^{(j)}) = \sqrt{\sum_{i=1}^{p} (x_i - X_i^{(j)})^2} \tag{2.13}$$

   - Khoảng cách Manhattan (or city block):

   $$D(x, X_i^{(j)}) = \sum_{i=1}^{p} |x_i - X_i^{(j)}| \tag{2.14}$$

   Bước thứ hai: Thay vì sử dụng hàm sigmoid thông dụng trong backprobagation, hàm không tuyến tính (kernel) được sử dụng. Một số hàm dành cho những khoảng cách ở trên như:

   - Cho dot product: (Specht, 1990)

   $$K(x, X_i^{(j)}) = \exp\left(\frac{D(x, X_i^{(j)}) - 1}{2\sigma^2}\right) \tag{2.15}$$

   - Cho khoảng cách Euclidean:

   $$K(x, X_i^{(j)}) = \exp\left(\frac{-D(x, X_i^{(j)})}{2\sigma^2}\right) \tag{2.16}$$

   - Cho khoảng cách Manhattan:

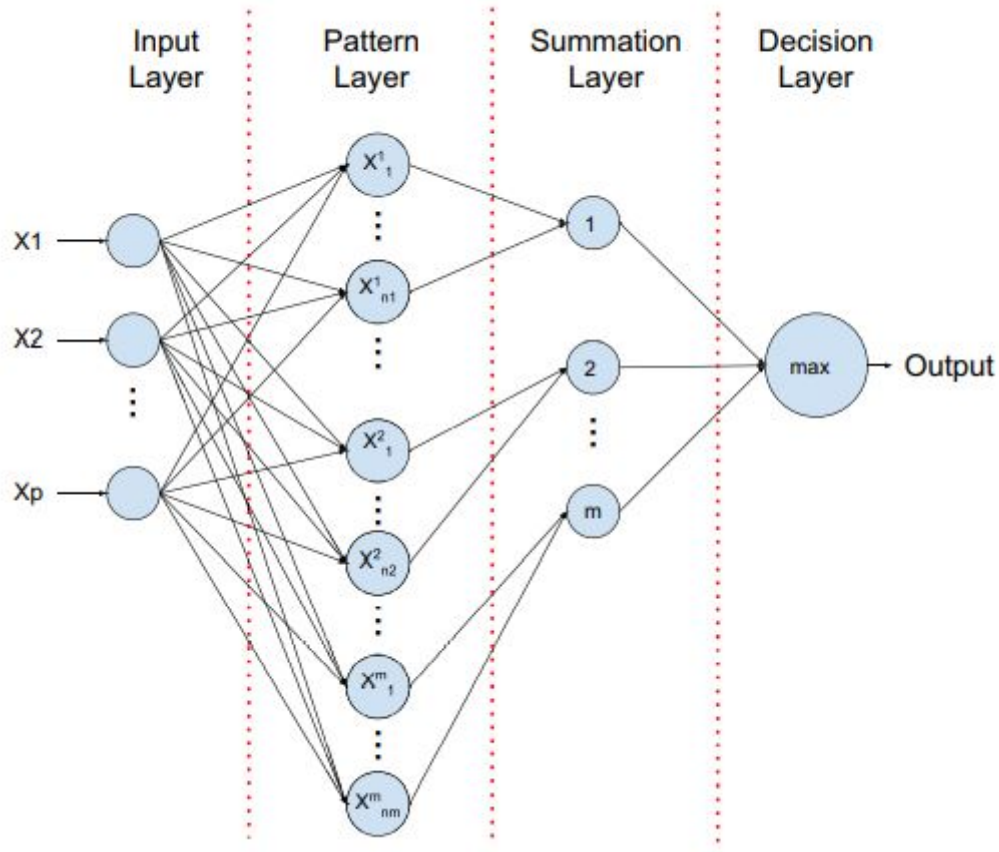
   $$K(x, X_i^{(j)}) = \exp\left(\frac{-D(x, X_i^{(j)})^2}{2\sigma^2}\right) \tag{2.17}$$

Từ đó có thể ước lượng hàm PDF bằng công thức:

$$\hat{f}_{j,n} = \frac{1}{\lambda_{nj}^{p}} \sum_{i=1}^{n_j} K(x, X_i^{(j)}) \tag{2.18}$$

3. Summation layer: các units đơn giản là tính tổng của các neuron ở pattern layer tương ứng với loại mà mẫu training được chọn. Một đặc trưng khác có thể kết hợp trong summation layer là tính hệ số: $\hat{d}_j(x) = p(xk)p(k)$ để dùng bộ phân loại Bayes.

4. Output layer (Decision layer): có vai trò là chọn $\hat{d}_j(x)$ lớn nhất và khai báo vector đầu vào x thuộc lớp j



Hình 2.8: Kiến trúc của PNN

Ưu điểm:

- Ưu điểm lớn nhất của PNN là có thể training dễ dàng và nhanh chóng
- Có thể ứng dụng cho hệ thống real time
- Mẫu thêm vào có thể được giám sát và lưu trữ trong mạng
- Sự tổng quát hóa được cải thiện và đường biên đươc quyết định trở nên phức tạp hơn.

Nhược điểm:

- Chậm hơn multilayer perceptron về khía cạnh phân loại dự liệu mới
- Yêu cầu nhiều không gian bộ nhớ để lưu trữ mô hình.

Những ứng dụng của PNN [23]

- Phân loại những mẫu dữ liệu có nhãn
- Phân loại những mẫu có dữ liệu là hàm mật độ xác suất thay đổi theo thời gian
- Xử lý số tín hiệu, với dạng sóng là mẫu dữ liệu
- Thuật toán không giám sát với dữ liệu chưa dán nhãn

2.3.3 General Regression Neural Networks

GRNN là một loại single-pass neural networks, được đề xuất bởi Donald J. Specht (1991). Dựa trên nền tảng toán học GRNN cung cấp một mạng có thể dễ dàng sử dụng mà có thể giảm thời gian training và tăng hiệu năng cho mô hình.

General regression: Giả sử $f(x, y)$ biểu diễn cho một hàm mật độ xác suất kết hợp liên tục của vector biến ngẫu nhiên x và biến vô hướng y (scalar). Gọi X là giá trị cụ thể của biến ngẫu nhiên x, trung bình có điều kiện của y được cho bởi X (còn được gọi là hồi quy của y trên X) là:

$$E[y|X] = \frac{\int_{-\infty}^{\infty} yf(X, y)dy}{\int_{-\infty}^{\infty} f(X, y)dy}. \tag{2.19}$$

Khi hàm mật độ $f(x, y)$ là chưa biết, nó thường được ước lượng bằng mẫu giám sát x và y. Để ước lượng phi tham số cho $f(x, y)$, chúng ta có thể sử dụng phương pháp ước lượng đồng nhất được cho bởi Parzen và ứng dụng cho trường hợp đa biến của Cacoullos. Xác suất ước lượng $\hat{f}(X, Y)$ phụ thuộc vào giá trị X^i và Y^i của biến ngẫu nhiên x và y, khi n là số lượng mẫu giám sát và p là số chiều của vector biến x.

$$\hat{f}(X, Y) = \frac{1}{(2\pi)^{(p+1)/2}\sigma^{p+1}} \cdot \frac{1}{n} \sum_{i=1}^{n} \exp[-\frac{(X - X^i)^T (X - X^i)}{2\sigma^2}] . \exp[-\frac{(Y - Y^i)^2}{2\sigma^2}]. \tag{2.20}$$

Thay 2.19 vào 2.20, chúng ta được:

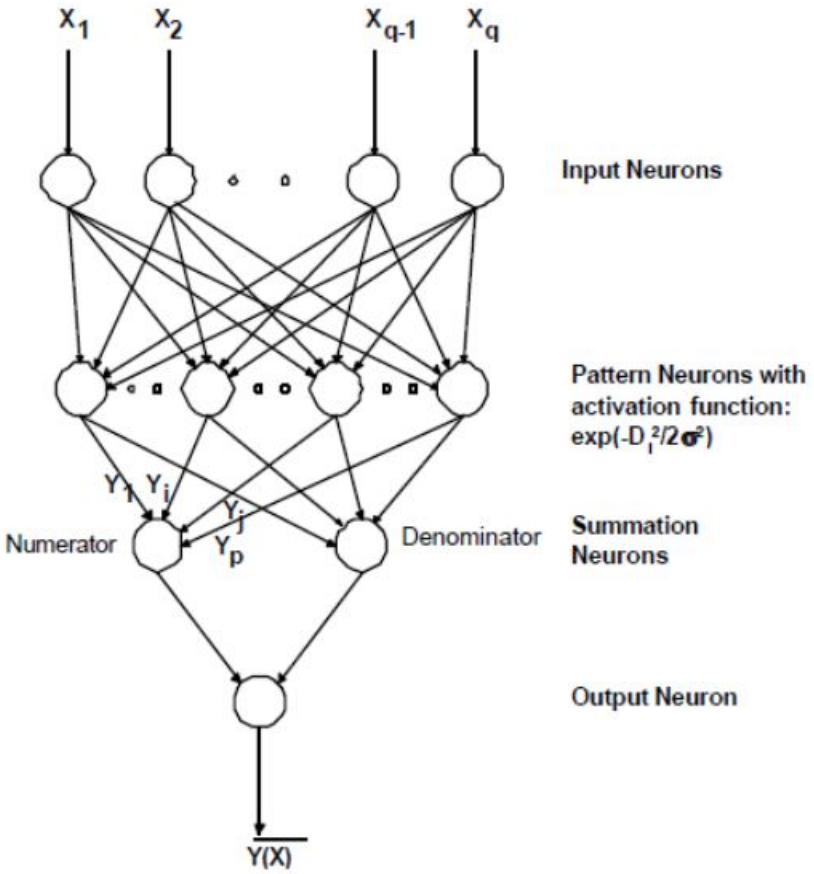
$$\hat{Y}(X) = \frac{\sum_{i=1}^{n} \exp[-\frac{(X-X^i)^T (X-X^i)}{2\sigma^2}] \int_{-\infty}^{\infty} y \exp[-\frac{(y-Y^i)^2}{2\sigma^2}]dy}{\sum_{i=1}^{n} \exp[-\frac{(X-X^i)^T (X-X^i)}{2\sigma^2}] \int_{-\infty}^{\infty} \exp[-\frac{(y-Y^i)^2}{2\sigma^2}]dy} \tag{2.21}$$

Trong đó: $D_i^2 = (X - X^i)^T (X - X^i)$ là khoảng cách Euclidien. Từ đó, chúng ta có hàm activation là $\exp(-D_i^2/2\sigma^2)$.

Chúng ta chỉ có một tham số chưa biết là σ. Quá trình training sẽ tìm ra điểm tối ưu của σ bằng cách tìm giá trị nhỏ nhất của Mean Squared Error (MSE) [26]

Kiến trúc: GRNN có 4 layers, với 2 layers đầu tiên tương tự như PNN:

1. Input layer: có nhiệm vụ chuyển giá trị ngõ vào sang các layer tiếp theo.
2. Pattern layer: tính toán khoảng cách và hàm activation.
3. Summation layer: dựa trên biểu thức 2.21, chúng ta chia thành 2 neurons là Numerator (tử số) neuron và Denominator (mẫu số) neurons. Numerator neuron tính tổng các giá trị trọng số từ pattern layer, Denominator neuron tính tổng của các phép nhân giữa trọng số của pattern layer và giá trị đích thực tế.
4. Output layer: chia giá trị thu được từ Numerator neuron cho giá trị của Denominator và sử dụng kết quả đó để dự đoán giá trị đích.



Hình 2.9: Kiến trúc của GRNN

Ưu điểm:

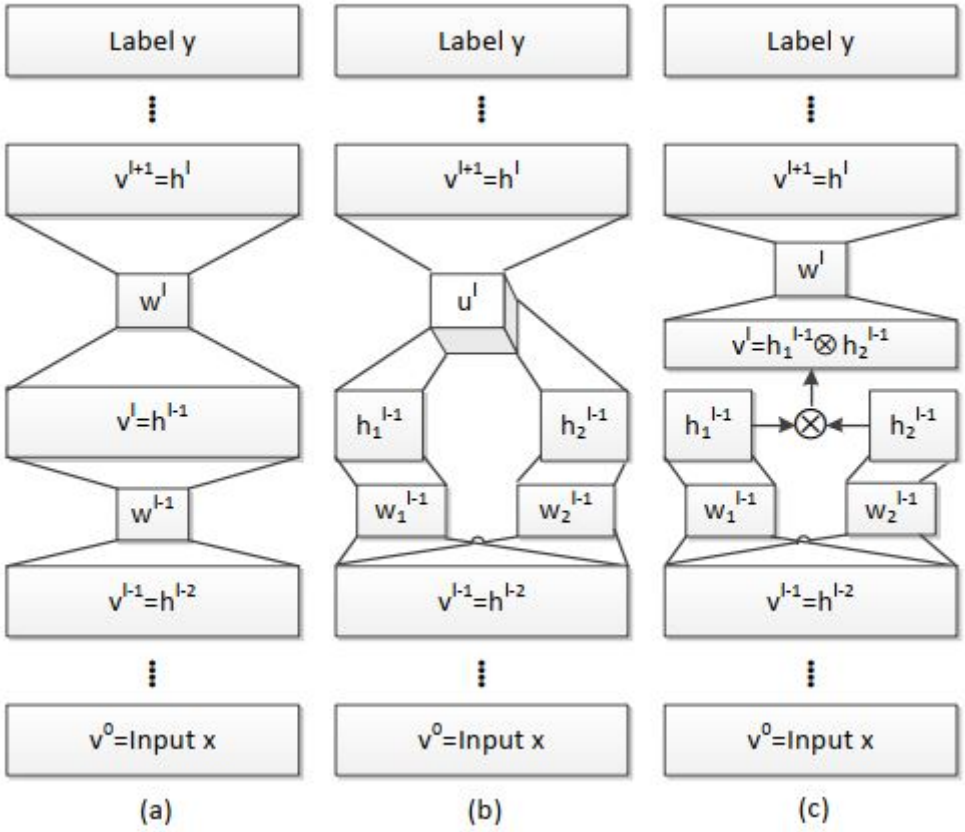
- Quá trình học single-pass, không yêu cầu backpropagation
- Độ chính xác xao dành cho việc ước lượng
- Có thể xử lý được nhiễu ở ngõ vào
- Chỉ yêu cầu một lượng nhỏ dữ liệu

Nhược điểm:

- Kích thước của nó có thể rất lớn, làm tăng chi phí tính toán
- Không có phương pháp tối ưu nào để cải thiện nó

2.3.4 Deep Tensor Neural Networks [3]

Giới thiệu DTNN là một mô hình được mở rộng từ Deep Neural Networks (DNN). Ý tưởng cơ bản của DTNN đến từ việc, những nhận biết của chúng ta đến từ nhiều yếu tố tương tác với nhau và dự đoán ngõ ra. Để thể hiện sự tương tác, chúng ta chiếu ngõ vào thành hai vùng không gian con tuyến tính.



Hình 2.10: Kiến trúc của DNN và DTNN. (a) DNN. (b) DTNN: hidden layer h^{l-1} bao gồm 2 phần h_1^{l-1} và h_2^{l-1}. Hidden layer h^l là tensor layer kết nối weight u^l với three way tensor. (c) Cách biểu diễn khác của (b): tensor u^l được thay thế bằng ma trận w^l với v^l được định nghĩa là tích vô hướng $h_1^{l-1} \otimes h_1^{l-1}$

Kiến trúc Trong đó, ngõ vào của DTNN là x, $I \times 1$ vector, và ngõ ra là y, $C \times 1$ vector. Hidden layer h^{l-1} được chia thành 2 phần h_1^{l-1} ($K_1^{l-1} \times 1$ vector) và h_2^{l-1} ($K_2^{l-1} \times 1$ vector).Hai phần này được kết nối với nhau bởi hidden layer h^l thông qua một three way tensor u^l được biểu diễn bằng hình lập phương như torng hình.

Trong hình (c), góc nhìn thay thế của DTNN được định nghĩa bởi:

$$v^l = vec(h_1^{l-1}(h_2^{l-1})^T) \tag{2.22}$$

Cách viết lại này cho phép chúng ta kết nối giữa những layer thông thường và tensor layers. Khi đó hidden layer h^l có thể được xem như là layer thông thường và có thể được học bằng thuật toán backpropagation. Tuy nhiên với hidden layer h^{l-1} vẫn chứa 2 thành phần h_1^{l-1} và h_2^{l-1}, chúng được xác định bởi 2 ma trận weights w_1^{l-1} và w_2^{l-1}. Chúng ta gọi nó là double-projection layer.

Tóm lại có 3 loại layers trong DTNN: hidden layer thông thường, double-projection layer và softmax layer (sigmoid layer với bài toán binary classification) kết nối layer cuối cùng với nhãn. Với layer dầu tiên, ta có:

$$v^l = v^0 = x \tag{2.23}$$

Với layers $l > 0$, nếu layer phía trước là hidden layer thông thường $v^l = h^{l-1}$. Với hidden layer thông thường ta có:

$$h_i^l = f(z^l(v^l)) = f((w^l)^T v^l + a^l), \tag{2.24}$$

Trong đó, w^l là ma trận trọng số, a^l là bias, $f(x)$ là hàm activation. Với Double-projection layer:

$$h_i^l = f(z_i^l(v^l)) = f((w_i^l)^T v^l + a - i^l) \tag{2.25}$$

Trong đó $i = 1, 2$ là số thứ tự các phần.

2.4 Ứng dụng neural networks để giảm méo dạng phi tuyến

2.4.1 Giới thiệu

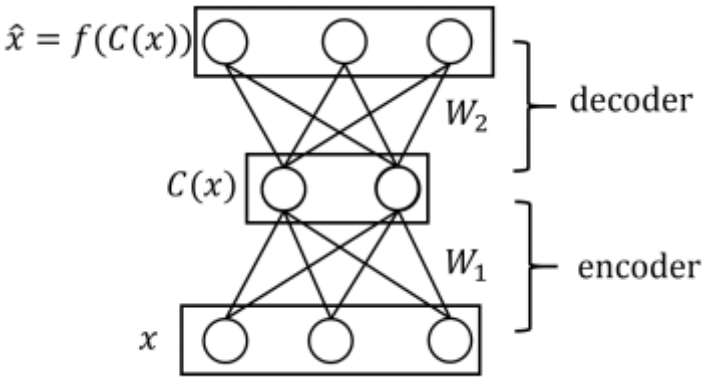
Ước lượng được tín hiệu sạch từ tín hiệu bị nhiễu, méo dạng, ... là một vấn đề rất quan trọng trong việc xử lý số tín hiệu. Trong những năm gần đây, với sự phát triển nhanh chóng của neural networks, nó ngày càng trở nên mạnh mẽ và những ứng dụng của nó cũng phổ biến hơn. Ngoài việc dùng để phân loại tín hiệu như đã trình bày ở chương trước, nó còn thể được ứng dụng vào việc hiệu chỉnh lại tín hiệu bị nhiễu hay méo dạng.

Trước đây, có khá ít nghiên cứu trong lĩnh vực này, tiêu biểu là: sử dụng Adversarial autoencoder để làm giảm méo dạng phi tuyến [27], tăng cường giọng nói bằng DAE [28], ...

2.4.2 Autoencoder Neural Networks

Autoencoder được giới thiệu đầu tiên năm vào thập niên 1980 bởi Hinton và nhóm PDF (Rumelhart và những người khác, 1986) để giải quyết vấn đề về backpropagation khi dữ liệu không có được dán nhãn. Autoencoder (AE) là một loại neural networks thuộc loại unsupervised learning, mục tiêu là học đặc tính của dữ liệu. Xét một mô hình Autoencoder đơn giản, [29] với 3 layers. Trong đó, layer ngõ vào và layer ngõ ra có số neuron bằng nhau, hidden layer thường có số neuron ít hơn. Ma trận W_1 là trọng số kết nối layer ngõ vào và hidden layer, W_2 là kết nối hidden layer và layer ngõ ra. Thông thường, $W_2 = W_1^T$, chúng ta đặt $W_1 = W$ và $W_2 = W^T$. Ngõ vào x khi đi qua layer ngõ vào sẽ tạo thành: $h = \sigma(Wx + b)$ ở hidden layer và $\hat{x} = \sigma(W^T h + c)$ ở ngõ ra.

Mục tiêu cơ bản của autoencoder là tạo ra ngõ ra $\hat{x}$ càng gần với ngõ vào x càng tốt. Tuy nhiên, nó không thể luôn được thực hiện nếu số neuron ở hidden layer nhỏ hơn nhiều



Hình 2.11: Kiến trúc đơn giản của Autoencoder

so với ở layer ngõ vào và ngõ ra. Trong trường hợp đó, giá trị h ở ngõ vào cần được nén lại như dữ liệu tại ngõ vào. Vì vậy, chúng ta có thể viết:

$$h = C(x). \tag{2.26}$$

Khi đó, neural network gồm layer ngõ vào và hidden layer gọi là encoder, gồm hidden layer và layer ngõ ra gọi là decoder, chúng ta viết:

$$\hat{x} = f(C(x)). \tag{2.27}$$

Để có thể thấy được cách mà encoder-decoder hoạt động, gọi tập dữ liệu là $D = {x^{(i)}}_{i=1}^{T}$, chúng ta viết:

$$\hat{x}^{(i)} = f(Cx^{(i)}). \tag{2.28}$$

Hàm lỗi thường được cho bởi L^2 error:

$$E = \frac{1}{2}\sum_i |\hat{x}^{(i)} - x^{(i)}|^2 = \frac{1}{2}\sum_i \sum_k |\hat{x}_k^{(i)} - x_k^{(i)}|, \tag{2.29}$$

Sau đó, chúng ta có thể dùng backpropagation để training như thông thường.

Những mô hình Autoencoder về cơ bản có 7 loại như sau: [30]

- Denoising Autoencoder
- Sparse Autoencoder
- Deep Autoencoder
- Contractive Autoencoder
- Undercomplete Autoencoder
- Concolutional Autoencoder
- Variational Autoencoder

Ứng dụng: [31]

- Giảm số chiều dữ liệu
- Phục hồi thông tin
- Phát hiện tương đồng
- Xử lý ảnh: tái tạo hình ảnh, tô màu hình ảnh, tạo ra ảnh với độ phân giải cao, ...
- Thay đổi đặc trưng và giảm nhiễu

2.4.3 Denoising Autoencoder

DAE được giới thiệu bởi Vincent (2008, 2010) và Glorot (2011), cung cấp một mạng cho phép tái tạo lại dữ liệu từ những điểm dữ liệu bị hư hại, nhiễu chính là loại hư hại được tác giả chọn để mô hình hóa.

Chúng ta có $x \in \mathbb{R}^d$ là ngõ vào của mạng. $y \in \mathbb{R}^{d'}$ là ngõ ra của mạng, có thể được tính toán đơn giản như sau: $f_\theta(x) = s(Wx + b)$, trong đó ma trận $W \in \mathbb{R}^{d' \times d}$ và vector $b \in \mathbb{R}^{d'}$ là tham số của mạng, s là hàm activation, thông thường là *sigmoid*. Chúng ta có thể viết, $\theta = (W, b)$, hàm f được gọi là *encoder*.

Phía *decoder* cũng tương tự như *encoder*, $g_{\theta'}(y) = t(W'y + b')$, trong đó $W \in \mathbb{R}^{d' \times d}$ và $b \in \mathbb{R}^d$. t là hàm activation thông thường thì nên khác với s. Thông thường W và W' bị ràng buộc bởi $W' = W^T$, vấn đề này được điều chỉnh lại trong định lý của Vincent (2011).

Trong suốt quá trình training, mục tiêu của chúng ta là học tham số encoder W và b, thêm vào đó chúng ta sẽ cần phải học tham số decoder b'. Chúng ta có thể làm được việc này bằng cách định nghĩa phân bố xác suất $p(\bar{x}|x, v)$, phần hư hại được điều khiển bởi a tham số v. Chúng ta train trọng số autoencoder để có thể tái tạo lại a tín hiệu đầu váo ngẫu nhiên từ phân bố của x có được từ phiên bản hư hại của nó $\bar{x}$, bằng cách chạy *encoder* và *decoder* tuần tự. Quá trình này có thể được mô tả bằng hàm mục tiêu:

$$\theta^*, \theta'^* = \underset{\theta,\theta'}{argmin} \mathbb{E}_{(X,\bar{X})}[L(X, g_{\theta'}(f_\theta(\bar{X}))] \tag{2.30}$$

Trong đó L là loss function như là squared error. Thông thường, chúng ta tối ưu hóa hàm mục tiêu bằng Stochastic Gradient Descent với mini-batches.

Có khá nhiều tham số tinh chỉnh ở đây như phân bố xác suất, chuyển đổi hàm s và t và loss function. Đối với loss function, cho trường hợp x liên tục, squared error có thể được sử dụng. Trong trường hợp x là nhị phân hoặc $x \in [0, 1]$, loss function thường được sử dụng là loss entropy:

$$L(x, z) = -\sum_{i=1}^{D}(x_i \log z_i + (1 - x_i) \log(1 - z_i)). \tag{2.31}$$

Đối với hàm activation, lựa chọn thông thường là sigmoid $s(v) = \frac{1}{1+e^{-v}}$ cho cả encoder và decoder hoặc có thể sử dụng rectifier $s(v) = max(0, v)$ cho phần kết nối giữa encoder và decoder.

Một trong những tham số quan trọng nhất cảu denoising autoencoder là phân bố xác suất p. Với tín hiệu x liên tục, nhiễu Gausisan $p(\bar{x}|x, v) = N(\bar{x}|x, v)$ có thể được sử dụng.

Đối với x là nhị phân hoặc $x \in [0, 1]$, nhiễu thường được sử dụng là masking noise, với mỗi $i \in 1, 2, ...d$, chúng ta lấy mẫu $\bar{x_i}$ độc lập.

$$p(\bar{x_i}|x_i, v) = \begin{cases} 0 & \text{with probability } \nu \\ x_i & \text{otherwise.} \end{cases} \tag{2.32}$$

Trong đó ν là mức nhiễu.

2.5 Biến đổi Wavelet liên tục

2.5.1 Giới thiệu

Trong xử lý tín hiệu, phép biến đổi Fourier là một công cụ quan trọng vì nó là cầu nối cho việc biểu diễn tín hiệu giữa miền không gian và miền tần số. Việc biểu diễn tín hiệu trong miền tấn số đôi khi có lợi hơn trong miền không gian. Tuy nhiên, biến đổi Fourier chỉ có thể cung cấp thông tin có tính toàn cục và chỉ thích hợp cho những tín hiệu tuần hoàn, không chứa các đột biến hoặc các thay đổi không dự báo được. Để khác phục được đặc điểm này, Gabor đã đã áp dụng phép biển đổi Short Time Fourier cho từng đoạn nhỏ của tín hiệu. Phép biển đổi này cho thấy mối liên hệ giữa không gian và tần số nhưng lại bị khống chế bởi nguyên lý bật định Heisenberg cho các thành phần tần số cao và thấp trong tín hiệu. Phép biến đổi wavelet và bước tiếp theo để khắc phục hạn chế này.

Năm 1975, Morlet, J., phát triển phương pháp đa phân giải. Trong đó, ông ta sử dụng một xung dao động, được hiểu là một "wavelet" cho thay đổi kích thước và so sánh với tín hiệu ở từng đoạn riêng biệt. Kỹ thuật này bắt đầu với sóng nhỏ (wavelet) chứa các dao động tần số khá thấp, sóng nhỏ này được so sánh với tín hiệu phân tích để có một bức tranh toàn cục của tín hiệu ở độ phân giải thô. Sau đó sóng nhỏ được nén lại để nâng cao dần tần số dao động. Quá trình này gọi là làm thay đổi tỉ lệ phân tích; khi thực hiện tiếp bước so sánh, tín hiệu sẽ được nghiên cứu chi tiết ở các độ phân giải cao hơn, giúp phát hiện các thành phần biến thiên nhanh còn ẩn bên trong tín hiệu.

2.5.2 Biến đổi wavelet thuận

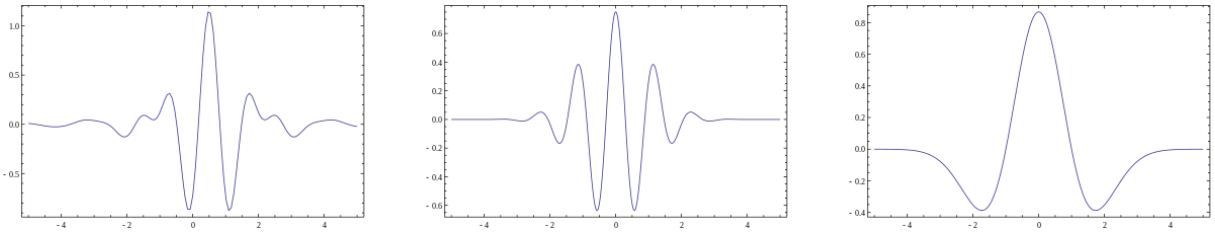
Gọi $f(x)$ là tín hiệu 1D, biến đổi wavelet liên tục của tín hiệu $f(x)$ sử dụng hàm wavelet Ψ_0 được biểu diễn bởi:

$$W(s, b) = \frac{1}{\sqrt{s}} \sum_{-\infty}^{+\infty} f(x) \Psi_0^*(\frac{x - b}{s}) dx \tag{2.33}$$

Trong đó:

- $W(s, b)$ là hệ số biến đổi wavelet liên tục của $f(x)$
- s là tỷ lệ (nghịch đảo của tần số)
- b là dịch chuyển đặc trưng của vị trí
- $\Psi_0^*(x)$ là hàm liên hiệp phức của của wavelet $\Psi_0(x)$.

Phép biến đổi wavelet có độ linh động cao so với biến đổi Fourier vì không nhất thiết phải dùng một hàm wavelet cố định, mà có thể lựa chọn hàm wavelet khác nhau cho bài toán thích hợp.



(a) Meyer (b) Morlet (c) Mexican hat

Hình 2.12: Một số mother wavelet thông dụng cho CWT

Độ rộng Quan hệ giữa độ rộng của hàm wavelet trong miền không gian và độ rộng trong miền tần số cho bởi nguyên lý bất định Heisenber. Nếu hàm wavelet bị hẹp về độ rộng trong miền không gian thì ngược lại, độ rộng của phổ tần số sẽ tăng lên. Vậy độ phân giải tối ưu trong miền tần số sẽ tương ứng với độ phân giải rất hạn chế trong miền không gian và ngược lại.

Rời rạc phép biến đổi wavelet liên tục: Để tính các hệ số của phép biến đổi wavelet liên tục trên máy tính, hai tham số tỉ lệ và tịnh tiến không thể nhận các giá trị liên tục mà nó phải là các giá trị rời rạc. Công thức rời rạc hóa phép biến đổi wavelet liên tục cho tín hiệu $f(n)$ một chiều được viết là:

$$W_{f(n)} = W(s,b) = \sum_n f(x)\frac{1}{\sqrt{s}}\Psi * (\frac{n-b}{s}) \tag{2.34}$$

Trong đó: s và b lần lượt là tham số tỉ lệ và dịch chuyển lấy giá trị rời rạc. Phép tổng hợp tín hiệu từ sự rời rạc hóa phép biến đổi wavelet liên tục được viết là:

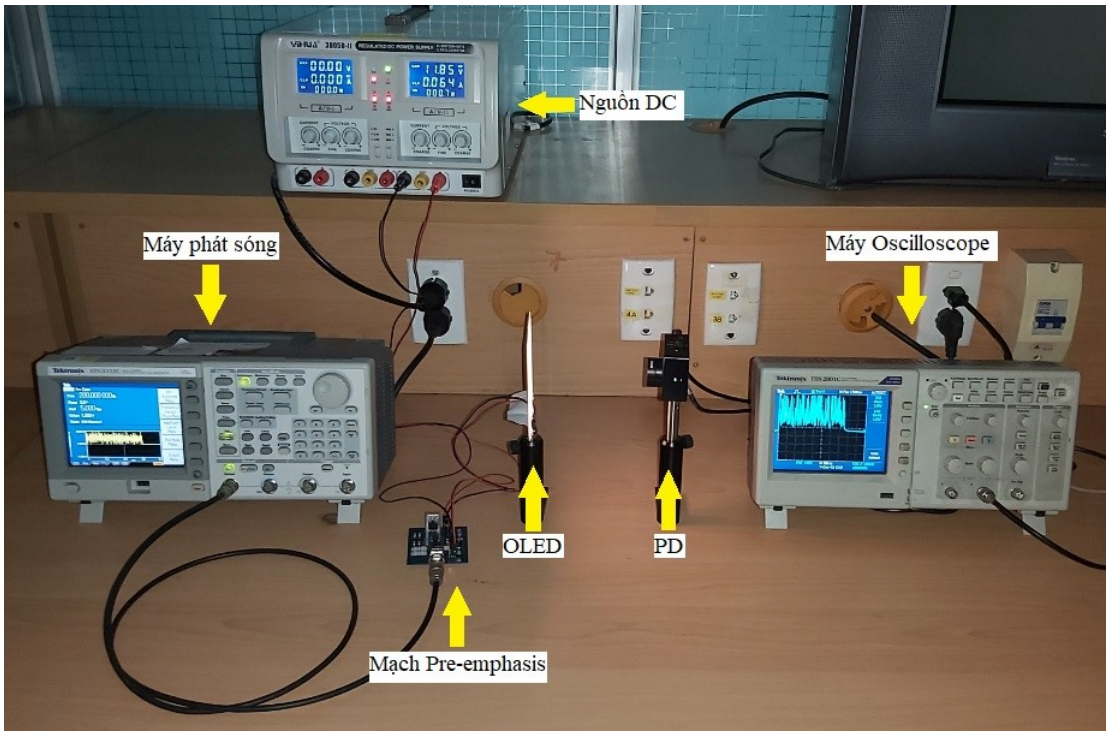
$$f(n) = c_g \sum_s \sum_b W(s,b)\Psi\frac{n-b}{s} \tag{2.35}$$

Với c_g là hằng số phụ thuộc vào hàm wavelet được sử dụng. Vì biểu thức phép biến đổi wavelet là một tích chập nên theo định lý tích chập, chúng ta có thể sử dụng phép biến đổi Fourier nhanh (FFT, Fast Fourier Transform) để tính phép biến đổi wavelet.

Chương 3. KẾT QUẢ NGHIÊN CỨU

3.1 Phương pháp tiếp cận

3.1.1 Hệ thống VLC thực tế



Hình 3.1: Hệ thống VLC thực tế tại phòng thí nghiệm 209B1

Điều kiện đo đạc thực tế: Hệ thống VLC được đặt tại phòng thí nghiệm 209B1, trường Đại học Bách Khoa thành phố Hồ Chí Minh. Nhiệt độ phòng thì nghiệm là 25^oC, có thể có hoặc không nhiễu từ đèn huỳnh quang, có ít nguồn sáng tự nhiên từ bên ngoài vào.

3.1.2 Dữ liệu

Tín hiệu phát Để giảm sự thay đổi tín hiệu đột ngột của tín hiệu NRZ, chúng tôi sử dụng xung Lorentz để thay thế.

Hàm Lorentzian với một đỉnh được định nghĩa bởi:

Bảng 3.1: Bảng các thành phần của hệ thống VLC thực tế

.

Thiết bị	Thông số
Generator Tektronik AFG 3152C	Tốc độ lấy mẫu tối đa 1GS/s, 2 kênh, băng thông 150MHz Có thể phát dạng sóng Sine, Square, Pulse, Ramp, Guassian,. . . . Điều chế AM, FM, PM, FSK, PWM Nguồn 100 - 240 V, 47 - 63 Hz
Oscilloscope Tektronik TDS 2024C	Tốc độ lấy mẫu tối đa 2GS/s Băng thông 200MHz Void/Div tối thiểu 2mV 4 Channels
OLED [32]	Băng thông 5KHz Kích thước: 100 $\times$100 $\times$ 0.88mm Công suất tiêu thụ: 1.28 W Độ sáng: 75 lm Dòng DC: 150 mA Điện áp: 8.5 V
Photodiode (PD) [33]	ADP410A/M của ThorLab Bước sóng từ 400 – 1000 nm Nguồn: $\pm$ 12 V, 250 mA
Nguồn DC Yihua 3005D	0-30 V*2 / 0-5 A*2 2.5 V / 3.3 V / 5 V * 3 A

$$L(x) = \frac{1}{\pi}\frac{\frac{1}{2}\Gamma}{(x-x_0)^2 + \frac{1}{2}\Gamma^2} \tag{3.1}$$

Trong đó x_0 là trung tâm, Γ là tham số được đặc trưng bởi chiều rộng có tên là Full Width at Half Maximum. Hàm Lorentz có phân bố xác suất theo phân bố Cauchy. Hàm Lorentz có biến đổi Fourier là

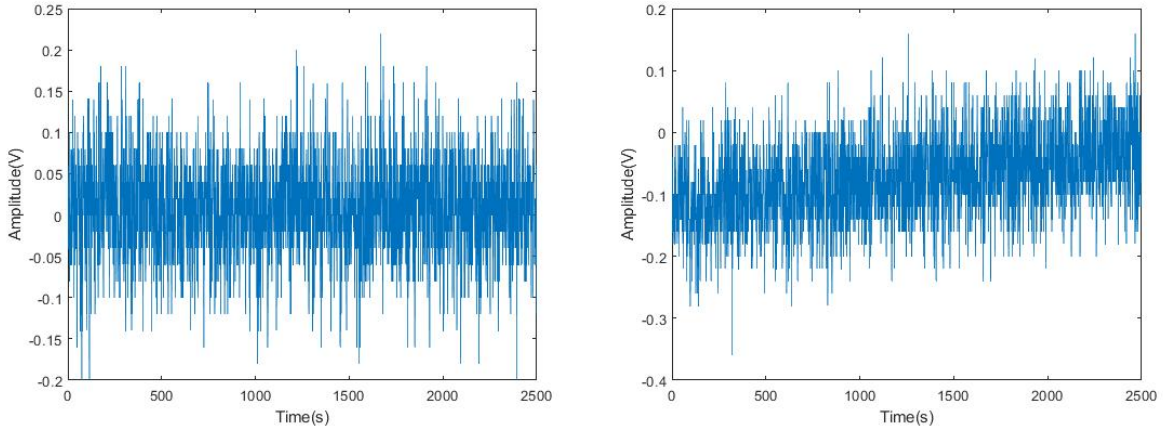
$$\mathcal{F}[\frac{1}{\pi}\frac{\frac{1}{2}\Gamma}{(x-x_0)^2 + (\frac{1}{2}\Gamma)^2}] = e^{-2\pi i k x_0 - \Gamma\pi|k|.} \tag{3.2}$$

Trích xuất dữ liệu: Tín hiệu truyền đi gồm có 250 ký hiệu, trong đó có 9 ký hiệu pilot dùng để nhận dạng điểm bắt đầu của chuỗi ký hiệu truyền đi. Sau khi thu được dữ liệu từ oscilloscope, chuỗi bit truyền đi được trích xuất lại bằng cách so sánh với với chuỗi bit truyền đi thông qua phép tính correlation.

Phân tích dữ liệu: Khi không dùng neural networks, việc giải điều chế sẽ dựa trên một giá trị ngưỡng có thể thay đổi được. Để ước lượng được SNR của tín hiệu, cần phải tính thông qua giản đồ mắt. Từ giá trị của SNR có thể tính ra được giá trị của BER.

Giá trị SNR được ước lượng qua công thức:

$$SNR = \frac{\frac{mean(HighLevel) - mean(LowLevel)}{\sigma(HighLevel) + \sigma(LowLevel)}}{M-1} \tag{3.3}$$



(a) Nhiễu nền trong điều kiện đo đạc của luận văn này

(b) Nhiễu nền trong điều kiện có bật thêm đèn huỳnh quang để chiếu sáng

Hình 3.2: Nhiễu nền trong hai điều kiện

Trong đó: giá trị mức cao và mức thấp được lấy tại thời điểm đồ thị mắt mở rộng nhất. Độ rộng của mắt được tính bằng $(mean(HighLevel) - 3\sigma(HighLevel)) - (mean(LowLevel) + 3\sigma(LowLevel))$.

Công thức trên được tính theo Volt, nên cần đổi ra dB bằng cách lấy $20\log_{10}$ kết quả ở trên.

Có 2 cách để ước lượng BER là ước lượng dựa trên SNR và lấy thương của tổng số bit bị sai và tổng số bit truyền đi. Có thể tính BER dựa trên SNR theo công thức: [34]

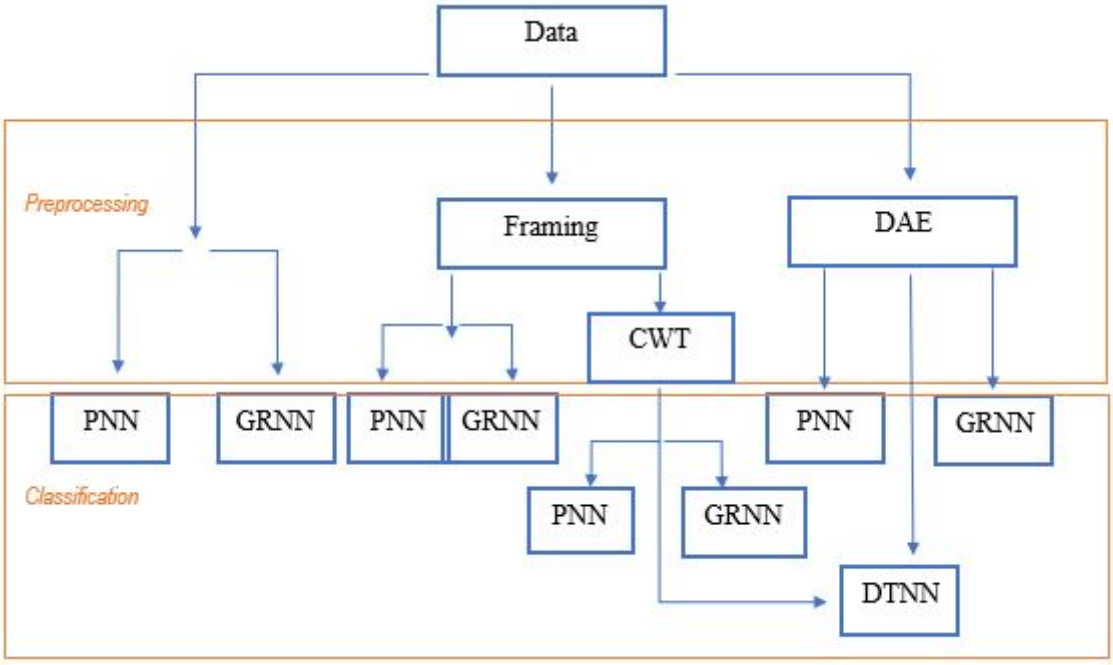
$$BER = \frac{M-1}{M} erfc(\sqrt{\frac{SNR}{2(M-1)}}) \tag{3.4}$$

Phương pháp đánh giá: Chất lượng của mô hình được đánh giá thông qua 2 thông số là SNR và BER được tính toán dựa trên 10000 bits thu được (bỏ qua vấn đề đồng bộ ở phía thu và phía phát). Sử dụng kỹ thuật K-fold cross validation để có thể test toàn bộ 10000 bits trên. Dữ liệu được chia ngẫu nhiên thành 5 nhóm, mỗi nhóm 2000 bits, sử dụng 4 nhóm cho việc train và 1 nhóm còn lại cho việc test. Lặp lại cho đến khi test đủ tất cả các nhóm.

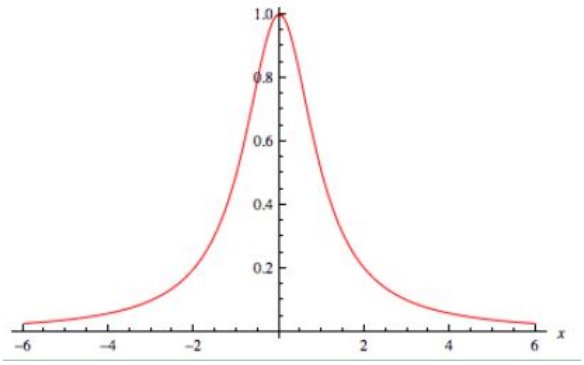
3.1.3 Tiền xử lý dữ liệu:

Framing Mỗi symbol đều phụ thuộc vào những symbols đứng trước và sau nó. Vì vậy khi trích dữ liệu để training, chúng ta cần chia dữ liệu thành frame, mỗi frame bao gồm 1 symbol thu được tương ứng với symbol truyền đi cùng với 2 symbols trước và 2 symbols phía sau nó. Như vậy, mỗi frame sẽ bao gồm 5 symbols, các frame liền nhau thì chồng lần lên nhau.

Đánh giá feature importance bằng Random Forest Tại mỗi bit rate, mối quan hệ giữa các symbols và những bit đứng trước và sau nó sẽ có sự khác nhau. Vì vậy, cần một phương pháp có thể đánh giá một cách tương đối mức độ quan trọng của các samples. Trong báo cáo này, chúng tôi sẽ sử dụng Random Forest để đánh giá feature importance.



Hình 3.3: Sơ đồ khối các phương pháp tiền xử lý và phân loại tín hiệu



Hình 3.4: Xung lorentz

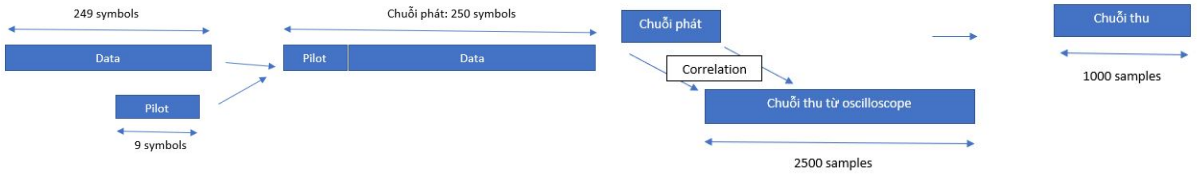
Áp dụng biến đổi wavelet liên tục Hệ thống VLC được sử dụng trong bài là một hệ thống phi tuyến và có nhiều nhiễu từ môi trường. Vì vậy, chúng ta cần một phương pháp để xử lý đối với vấn đề này. Các nghiên cứu trước đây đã chứng minh CWT là một phương pháp rất hữu dụng trong việc xử lý tín hiệu có nhiễu.

Mother wavelet được chọn là Mexican hat (Richer) vì nó có sự tương đồng nhiều nhất với tín hiệu.

$$\psi(t) = \frac{2}{\sqrt{3}\sqrt[4]{\pi}} \exp^{-\frac{t^2}{2}} \left(1 - t^2\right) \tag{3.5}$$

Bề rộng (width/scale) được chọn là 3. CWT chỉ cho kết quả rõ ràng khi tín hiệu có nhiều thông tin hơn. Vì vậy, trong báo cáo này, chúng tôi chỉ áp dụng CWT cho trường hợp tín hiệu trước đó đã được tiền xử lý bằng cách lấy theo frame.

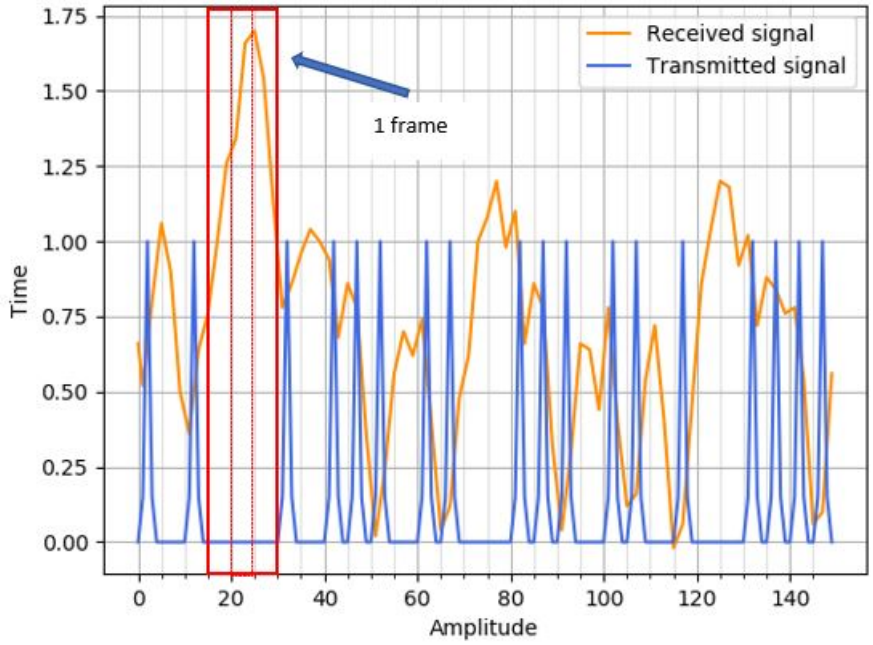
Hiệu chỉnh tín hiệu bằng DAE: Như đã trình bày ở chương trước, với giới hạn băng thông điều chế của OLED, chúng ta phải sử dụng thêm bộ pre-emphasis để làm tăng băng thông. Nhưng cũng chính vì điều này mà méo dạng phi tuyến cũng tăng theo. Trong



(a) Cấu tạo chuỗi dữ liệu

(b) Quá trình trích xuất dữ liệu tại phía thu

Hình 3.5: Trích xuất dữ liệu:



Hình 3.6: Cách một frame được trích ra từ tín hiệu thu

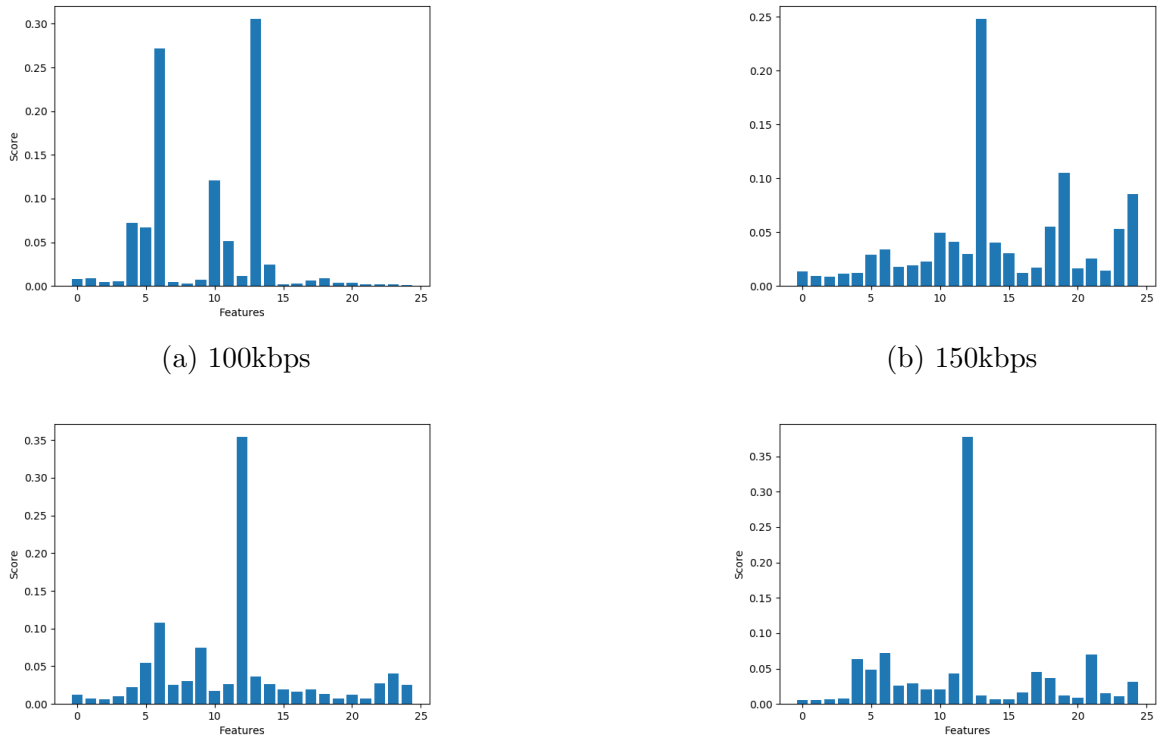
báo cáo này, chúng ta sẽ tập trung vào việc làm giảm sự méo dạng phi tuyến của OLED bằng bộ bù post-distortion.

Ngõ vào của mô hình là tín hiệu cùng loại với tín hiệu trước khi qua kênh truyền VLC. Mô hình sẽ học theo kiểu unsupervised learning, nghĩa là chỉ học những đặc tính của tín hiệu trước khi qua kênh truyền mà không cần phải dán nhãn. Tín hiệu sử dụng trong hệ thống VLC rất đơn giản nên việc bị overfitting là không thể tránh khỏi. Để giảm thiểu overfitting có thể sử dụng kỹ thuật noise injection để tăng độ phức tạm cho dữ liệu đầu vào.

Để tạo ra dữ liệu để train cho mô hình DAE, chúng ta cần tạo ra một chuỗi tín hiệu ngẫu nhiên có số lượng mẫu bằng với dữ liệu cần hiệu chỉnh. Ví tín hiệu tạo ra được rất đơn giản nên mô hình sẽ rất dễ bị overfitting, để giảm thiểu tình trạng này, cần áp dụng kỹ thuật noise injection để tăng độ phức tạp cho tín hiệu.

Như vậy, mô hình DAE sẽ sử dụng tín hiệu sau khi thêm nhiễu để học mà không cần dán nhãn cho tín hiệu. Mô hình sẽ học những đặc trưng của tín hiệu 2 mức khi không bị méo dạng. Sau khi mô hình DAE được training xong, tín hiệu ngõ ra của hệ thống VLC sẽ được đưa vào để có thể hiệu chỉnh lại tín hiệu bị méo dạng. (Vì nhiễu là ngẫu nhiên nên chất lượng của mô hình qua mỗi lần train sẽ cho kết quả có đôi chút khác biệt).

Để tăng hiệu quả cho mô hình DAE, chúng tôi xây dựng các layer dựa trên mô hình



(a) 100kbps

(b) 150kbps

(c) 200kbps

(d) 250kbps

Hình 3.7: Đánh giá dữ liệu bằng feature importance

Long Short Term Memory (LSTM). LSTM sẽ giúp mô hình DAE có thể học được những đặc điểm, mối quan hệ của những bit liền kề. (trong báo cáo này, chúng tôi sẽ không đề cập về cơ sở lý thuyết LSTM). Một số thông số cho mô hình DAE như:

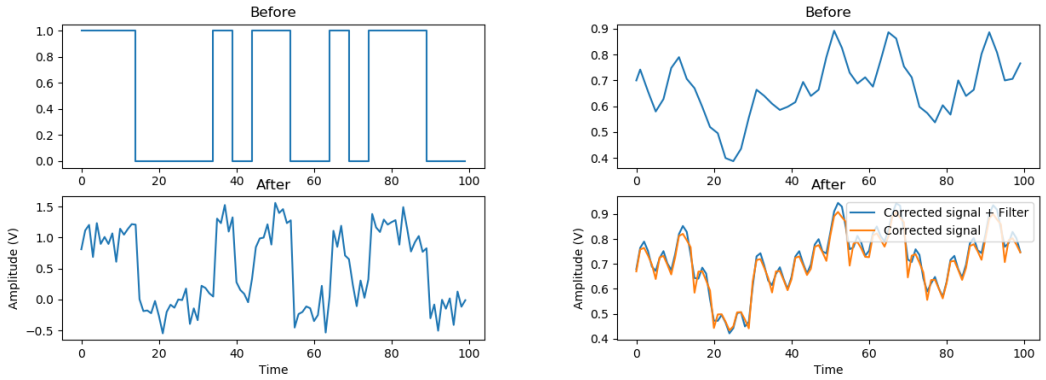
- Optimizer: Stochastic Gradient Descent with mini batch
- Learning rate: 0.001
- Epoch: 100

Tín hiệu sau khi đi qua mô hình DAE sẽ có khá nhiều nhiễu, vì vậy nó cần được đi qua thêm một bộ lọc để giảm bớt nhiễu. Có thể thấy tín hiệu sau khi đi qua DAE và bộ lọc có chất lượng tốt hơn nhiều so với tín hiệu ban đầu. Các mức của tín hiệu có thể dễ dàng phân biệt được, như vậy là đã làm giảm đáng kể tình trạng méo dạng của tín hiệu.

3.2 Kết quả và phân tích

3.2.1 Khảo sát tín hiệu:

Trong phần này, chúng tôi sẽ tiến hành khảo sát tín hiệu tại các tốc độ bit khác nhau để tìm ra tốc độ bit phù hợp để phân tích và so sánh tín hiệu NRZ với tín hiệu xung Lorentz. Để thấy hiệu quả của các phương pháp tiền xử lý, phân loại cũng như việc sử dụng xung Lorentz, chúng tôi sẽ so sánh các phương pháp này với nhau và với kết quả của những nghiên cứu trước đó cũng tại trường đại học Bách Khoa TP. HCM với cùng

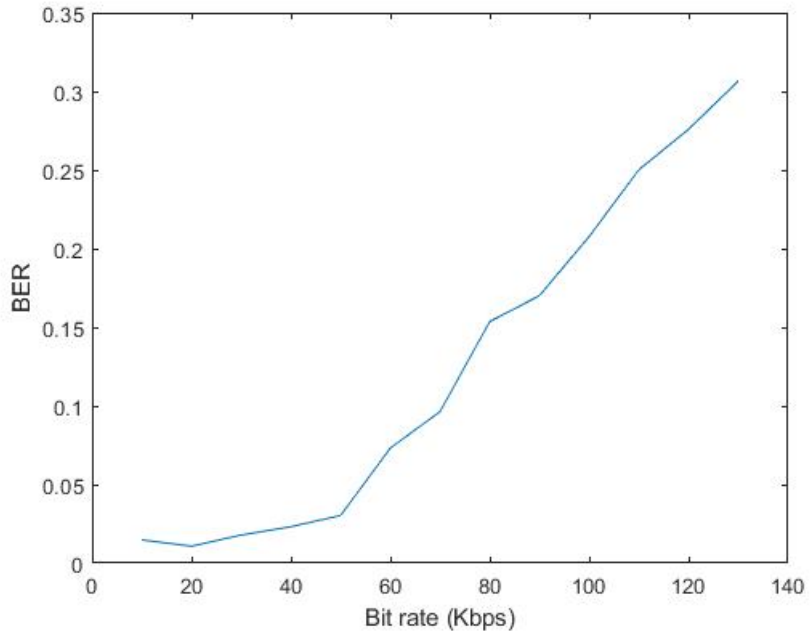


(a) Tín hiệu trước và sau khi thêm nhiễu (dùng cho training)

(b) Tín hiệu ngõ ra của hệ thống VLC trước và sau khi được hiệu chỉnh

Hình 3.8: Mô hình DAE

một hệ thống VLC. Trong đó, sử dụng điều chế NRZ 2 mức, phân loại bằng PNN, không qua tiền xử lý. Tốc độ tối đa đạt được là 25kbps khi ước lượng BER thông qua SNR và 53kbps khi dùng PNN để phân loại (không qua tiền xử lý). Ngoài ra chúng tôi còn khảo sát tác động từ nhiễu từ đèn huỳnh quang trong điều kiện trong nhà.



Hình 3.9: BER ước lượng tại các tốc độ bit khác nhau khi dùng xung Lorentz

Hình 4.1 cho thấy, nhìn chung việc dùng tín hiệu xung Lorentz ngõ vào làm cho tín hiệu ngõ ra khó nhận dạng hơn bằng cách ước lượng BER thông qua SNR, không có tốc độ bit nào đạt ngưỡng khi dùng Lorentz. Tuy nhiên nó lại giúp mạch đáp ứng tốt hơn, vẫn có thể phân biệt bằng mắt khi tốc độ vượt quá 70kbps, trong khi đó tín hiệu NRZ thì không còn phân biệt được nữa.

Vì vậy, để chứng minh hiệu quả của việc sử dụng xung Lorentz, trong luận văn này chúng tôi sẽ lựa chọn tốc độ bit là 100 kbps để tiến hành khảo sát, phân tích.

3.2.2 Phân loại các mức tín hiệu dùng neural networks:

Hai mô hình được sử dụng trong báo cáo này là PNN và GRNN, cả hai mô hình trên đều được phát triển bởi Specht và đều là mô hình one-pass. Với lợi thế là về mặt tốc độ cũng như khả năng chống chịu nhiễu. Để thể hiện được hiệu quả của 2 mô hình trên, chúng tôi sẽ về BER và thời gian (thực hiện trên Google Colab).

Đầu tiên, chúng tôi sẽ khảo sát dữ liệu khi chỉ dùng một symbol để phân loại. Khi xử lý theo kiểu này, DAE là một phương pháp rất hữu hiệu để cải thiện kết quả phân loại.

Độ lệch chuẩn được chọn cho phương pháp PNN khi chỉ lấy một symbol khá nhỏ (từ 0.05 đến 0.3) và tăng dần khi tăng khoảng cách. Đối với GRNN, độ lệch chuẩn lớn hơn nhiều lần so với PNN (từ 3 đến 7), và cần giảm dần khi tăng khoảng cách. Đối với trường hợp sử dụng DAE, độ lệch chuẩn cần nhỏ hơn từ 5 đến 10 lần với PNN và lớn hơn từ 5 đến 10 lần với GRNN.

Bảng 3.2: Bảng so sánh độ chính xác của các phương pháp khi chỉ lấy một symbol

.

Khoảng cách(cm) / Phương pháp	10	20	30	40
Không dùng neural networks	0.1928	0.2077	0.2006	0.2194
PNN	0.1377	0.1867	0.1744	0.2089
GRNN	0.1394	0.1861	0.1745	0.2090
PNN + DAE	0.0350	0.0544	0.0585	0.0999
GRNN + DAE	0.0347	0.0530	0.0566	0.0910

Thứ hai, nhận thấy các symbol nằm cạnh nhau có sự tương quan với nhau, chúng tôi đã áp dụng một kỹ thuật trong xử lý tiếng nói để xử lý tình trạng này. Độ lệch chuẩn cho PNN cao hơn và GRNN là thấp hơn đôi chút so với khi chỉ lấy một symbol. Khi lấy dữ liệu theo frame và áp dụng CWT, cần chọn độ lệch chuẩn trong hai phương pháp là PNN và GRNN cao hơn so với không dùng CWT, thông thường là gấp đôi.

Bảng 3.3: Bảng so sánh độ chính xác của các phương pháp khi lấy một frame

.

Khoảng cách(cm) / Phương pháp	10	20	30	40
PNN+ Framing	0.0048	0.0053	0.0123	0.0313
GRNN+ Framing	0.0050	0.0053	0.0127	0.0311
PNN + CWT	0.0054	0.0047	0.0118	0.0341
GRNN + CWT	0.0067	0.0088	0.0119	0.0338
DTNN	0.0037	0.0047	0.0126	0.0317

Từ kết quả thu được như trên, có thể thấy việc dùng DAE để bù méo dạng đem lại hiệu quả tốt nhất khi chỉ dùng một symbol để phân loại. Trong khi đó, framing mang lại hiệu quả đáng kể cho việc nhận dạng. Từ đó, ý tưởng của chúng tôi là kết hợp hai loại dữ liệu này lại với nhau và mô hình DTNN là lựa chọn vô cùng phù hợp trong trường hợp này. Một số thông số của mô hình DTNN được sử dụng trong luận văn:

- Tối ưu: Adam

- Số epochs: 30
- Loss function: Sparse categorical cross entropy

Bảng 3.4: Bảng so sánh thời gian train và test các phương pháp

Phương pháp	Thời gian train (s)	Thời gian test (s)
PNN	-	4.63
GRNN	-	19.53
DAE	103.25	1.9291
PNN + Framing	-	12.18
GRNN + Framing	-	35.11
PNN + CWT	-	47.37
GRNN + CWT	-	86.22
DTNN	-	63.51

Thời gian train và test của các phương pháp được thể hiện trong bảng 3.4, trong đó kết quả test trên Google Colab. Chỉ có mô hình DAE là có thể lưu lại mô hình và sử dụng cho nhiều bit rate và khoảng cách khác nhau. Vì sử dụng K-fold validation để đánh giá dữ liệu nên không thể lưu lại mô hình cho DTNN mà phải train lại cho mỗi lần cần test. Trong trường hợp có đủ dữ liệu (lớn hơn 20000 bits), có thể lưu lại mô hình DTNN, từ đó thời gian test có thể giảm đi khoảng 3-4 lần.

Kế đến, chúng tôi sẽ đánh giá nhanh qua hiệu quả của các phương pháp xử lý mới như bên trên và tác động của nhiễu từ đèn huỳnh quang.

- Với tín hiệu NRZ thu được từ học kỳ trước, phương pháp PNN kết hợp Framing có thể giúp giảm BER tại 60kbps xuống còn 0.0021. Từ đó giúp tăng tốc độ bit đối đa lên 7 kbps so với học kỳ trước.
- Trong điều kiện có bật đèn huỳnh quang khoảng cách 20 cm, tốc độ bit 100 kbps. BER đạt 0.0315, gấp 5 lần tại 20 cm, tương đương với 40 cm khi không có không bật đèn.

3.3 Kết luận chương

Trong báo cáo này, chúng tôi đã xây dựng các mô hình có khả năng phân loại các mức tín hiệu và các mô hình để tiền xử lý cho tín hiệu. Để thấy được sự hiệu quả, chúng tôi đã so sánh các phương pháp này với nhau và với kết quả từ nghiên cứu trước đó.

Đầu tiên, chúng tôi so sánh các phương pháp tiền xử lý tín hiệu. Từ bảng 3.2 có thể thấy việc phân loại tín hiệu bằng neural networks (không qua tiền xử lý) vẫn đem lại hiệu quả đôi chút so với khi không dùng. Tuy nhiên sai số vẫn còn rất lớn và không thể ứng dụng cho mô hình thực tế. Sau đó, sau khi dùng thêm DAE để hiệu chỉnh tín hiệu BER đã giảm đáng kể, giảm 4 lần. Mặc dù vậy, BER vẫn còn khá cao, chưa thể áp dụng vào hệ thống VLC được. Việc hiệu quả của DAE chưa cao như kỳ vọng là do ngay từ đầu tín hiệu truyền đi không phải là xung vuông, trong khi đó nhiệm vụ của DAE là đưa tín hiệu về càng giống tín hiệu NRZ càng tốt.

Nhận thấy các bit liền kề có sự liên hệ với nhau, chúng tôi đã ứng dụng một kỹ thuật của xử lý tiếng nói là Framing để tăng độ chính xác cho mô hình. Từ bảng 3.3, có thể thấy Framing giúp BER giảm đi 20 lần xuống dưới mức 0.01 tại các khoảng cách 10 và 20 cm. Ngoài ra, chúng tôi còn áp dụng thêm một kỹ thuật khác là CWT, kết quả cho thấy BER giảm đi đôi chút. Từ kết quả của Framing và DAE, chúng tôi đã sử dụng DTNN để giảm BER, kết quả là DTNN cũng giúp giảm BER, đặc biệt là ở 10 cm khi BER nhỏ hơn mức ngưỡng cho phép của hệ thống VLC là 3.8e -3.

Kế đến, chúng tôi so sánh ba phương pháp được sử dụng trong luận văn là PNN, GRNN và DTNN. Từ ba bảng 3.2, 3.3 và 3.4 cho thấy PNN và GRNN mang lại hiệu quả tương đối ngang nhau, DTNN thì nhỉnh hơn đôi chút. Tuy nhiên, thời gian chính là ưu thế rất lớn của PNN so với hai phương pháp còn lại. Ngoài ra, xét về tính khả thi khi triển khai mô hình thực tế, PNN và GRNN cũng chiếm phần lợi thế hơn so với DTNN vì chỉ sử dụng các phép tính toàn đơn giản, hoàn toàn có thể lập trình bằng C/C++ trên những vi xử lý giá rẻ, không cần lập trình Python trên các vi xử lý đắt tiền hơn như DTNN.

Tóm lại, qua khảo sát ở trên có thể thấy xung Lorentz làm tín hiệu trở nên khó phân loại hơn so với NRZ. Tuy nhiên, nó lại giúp mạch đáp ứng tốt hơn. Từ đó, khi kết hợp Lorentz cùng với những phương pháp mạnh mẽ hơn có thể mang lại hiệu quả đáng kể. Khảo sát của chúng tôi cho thấy, phương pháp của chúng tôi có thể giúp bit rate tăng 4 lần so với phương pháp truyền thống và tăng gần 2 lần so với khi áp dụng tín hiệu NRZ cùng với những phương pháp xử lý đơn giản.

Chương 4. KẾT LUẬN

4.1 Tóm tắt và kết luận chung

4.1.1 Ưu điểm

- Xây dựng thành công 3 mô hình có thể phân loại được 2 mức của tín hiệu.
- So sánh 2 phương pháp phân loại tín hiệu là PNN và GRNN với phương pháp thường được sử dụng là CNN
- Xây dựng mô hình có khả năng cải thiện tín hiệu bị méo dạng.
- Đề xuất phương pháp xử lý tín hiệu bằng cách tách dữ liệu theo frame.
- Đề xuất sử dụng xung Lorentz thay thế cho tín hiệu NRZ.

4.1.2 Nhược điểm

- Tín hiệu chỉ có 2 mức
- Số tốc độ bit và khoảng cách được khảo sát ít.

4.1.3 Nguyên nhân chưa đạt

- Chưa giành đủ thời gian nghiên cứu, thí nghiệm.
- Chưa có đủ kiến thức về generative model.

4.1.4 Cách khắc phục

- Khảo sát thêm các nhiều tốc độ bit ở những khoảng cách khác nhau.
- Học tập và nghiên cứu sâu về AI và generative model.
- Thay đổi mạch lái OLED

4.2 Hướng phát triển

- Xây dựng mô hình bù méo dạng tốt hơn: có thể ứng dụng những phương pháp như Generative adversarial networks (GAN), Adversarial Denoising Autoencoder, ...
- Kết hợp sức mạch của nhiều mô hình lại với nhau thông qua mô hình Stacking.

- Phát triển hệ thống thành MINO
- Thiết kế mạch lái OLED phù hợp.

TÀI LIỆU THAM KHẢO

[1] F. Ahmed, S. Ali, and M. Jawaid, "A review of modulation schemes for visible light communication," *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND NETWORK SECURITY*, vol. 18, pp. 117–125, 02 2018.

[2] P. Q. Thai, F. Rottenberg, P. T. Dat, and S. Shigeru, "Increase data rate of oled vlc system using pre-emphasis circuit and fbmc modulation," in *Imaging and Applied Optics 2018 (3D, AO, AIO, COSI, DH, IS, LACSEA, LS&C, MATH, pcAOP)*. Optical Society of America, 2018, p. SM2H.4. [Online]. Available: http://www.osapublishing.org/abstract.cfm?URI=LSC-2018-SM2H.4

[3] D. Yu, L. Deng, and F. Seide, "Large vocabulary speech recognition using deep tensor neural networks," in *INTERSPEECH*, 2012.

[4] (2020) The CISCO website. [Online]. Available: https://www.cisco.com/c/m/en_us/solutions/service-provider/forecast-highlights-mobile.html

[5] (2016) Introduction to oled lighting and key challenges for the industry. [Online]. Available: https://www.oledworks.com/wp-content/uploads/2017/05/OLED-Lighting-Berlin-Masterclass.pdf

[6] (2020) The CISCO website. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[7] S. Shao, A. Khreishah, M. Ayyash, M. B. Rahaim, H. Elgala, V. Jungnickel, D. Schulz, and T. D. Little, "Design and analysis of a visible-light-communication enhanced wifi system," *ECCS*, 2015.

[8] A. Cailean, B. Cagneau, L. Chassagne, V. Popa, and M. Dimian, "Evaluation of the noise effects on visible light communications using manchester and miller coding," *International Conference on Development and Application Systems (DAS)*, pp. 85–89, 2014.

[9] (9-2020) The Allied Market Research website. [Online]. Available: https://www.alliedmarketresearch.com/solid-state-lighting-system-market

[10] (1-2021) The Bussinesswire website. [Online]. Available: https://www.businesswire.com/news/home/20210114005882/en/291.1-Million-OLED-Lighting-Panels-Markets---Global-Trajectory-Analytics-to-2027---ResearchAndMcom

[11] H. Chen, Z. Xu, Q. Gao, and S. Li, "A 51.6 mb/s experimental vlc system using a monochromic organic led," *IEEE Photonics Journal*, vol. 10, no. 2, pp. 1–12, 2018.

[12] P. A. Haigh, Z. Ghassemlooy, and I. Papakonstantinou, "1.4-mb/s white organic led transmission system using discrete multitone modulation," *IEEE Photonics Technology Letters*, vol. 25, no. 6, pp. 615–618, 2013.

[13] P. A. Haigh, Z. Ghassemlooy, I. Papakonstantinou, and H. Le Minh, "2.7 mb/s with a 93-khz white organic light emitting diode and real time ann equalizer," *IEEE Photonics Technology Letters*, vol. 25, no. 17, pp. 1687–1690, 2013.

[14] P. Q. Thai, "Real-time 138-kb/s transmission using oled with 7-khz modulation bandwidth," *IEEE Photonics Technology Letters*, vol. 27, no. 24, pp. 2571–2574, 2015.

[15] Y. H. Hu and J.-N. Hwang, *Handbook of neural network signal processing.* 2000 N.W. Corporate Blvd., Boca Raton, Florida: CRC Press LLC, 2002.

[16] H. Nishizaki and K. Makino, "Signal classification using deep learning," in *2019 IEEE International Conference on Sensors and Nanotechnology*, 2019, pp. 1–4.

[17] G. Xu, X. Shen, S. Chen, Y. Zong, C. Zhang, H. Yue, M. Liu, F. Chen, and W. Che, "A deep transfer convolutional neural network framework for eeg signal classification," *IEEE Access*, vol. 7, pp. 112 767–112 776, 2019.

[18] A. Ochoa, L. J. Mena, and V. G. Felix, "oise-tolerant neural network approach for electrocardiogram signal classification," *2017 International Conference on Compute and Data Analysis*, 2017.

[19] M. Onder, A. Akan, and H. Dogan, "Neural network based receiver design for software defined radio over unknown channels," 11 2013, pp. 297–300.

[20] X. Lin, R. Liu, H. Wenmei, Y. Li, X. Zhou, and X. He, "A deep convolutional network demodulator for mixed signals with different modulation types," 11 2017, pp. 893–896.

[21] S. Ma, J. Dai, S. Lu, H. Li, H. Zhang, C. Du, and S. Li, "Signal demodulation with machine learning methods for physical layer visible light communications: Prototype platform, open dataset, and algorithms," *IEEE Access*, vol. 7, pp. 30 588–30 598, 2019.

[22] V. H. Tiệp, *Machine learning cơ bản*, Hà Nội, Việt Nam, 2020.

[23] B. Mohebali, A. Tahmassebi, A. Meyer-Baese, and A. H. Gandomi, *Chapter 14 - Handbook of Probabilistic Models.* Oxford OX5 1GB, United Kingdom and Cambridge, MA 02139, United States: Elsevier Inc., 2002.

[24] F. Ancona, A. Colla, S. Rovetta, and R. Zunino, "Implementing probabilistic neural networks." *Neural Computing and Applications*, vol. 5, pp. 152–159, 09 1997.

[25] D. F. Specht, "Probabilistic neural networks," *Neural networks*, vol. 3, pp. 109–118, 1990.

[26] (2019) The Github website. [Online]. Available: https://github.com/muhendis/Generalized-regression-neural-networks-library-from-scratch/blob/master/GeneralizedRegressionNeuralNetworks/GeneralizedRegressionNeuralNetworks.py

[27] N. Tawara, T. Kobayashi, M. Fujieda, K. Katagiri, T. Yazu, and T. Ogawa, "Adversarial autoencoder for reducing nonlinear distortion," in *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2018, pp. 1669–1673.

[28] X. Lu, Y. Tsao, S. Matsuda, and C. Hori, "Speech enhancement based on deep denoising autoencoder," in *INTERSPEECH*, 2013.

[29] H. I. Choi, "Lecture on machine learning, autoencoders," Fall 2017.

[30] (2020) The Opengenus website. [Online]. Available: https://iq.opengenus.org/types-of-autoencoder/

[31] (2020) The Wikipedia website. [Online]. Available: https://en.wikipedia.org/wiki/Autoencoder

[32] *OLED panel*, LG Chem Ltd., 9 2015.

[33] *Photodetector*, Thorlabs, 9 2020.

[34] W. Apena, "Performance evaluation of pam-4 and pam-2 modulation techniques using matlab," *European Journal of Engineering Research and Science*, vol. 3, no. 5, pp. 36–40, May 2018. [Online]. Available: https://www.ejers.org/index.php/ejers/article/view/679

PHỤ LỤC

Ngôn ngữ lập trình: Python, Matlab.

Thư viện Python hỗ trợ:

- Tensorflow (2.4.1)
- Keras (2.2.5)
- Matplotlib (3.1.0)
- Numpy (1.19.5)
- Pandas (0.24.2)
- Statsmodels (0.11.1)
- Scikit learn (0.23.1)
- Scipy (1.5.2)
- Pywavelet (1.1.1)
- Streamlit (0.75.0)

Code: Google Drive Hoặc Github
Ghi chú:

- Configure.json: Cấu hình các phương pháp xử lý, khoảng cách, bitrate, mô hình, ...
- Main.py: file thực thi chính cho chương trình
- Getdata.py: trích xuất và xử lý dữ liệu
- Model.py: lưu trữ các mô hình như PNN, GRNN, DTNN, DAE.

Hướng dẫn sử dụng chi tiết (readme.txt) cũng như các file configure.json, default-std-PNN.json, default-std-GRNN.json có thể được tìm thấy tại các đường dẫn phía trên.

Main.py

```
----------------------
Bit_rate:  100
Distance:  20
Model:  DTNN
Preprocessing:  related_bit
Enable CWT: True
----------------------
2021-07-26 19:36:35.760670: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not creating XLA devices, tf_xla_enable_xla_device
s not set
2021-07-26 19:36:35.762778: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'nv
cuda.dll'; dlerror: nvcuda.dll not found
2021-07-26 19:36:35.763286: W tensorflow/stream_executor/cuda/cuda_driver.cc:326] failed call to cuInit: UNKNOWN ERROR (303)
2021-07-26 19:36:35.768608: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic information
for host: DESKTOP-NUL5C86
2021-07-26 19:36:35.770998: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: DESKTOP-NUL5C86
2021-07-26 19:36:35.771851: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneA
PI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-07-26 19:36:35.773661: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not creating XLA devices, tf_xla_enable_xla_device
s not set
2021-07-26 19:36:35.951960: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization pass
es are enabled (registered 2)
Accuracy:  0.9919999837875366
Accuracy:  0.984000027179718
Accuracy:  0.9800000190734863
Accuracy:  0.9869999885559082
Accuracy:  0.9850000143051147
Accuracy:  0.9959999918937683
Accuracy:  0.9950000047683716
Accuracy:  0.9860000014305115
Accuracy:  0.9980000257492065
Accuracy:  0.9610000252723694
--------------------
BER:  0.013599991798400879
--- 87.24133586883545 seconds ---          & c:/users/hp/appdata/local/programs/python/python37/python.exe "e:/Project/Thesi
```

Hình 4.1: Kết quả thu được

```python
import numpy as np
import pandas as pd
import json
import Getdata
import Model
from sklearn.model_selection import KFold
from sklearn import metrics
import time
def read_configure_file():
        #read configure file
    start_time = time.time()
    f = open("configure.json")
    configure = json.load(f)
    mode_1 = configure["preprocessing"]
    if mode_1 == "Framing":
        mode_1 = "related_bit"
    if configure["enableCWT"] == "true":
        mode_2 = "CWT"
    else:
        mode_2 = ""
    bit_rate = configure["bitrate"]
    distance = configure["distance"]
    model = configure["model"]
    f.close()
    return model, bit_rate, distance, mode_1, mode_2
def get_std(model, bit_rate, distance, mode_1, mode_2):
    #get std: for PNN and GRNN
    if model == "PNN":
```

```
29          f = open("default_std_PNN.json")
30      else:
31          f = open("default_std_GRNN.json")
32      data = json.load(f)
33      if bit_rate == 100 or bit_rate == 200:
34          std = data[mode_1][str(bit_rate)][str(distance)]
35      else:
36          std = data[mode_1][str(bit_rate)]
37      if mode_2 == "CWT":
38          std = std*2
39      f.close()
40      return std
41  def get_data(model, bit_rate, distance, mode_1, mode_2):
42      if model == "DTNN":
43          try:
44              file_name_1 = "Lorentz_dataset/Lorentz_%scm_%sk_%s%s.csv" %
    (distance, bit_rate, "DAE", mode_2)
45              file_name_2 = "Lorentz_dataset/Lorentz_%scm_%sk_%s%s.csv" %
    (distance, bit_rate, "related_bit", mode_2)
46              file_name_3 = "Lorentz_dataset/Lorentz_label_%scm_%sk.csv" %
     (distance, bit_rate)
47              X1 = pd.read_csv(file_name_1, header=None)
48              X2 = pd.read_csv(file_name_2, header=None)
49              y = pd.read_csv(file_name_3, header=None)
50              X1 = X1.values
51              X2 = X2.values
52              y = y.values
53          except:
54              X1, y = Getdata.exe("DAE", bit_rate, distance)
55              X2, y = Getdata.exe("Framing", bit_rate, distance)
56              X1 = Getdata.CWT(X1)
57              X2 = Getdata.CWT(X2)
58          X_1 = np.arange(0, X1.shape[0]*X1.shape[1], 1)
59          X_2 = np.arange(0, X2.shape[0]*X2.shape[1], 1)
60          X1 = X1.reshape(-1)
61          X1 = np.interp(X_2, X_1, X1)
62          X1 = X1.reshape([X2.shape[0], X2.shape[1]])
63          X = np.hstack([X1, X2])
64          X = X.reshape([X2.shape[0], 2, X2.shape[1]])
65      else:
66          try:
67              filename_1 = "Lorentz_dataset/Lorentz_%scm_%sk_%s%s.csv" % (
    distance, bit_rate, mode_1, mode_2)
68              filename_2 = "Lorentz_dataset/Lorentz_label_%scm_%sk.csv" %
    (distance, bit_rate)
69              X = pd.read_csv(filename_1, header = None)
70              y = pd.read_csv(filename_2, header = None)
71              X = X.values
72              y = y.values
73          except:
74              X, y = Getdata.exe(mode_1, bit_rate, distance)
75              if mode_2 == "CWT":
76                  X = Getdata.CWT(X)
77      y = y.reshape([y.shape[0], ])
78      return X, y
79  def main():
80      start_time = time.time()
81      model, bit_rate, distance, mode_1, mode_2 = read_configure_file()
```

```
    if model == "PNN" or model == "GRNN":
        std = get_std(model, bit_rate, distance, mode_1, mode_2)
    X, y = get_data(model, bit_rate, distance, mode_1, mode_2)
    print("----------------------")
    print("Bit_rate: ", bit_rate)
    print("Distance: ", distance)
    print("Model: ", model)
    print("Preprocessing: ", mode_1)
    if mode_2 == "CWT":
        print("Enable CWT: True")
    else:
        print("Enable CWT: False")
    print("----------------------")
    ber = []
    kf = KFold(n_splits=10)
    for train_index, test_index in kf.split(X):
        #print("TRAIN:", train_index, "TEST:", test_index)
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        if model == "DTNN":
            m = Model.DTNN(X_train, y_train, X_test, y_test)
        elif model == "GRNN":
            m = Model.GRNN(X_train, y_train, X_test, y_test, std)
        else:
            m = Model.PNN(X_train, y_train, X_test, y_test, std)
        if model == "DTNN":
            score = m.predict()
        else:
            y_predicted = m.predict()
            score = metrics.accuracy_score(y_test, y_predicted)
        print("Accuracy: ", score)
        ber = np.append(ber, 1-score)
    print("--------------------")
    print("BER: ", np.mean(ber))
    print("--- %s seconds ---" % (time.time() - start_time))
if __name__ == '__main__':
    main()
```

Model.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import metrics
from numpy.core.umath_tests import inner1d
import math
from sklearn import metrics
import tensorflow as tf
import random
from sklearn.preprocessing import MinMaxScaler
from os import path
import scipy as sp
from scipy import signal
class GRNN:
    def __init__(self, x_train, y_train, x_test, y_test, std):
        self.x_train= x_train
        self.y_train= y_train
        self.x_test= x_test
        self.y_test= y_test
```

```
        self.std = np.full((1, self.y_train.size), std)
    def activation_func(self, distances): # gaussian kernel
        kernel = np.exp(- (distances) / 2*(self.std**2))
        return kernel
    def output(self,i):#sometimes called weight
        distances = np.sqrt(np.sum((self.x_test[i]-self.x_train)**2,axis=1))# euclidean distance
        return self.activation_func(distances)
    def denominator(self,i):
        return np.sum(self.output(i))
    def numerator(self,i):
        return np.sum(self.output(i) * self.y_train)
    def predict(self):
        predict_array = []
        for i in range(self.y_test.size):
            if self.numerator(i) == 0 and self.denominator(i) == 0:
                predict = 1
            else:
                predict=np.array([self.numerator(i)/self.denominator(i)])
            predict_array=np.append(predict_array,predict)
            predict_array = predict_array.astype(int)
        return predict_array*2
class PNN:
    def __init__(self, x_train, y_train, x_test, y_test, std):
        self.x_train = x_train
        self.y_train = y_train
        self.x_test = x_test
        self.y_test = y_test
        self.std = std
    def activation_func(self, distances): #kernel
        kernel = np.exp((-distances)/(2*self.std**2))   # for dot product
        return np.sum(kernel)
    def output(self, my_class, i):
        distances = np.subtract(my_class, self.x_test[i])
        # distances = distances * signal.windows.triang(25)
        distances = inner1d(distances, distances) #dot product
        return self.activation_func(distances)
    def pdf(self, my_class, i):
        pdf = (1/(math.sqrt(2*math.pi)*self.std))*(1/self.y_test.size)*(self.output(my_class, i))
        return pdf
    def predict(self):
        label_1 = 0
        label_0 = 0
        predict_array = []
        class_0 = []
        class_1 = []
        a = self.x_train.shape
        for j in np.arange(len(self.y_train)):
            if self.y_train[j] == 1:
                label_1+=1
                class_1 = np.append(class_1, self.x_train[j])
            else:
                label_0+=1
                class_0 = np.append(class_0, self.x_train[j])
        class_0 = np.reshape(class_0, [int(len(class_0)/a[1]), a[1]])
```

```
74          class_1 = np.reshape(class_1, [int(len(class_1)/a[1]), a[1]])
75          p_prior_0 = label_0 / len(self.y_train)
76          p_prior_1 = label_1 / len(self.y_train)
77          for i in range(self.y_test.size):
78              c_0 = p_prior_0 * self.pdf(class_0, i)
79              c_1 = p_prior_1 * self.pdf(class_1, i)
80              c = np.argmax(np.array([c_0, c_1]))
81              predict_array = np.append(predict_array, c)
82          return predict_array
83  class DAE:
84      def __init__(self, bit_rate, X, y):
85          self.bit_rate = bit_rate
86          self.num = 1500
87          self.z_dim = 5
88          self.noise_level = 0.2
89          self.X_test = X
90          self.y = y
91      def scale(self, X):
92          s = MinMaxScaler()
93          X = X.reshape(20*self.z_dim, 1)
94          X = s.fit_transform(X)
95          X = X.reshape(20, 1, self.z_dim)
96          return X
97      def getDeepAE(self):
98          # input layer
99          input_layer = tf.keras.layers.Input(shape=(1, self.z_dim))
100         encode_layer = tf.keras.layers.LSTM(20, activation='relu',
    return_sequences = True, kernel_regularizer=tf.keras.regularizers.L1
    (0.0))(input_layer)
101         encode_layer = tf.keras.layers.LSTM(self.z_dim, activation='relu
    ', return_sequences = False)(encode_layer)
102         latent_view = tf.keras.layers.RepeatVector(self.z_dim)(
    encode_layer)
103         decode_layer = tf.keras.layers.LSTM(self.z_dim, activation='relu
    ', return_sequences = True)(latent_view)
104         decode_layer = tf.keras.layers.LSTM(20, activation='relu',
    return_sequences = True)(decode_layer)
105         # decode_layer = tf.keras.layers.Dropout(0.2)(decode_layer)
106         output_layer = tf.keras.layers.TimeDistributed(tf.keras.layers.
    Dense(1))(decode_layer)
107         # model
108         model = tf.keras.Model(input_layer, output_layer)
109         return model
110     def correct(self):
111         autoencoder = self.getDeepAE()
112         optimizer = tf.keras.optimizers.SGD(lr=0.001, momentum = 0.9,
    nesterov=False)
113         loss = tf.keras.losses.MeanSquaredError()
114         autoencoder.compile(optimizer = optimizer, loss = loss)
115         self.X_test = self.X_test.reshape([self.X_test.shape[0]*self.
    z_dim, 1])
116         scale_1 = MinMaxScaler()
117         self.X_test = scale_1.fit_transform(self.X_test)
118         self.X_test = self.X_test.reshape([int(self.X_test.shape[0]/self
    .z_dim), 1, self.z_dim])
119         bit_0 = np.zeros(self.z_dim)
120         bit_1 = np.full((self.z_dim, ), 1)
121         X_train = []
```

```
        for i in range(self.num):
            a = random.randint(0, 1)
            if a == 0:
                X_train = np.append(X_train, bit_0 + np.random.randn()/5)
            else:
                X_train = np.append(X_train, bit_1 + np.random.randn()/5)
        noise = np.random.normal(0, self.noise_level, self.num*self.z_dim)
        X_train_noisy = X_train + noise
        X_train_noisy = np.reshape(X_train_noisy, [self.num, 1, self.z_dim])
        X_train = np.reshape(X_train, [self.num, 1, self.z_dim])
        X_val = self.X_test[:20]
        X_val = X_val.reshape(X_val.shape[0], 1, self.z_dim)
        X_val_true = np.kron(self.y[:20], np.full((self.z_dim, ), 1))
        X_val_true = X_val_true.reshape(20, 1, self.z_dim)
        if path.exists("model_ae.h5"):
            h = autoencoder.load_weights("model_ae.h5")
        else:
            h = autoencoder.fit(X_train_noisy, X_train, epochs=100, batch_size=5, shuffle=True, validation_data=(self.scale(X_val), X_val_true), verbose = 0)
            autoencoder.save("model_ae.h5")
        X_pred = autoencoder.predict(self.X_test)
        X_pred = X_pred.reshape(-1)
        b, a = sp.signal.butter(3, 0.6)
        filt = sp.signal.filtfilt(b, a, X_pred)
        filt = filt.reshape([self.X_test.shape[0], self.z_dim])
        return filt
class DTNN:
    def __init__(self, x_train, y_train, x_test, y_test):
        self.x_train = x_train
        self.y_train = y_train
        self.x_test = x_test
        self.y_test = y_test
    def DTNN_model(self):
        inputs = tf.keras.Input(shape = (2, 75))
        Layer1 = tf.keras.layers.Dense(75, activation = 'relu')(inputs)
        split0, split1 = tf.split(Layer1, num_or_size_splits = 2, axis = 1)
        H1 = tf.keras.layers.Dense(3, activation='relu')(split0)
        H2 = tf.keras.layers.Dense(3, activation='relu')(split1)
        V = tf.linalg.cross(H1, H2)
        Layer2 = tf.keras.layers.Dense(10, activation = 'relu')(V)
        outputs = tf.keras.layers.Dense(2, activation = 'sigmoid')(Layer2)
        model = tf.keras.Model(inputs = inputs, outputs = outputs)
        return model
    def predict(self):
        num_epochs = 25
        optimizers = tf.keras.optimizers.Adam(learning_rate=0.001)
        loss = tf.keras.losses.SparseCategoricalCrossentropy()
        model = self.DTNN_model()
        model.compile(loss = loss, optimizer = optimizers, metrics = ['accuracy'])
        x_train, x_val, y_train, y_val = train_test_split(self.x_train,
```

```
    self.y_train, test_size=0.2, random_state=0)
171         h = model.fit(x_train, y_train, validation_data =(x_val, y_val),
     batch_size=25, epochs=num_epochs, verbose=0)
172         score = model.evaluate(self.x_test, self.y_test, verbose = 0)
173         return score[1]
```

Getdata.py

```
1  import numpy as np
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  from sklearn.preprocessing import MinMaxScaler
5  import Model
6  import pywt
7  def read_data(filename):
8      data = pd.read_csv(filename, header=None)
9      X = data.values
10     X = X.reshape([len(X)*5, 1])
11     scaler = MinMaxScaler()
12     X = scaler.fit_transform(X)
13     return X
14 def read_label(filename):
15     label = pd.read_csv(filename, header = None)
16     y = label.values
17     return y
18 def get_data(bit_rate, distance):
19     i = 1
20     X = []
21     y = []
22     num_file = 40
23     while i < (num_file+1):
24         X_path = 'Lorentz_dataset/%scm/%sk/%s000_250_%s.csv' % (distance
    , bit_rate, bit_rate, i)
25         y_path = 'Lorentz_dataset/%scm/%sk/label_%s000_250_%s.csv' % (
    distance, bit_rate, bit_rate, i)
26         X_t = read_data(X_path)
27         y_t = read_label(y_path)
28         X = np.append(X, X_t)
29         y = np.append(y, y_t)
30         i+=1
31     X = X.reshape(num_file*250, 5)
32     y = y.reshape(-1)
33     return X, y
34 def get_data_related_bit(bit_rate, distance):
35     j = 1
36     X = []
37     y = []
38     num_file = 40
39     while j < (num_file+1):
40         data_path = 'Lorentz_dataset/%scm/%sk/%s000_250_%s.csv' % (
    distance, bit_rate, bit_rate, j)
41         y_path = 'Lorentz_dataset/%scm/%sk/label_%s000_250_%s.csv' % (
    distance, bit_rate, bit_rate, j)
42         data_t = read_data(data_path)
43         y_t = read_label(y_path)
44         data_t = data_t.reshape([250, 5])
45         y_t = y_t.reshape(-1)
46         data = []
47         for i in np.arange(len(data_t)):
```

```
48              if i == 0:
49                  temp_1 = data_t[len(data_t)-2]
50                  temp_2 = data_t[len(data_t)-1]
51                  temp_3 = data_t[i]
52                  temp_4  = data_t[i+1]
53                  temp_5  = data_t[i+2]
54                  temp_6 = np.hstack([temp_1, temp_2, temp_3, temp_4,
    temp_5])
55              elif i == 1:
56                  temp_1 = data_t[len(data_t)-1]
57                  temp_2 = data_t[len(data_t)-1]
58                  temp_3 = data_t[i]
59                  temp_4 = data_t[i+1]
60                  temp_5 = data_t[i+2]
61                  temp_6 = np.hstack([temp_1, temp_2, temp_3, temp_4,
    temp_5])
62              elif i == len(data_t)-2:
63                  temp_1 = data_t[i-2]
64                  temp_2 = data_t[i-1]
65                  temp_3 = data_t[i]
66                  temp_4 = data_t[i+1]
67                  temp_5 = data_t[0]
68                  temp_6 = np.hstack([temp_1, temp_2, temp_3, temp_4,
    temp_5])
69              elif i == len(data_t)-1:
70                  temp_1 = data_t[i-2]
71                  temp_2 = data_t[i-1]
72                  temp_3 = data_t[i]
73                  temp_4 = data_t[0]
74                  temp_5 = data_t[1]
75                  temp_6 = np.hstack([temp_1, temp_2, temp_3, temp_4,
    temp_5])
76              else:
77                  temp_1 = data_t[i-2]
78                  temp_2 = data_t[i-1]
79                  temp_3 = data_t[i]
80                  temp_4 = data_t[i+1]
81                  temp_5 = data_t[i+2]
82                  temp_6 = np.hstack([temp_1, temp_2, temp_3, temp_4,
    temp_5])
83              data = np.append(data, temp_6)
84          X = np.append(X, data)
85          y = np.append(y, y_t)
86          j+=1
87      X = X.reshape(num_file*250, 25)
88      return X, y
89  def CWT(X):
90      cwt = []
91      for i in X:
92          coef, freqs = pywt.cwt(i, np.arange(1, 4), 'mexh')
93          cwt = np.append(cwt, coef)
94      cwt = cwt.reshape(10000, X.shape[0]*3)
95      return cwt
96  def exe(mode, bit_rate, distance):
97      if mode == "related_bit":
98          X, y = get_data_related_bit(bit_rate, distance)
99      elif mode == "DAE":
100         X, y = get_data(bit_rate, distance)
```

```
101         m = Model.DAE(bit_rate, X, y)
102         X = m.correct()
103     else:
104         X, y = get_data(bit_rate, distance)
105     filename_1 = "Lorentz_dataset/Lorentz_%scm_%sk_related_bit.csv" % (
    distance, bit_rate)
106     filename_2 = "Lorentz_dataset/Lorentz_label_%scm_%sk.csv" % (
    distance, bit_rate)
107     np.savetxt(filename_1, X, delimiter=",")
108     np.savetxt(filename_2, y, delimiter=",")
109     return X, y
```